Skalierbares und robustes Routing auf Internet-ähnlichen Topologien

zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Dipl.-Inform. Christoph Werle

aus Bad Mergentheim

Tag der mündlichen Prüfung: 9. Mai 2014

Erste Gutachterin: Prof. Dr. rer. nat. Martina Zitterbart
Karlsruher Institut für Technologie (KIT)

Zweiter Gutachter: Prof. Dr.-Ing. Georg Carle
Technische Universität München (TUM)

DOI: 10.5445/IR/1000049128

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

KAPITEL 1

Einleitung

Das Internet ist seit seinen Anfängen um 1970, als es nur eine handvoll von Forschungseinrichtungen verband, zu einem weltweiten Kommunikationssystem gewachsen, dessen Bedeutung für unser soziales und wirtschaftliches Leben nicht überschätzt werden kann und das stetig weiterwächst. Damit einhergehend wachsen auch die Routing-Tabellen und die daraus bestimmten Weiterleitungs-Tabellen, die die Router des Internets zur Weiterleitung von Paketen benötigen. Um den Aufbau und das Speichern von Routing-Tabellen sowie das Nachschlagen in den Routing-Tabellen zu bewältigen, benötigen Router sehr teure Hardware-Komponenten und haben einen hohen Energieverbrauch. Wie auch das Internet Architecture Board festhält, ist die Skalierbarkeit des Routings ein zentrales Problem, dem sich das heutige Internet stellen muss [1]. Während es durch kontinuierliche Aktualisierung der Hardware zwar vermutlich möglich ist, mit dem Wachstum der Routing-Tabellen Schritt zu halten, besteht die Befürchtung, dass dies steigende Kosten zur Folge hat.

Für die heute üblicherweise verwendete Weiterleitung von Paketen entlang optimaler Pfade ist im Allgemeinen ein Routing-Tabelleneintrag pro Ziel, also beispielsweise ein Eintrag pro IP-Präfix, notwendig. Die Routing-Tabellen wachsen daher linear mit der Anzahl der Ziele. Für ein skalierbares Routing sind jedoch sublinear wachsende Routing-Tabellen erforderlich. Dies bedeutet, dass es nicht mehr möglich ist, für jedes Ziel die optimale Möglichkeit zur Weiterleitung in den Routing-Tabellen zu speichern. Als Konsequenz daraus kann es zu einer Verschlechterung der durch das Routing bestimmten Pfade gegenüber den optimalen Pfaden kommen, die als Stretch, also als Dehnung oder Verlängerung, bezeichnet wird. Bei der Entwicklung von skalierbaren Routing-Protokollen muss daher abgewägt werden zwischen der Größe der Routing-Tabellen und dem resultierenden Stretch. Vor dem Hintergrund der eingeschränkten Skalierbarkeit des heutigen Routings ist die Entwicklung und Untersuchung solcher skalierbarer Routing-Protokolle wichtig.

Während diese neuen Ansätze für skalierbares Routing nicht notwendigerweise mit dem heutigen Internet kompatibel sind, gibt es mit der Netzvirtualisierung ein mögliches Einsatz-

Szenario. Netzvirtualisierung ermöglicht, als logische Erweiterung der Systemvirtualisierung auf Netze, den parallelen Betrieb virtueller Netze auf Basis einer gemeinsamen Infastruktur und ist ein fundamentales Konzept für das künftige Internet. In virtuellen Netzen können neuartige, auf bestimmte Anwendungsszenarien zugeschnittene Netzwerkarchitekturen eingesetzt werden. Dabei kann unter anderem das eingesetzte Routing-Protokoll frei gewählt werden. Dadurch lassen sich dort neben existierenden auch neuartige skalierbare Routing-Protokolle verwenden. Vor diesem Hintergrund und der Tatsache, dass Routing entlang optimaler Pfade nicht skalierbar ist, ist die Entwicklung und Untersuchung von neuartigen, skalierbaren Routing-Protokollen eine wichtige Herausforderung.

1.1 Problemstellung

Der Trade-Off zwischen der Größe der Routing-Tabellen und dem resultierenden Stretch wird im Forschungsbereich kompakter Routing-Verfahren formalisiert. Kompakte Routing-Verfahren erreichen bei skalierbarer, beschränkter Routing-Tabellengröße eine gleichzeitige Beschränkung des maximalen Stretch, die in vielen Fällen wünschenswert ist. Ein Problem der kompakten Routing-Verfahren ist allerdings, dass diese in der Regel eine zentrale Entität mit einer konsistenten globalen Netzsicht annehmen und nicht für den Einsatz als verteiltes Routing-Protokoll entworfen wurden. Während kompakte Routing-Verfahren also durch die Beschränkung der Routing-Tabellengröße und des maximalen Stretch sehr positive Eigenschaften besitzen, sind diese nicht ohne weiteres als verteiltes, adaptives Routing-Protokoll umsetzbar. Dies ist aber eine Voraussetzung für den Einsatz in einem verteilten Netz.

Ein kompaktes Routing-Verfahren von Brady und Cowen ist für eine verteilte Realisierung besonders interessant, da es für den Einsatz auf Internet-ähnlichen Topologien entwickelt wurde und eine konfigurierbare Beschränkung des Stretch ermöglicht [2]. Die erste Fragestellung **V**, die im Rahmen dieser Arbeit behandelt wird, ist daher die **verteilte Realisierung** dieses Verfahrens:

V	Kann das zentralisierte Routing-Verfahren von Brady und Cowen als verteiltes Routing-Protokoll umgesetzt werden?

Viele skalierbare Routing-Protokolle basieren, wie auch das kompakte Routing-Verfahren von Brady und Cowen, auf der Konstruktion einiger weniger Spannbäume und der Vergabe lokationsabhängiger Adressen innerhalb dieser Bäume. Zur Weiterleitung von Paketen wird nach dem Greedy-Prinzip der Nachbar ausgewählt, der die Distanz zum Ziel minimiert.

Ein Problem dieser Klasse spannbaum-basierter gieriger Routing-Verfahren ist, dass bei Ausfällen von Knoten oder Kanten eine aufwändige Reparatur notwendig wird. So kann es bereits durch den Ausfall eines einzelnen Knotens erforderlich sein, dass beinahe alle Knoten im Netz die Ihnen zugewiesene lokationsabhängige Adresse neu bestimmen müssen. In einer der detailliertesten Studien von Ausfällen in Netzen im Produktiveinsatz [3] wurde auf der Topologie eines amerikanischen Internetanbieters die Beobachtung gemacht, dass viele Ausfälle Kanten betreffen und diese Ausfälle meist nur von kurzer Dauer sind. Vor

dem Hintergrund dieser beiden Beobachtung stellt sich die zweite im Rahmen dieser Arbeit behandelte Frage **R** zur **Robustheit** folgendermaßen:

> **R** | Wie können spannbaum-basierte gierige Routing-Protokolle zur Vermeidung einer aufwändigen Reparatur robuster gestaltet werden, so dass die Auswirkungen von kurzlebigen Ausfällen auf die Qualität des Routings, gemessen durch die Anzahl zustellbarer Pakete, möglichst gering sind?

Diese beiden übergeordneten Fragestellungen werden in den entsprechenden Kapiteln weiter verfeinert und entsprechende Lösungen dafür entwickelt.

1.2 Beiträge und Gliederung

Die Beiträge dieser Arbeit orientieren sich an den zuvor eingeführten Fragestellungen und umfassen zunächst zwei vorbereitende Arbeiten.

Architektur für Netzvirtualisierung Ein mögliches Einsatz-Szenario für die hier entwickelten Routing-Ansätze sind virtuelle Netze, die den Betrieb neuartiger Netzarchitekturen und Protokolle ermöglichen. Dafür wurde eine flexible provider-übergreifende Architektur für Netzvirtualisierung vorgeschlagen, die virtuelle Netze als fundamentales Konzept eines künftigen Internets betrachtet und unter Berücksichtigung wirtschaftlicher Aspekte die notwendigen Rollen und Schnittstellen herausgearbeitet.

Simulationsumgebung Da für die Durchführung der im Rahmen dieser Arbeit erforderlichen Untersuchungen keine geeignete Simulationsumgebung zur Verfügung stand, wurde mit RoutingSim ein skalierbarer Simulator zur Untersuchung von Routing-Protokollen entwickelt und als Open Source Software veröffentlicht.

Als Lösungen für die übergeordneten wissenschaftlichen Fragestellungen wurden die folgenden Beiträge geleistet:

Sprinkles Mit Sprinkles wurde, basierend auf dem kompakten Routing-Verfahren von Brady und Cowen, ein verteiltes Routing-Protokoll entwickelt, das dessen parametrisierbare Stretch-Schranke umsetzt. Da sich die Länge der lokationsabhängig vergebenen Adressen als Engpass herausstellte, wurde mit dem Sparse Mode ein Ansatz entwickelt, der die resultierenden Adresslängen um mehr als die Hälfte reduzieren kann und es dadurch ermöglicht, niedrigere Stretch-Schranken zu verwenden.

Greedy Failure-Carrying Packets Zur Erhöhung der Robustheit spannbaum-basierter gieriger Routing-Verfahren wurde mit Greedy Failure-Carrying Packets eine Lösung entwickelt, die einen Verzicht auf die aufwändige Reparatur nach Knoten- oder Kantenausfällen ermöglicht. Die Erweiterung eines spannbaum-basierten gierigen Routing-Protokolls mit Greedy Failure-Carrying Packets erhöht insbesondere bei den praxisrelevanten Kantenausfällen die Zustellrate deutlich. Gleichzeitig wird im Vergleich zu verwandten Arbeiten das Netz entlastet, da Greedy Failure-Carrying Packets Pakete auf kurzen Wegen um Fehlerstellen herum zustellt oder diese schnell verwirft.

1.3 Aufbau der Arbeit

Im folgenden Kapitel 2 werden zunächst die für das Verständnis der Arbeit notwendigen Grundlagen erarbeitet. Dazu werden Ansätze für skalierbares Routing und die erforderlichen Grundlagen aus der Graphentheorie vorgestellt. Kapitel 3 stellt eine provider-übergreifende Architektur für Netzvirtualisierung vor, auf deren Basis beispielsweise neue Ansätze für Routing, wie die hier entwickelten, ausgebracht werden können. Zur Untersuchung der Ansätze wurde mit dem in Kapitel 4 vorgestellten RoutingSim eine leichtgewichte und skalierbare Simulationsumgebung zur Untersuchung von Routing-Protokollen entwickelt. Als neuer verteilter Ansatz für skalierbares Routing bei beschränktem Stretch wird in Kapitel 5 Sprinkles vorgestellt. Da sowohl Sprinkles als auch verwandte Routing-Protokolle nur schlecht mit Knoten- und Kantenausfällen umgehen können, wurde anschließend mit dem in Kapitel 6 beschriebenen Greedy Failure-Carrying Packets eine neuartige Rerouting-Strategie entwickelt, die in Gegenwart inkonsistenter Routing-Tabellen eine deutliche Erhöhung der Zustellrate ermöglicht und im Vergleich zu alternativen Ansätzen das Netz weniger belastet. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick auf mögliche weiterführende Arbeiten in Kapitel 7.

1.4 Zugrundeliegende Veröffentlichungen

Die in dieser Dissertation vorgestellten Konzepte und Verfahren basieren auf Veröffentlichungen auf internationalen Konferenzen und Workshops. Im Rahmen dieser Dissertation wurden diese Beiträge erweitert und deutlich umfangreichere Evaluierungen durchgeführt als es für die Publikationen aufgrund des zur Verfügung stehenden Platzes möglich war.

- Die in Kapitel 3 vorgestellte Architektur zur Virtualisierung von Netzen wurde auf dem im Rahmen der ACM SIGCOMM stattfindenden Workshop VISA (Virtualized Infastructure Systems and Architectures) veröffentlicht [4]. Eine Analyse von Signalisierungsaspekten in dieser Architektur wurde auf dem International Workshop on the Network of the Future publiziert, der im Rahmen der International Conference on Communications (ICC) stattfand [5].
- Die für die Evaluierungen verwendete Simulationsumgebung RoutingSim, die hier in Kapitel 4 beschrieben wird, wurde in Kooperation mit Sebastian Mies entwickelt und auch für Evaluierungen im Rahmen seiner Dissertation [6] verwendet. Die Veröffentlichung von RoutingSim erfolgte auf der International Conference on Networks (ICON) [7].
- Sprinkles wurde auf der Second International Conference on Future Generation Communication Technologies (FGCT) publiziert und erhielt dort auch eine Auszeichnung als beste Arbeit [8].
- Greedy Failure-Carrying Packets wurde auf der International Conference on Computing, Networking and Communications (ICNC) vorgestellt [9].

KAPITEL 2

Grundlagen

In diesem Kapitel werden zunächst einige Grundlagen für den nachfolgenden Teil der Arbeit vorgestellt. Da Netze im Rahmen dieser Arbeit, wie auch sonst üblich, als Graphen modelliert werden, stellt Abschnitt 2.1 zunächst die für das Verständnis der Arbeit notwendigen Grundlagen aus dem Bereich der Graphentheorie dar. Anschließend wird in Abschnitt 2.2 zunächst eine Definition von Routing und eine Klassifikation verschiedener Routing-Verfahren eingeführt. Als prominente und für diese Arbeit relevante Vertreter im Kontext von skalierbarem Routing werden das kompakte Routing und Routing nach dem Greedy-Prinzip eingeführt und bedeutende Vertreter vorgestellt. Abschließend fasst Abschnitt 2.3 die in den Grundlagen eingeführten Notationen in Form einer Tabelle als Referenz zusammen.

2.1 Graphentheoretische Grundlagen

Kommunikationsnetze werden zur formalen Betrachtung üblicherweise als Graphen modelliert. Die Router eines Netzes werden dabei zu den Knoten eines Graphen und die Verbindungen zwischen Routern zu den Kanten des Graphen, über die Nachrichten versendet werden können. Die Modellierung des Netzes erfolgt hierbei mit Fokus auf den für das Routing relevanten Kern des Netzes. Endsysteme, die sich beispielsweise in einem lokalen Netz hinter einem der Router befinden, können über einen zusätzlich in Paketen mitzuführenden Endpunkt-Identifikator angesprochen werden, wie dies auch beim Locator-Identifier Separation Protocol (LISP) [10] der Fall ist. In dieser Arbeit wird der Fokus aber, wie eingangs erwähnt, auf die Routing-Infrastruktur gelegt.

Im folgenden Abschnitt 2.1.1 werden zunächst grundlegende graphentheoretische Definitionen eingeführt. Anschließend werden mit den Distanz-Labels (siehe Abschnitt 2.1.2) und Einbettungen (siehe Abschnitt 2.1.3) grundlegende Mechanismen für den Entwurf von Routing-Verfahren vorgestellt. In Abschnitt 2.1.4 werden abschließend die hier besonders

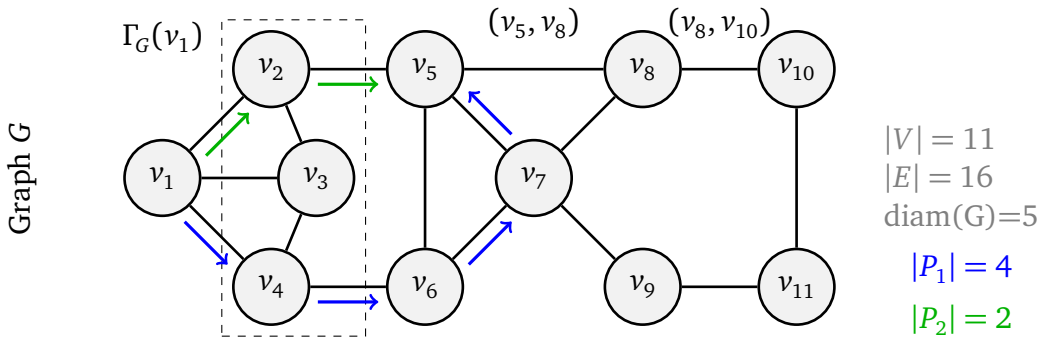


Abbildung 2.1: Beispielgraph zur Veranschaulichung der graphentheoretischen Definitionen

im Fokus stehenden Internet-ähnlichen Topologien vorgestellt und deren Eigenschaften diskutiert.

2.1.1 Definitionen

Zur formalen Betrachtung von Netzen werden die Router als Knoten V eines Graphen G dargestellt und die Verbindungen als Kanten E des Graphen G modelliert. Der **Graph** $G = (V, E)$ besteht dann aus $n := |V|$ **Knoten V** und $m := |E|$ **Kanten E**. Die Begriffe Knoten und Router beziehungsweise Kante und Verbindung bzw. Link werden im Folgenden synonym verwendet. Eine Kante e zwischen zwei Knoten v_i und v_j wird auch notiert als $e = (v_i, v_j)$. Eine Funktion $w : E \rightarrow \mathbb{R}^+$ weist jeder Kante $e \in E$ ein **Gewicht** zu, das als Kostenmetrik für diese Kante dient. Im Folgenden werden in Beispielen in der Regel ungewichtete Graphen betrachtet und damit jeder Kante $e_i \in E$ ein Einheitsgewicht $w(e_i) = w_i = 1$ zugewiesen.

Den Knoten V eines Graphen wird durch eine Abbildung $ID : V \rightarrow S$ jeweils ein eindeutiger **Identifikator** aus $S = \{s_1, s_2, ..., s_n\}$, beispielsweise eine Ganzzahl, zugewiesen. Die Menge S ist dabei eine geordnete Menge, wobei jeweils $s_i < s_{i+1}$ ist. Diese Identifikatoren können durch ihre Eindeutigkeit und Ordnung verwendet werden, um beispielsweise eine eindeutige Auswahl aus einer Menge von Knoten zu treffen. Die formale Unterscheidung zwischen einem Knoten v_i und dessen Identifikator s_i wird im Folgenden nur getroffen, wenn die Bedeutung aus dem Kontext nicht eindeutig hervorgeht.

Abbildung 2.1 zeigt einen Graphen, der aus 11 Knoten $\{v_1, v_2, ..., v_{11}\}$ und 16 Kanten besteht. Als Beispiele für die Notation einer Kante sind die Kanten (v_5, v_8) und (v_8, v_{10}) entsprechend beschriftet.

Die **Nachbarschaft** $\Gamma_G(v)$ eines Knotens v besteht aus allen Knoten, die mit diesem Knoten in G direkt über eine Kante verbunden sind. Im obigen Beispiel besteht die Nachbarschaft $\Gamma_G(v_1)$ des Knotens v_1 aus den Knoten v_2, v_3 und v_4. Die Anzahl von Knoten in der Nachbarschaft eines Knotens v bezeichnet man auch als den **Grad** $\deg_G(v)$ eines Knotens. Damit ist $\deg_G(v) = |\Gamma_G(v)|$. Für den **Knoten mit dem höchsten Grad** eines Graphen wird der Bezeichner h eingeführt, da diesem in einigen Routing-Verfahren eine besondere Rolle zufällt. Gegebenfalls auftretende Mehrdeutigkeiten werden über den Identifikator der Knoten aufgelöst und h damit eindeutig bestimmt. In Abbildung 2.1 ist der höchste Knotengrad

4 und dies sowohl für Knoten v_5 als auch für Knoten v_7. Der Knoten mit dem kleineren Identifikator v_5 wird dann auch mit h bezeichnet.

Ein **Pfad** ist eine endliche Folge von paarweise adjazenten Kanten $(e_1, e_2, ..., e_k)$. Die **Länge eines Pfades** P von einem Startknoten v_i zu einem Endknoten v_j ist dann bestimmt durch $|P| = \sum_{p=1}^{k} w(e_p)$. Werden Einheitskantengewichte verwendet, ist die Länge des Pfades durch die Anzahl der Kanten des Pfades $|P| = \sum_{p=1}^{k} 1 = k$ bestimmt. Im hier üblicherweise betrachteten Fall von Einheitskantengewichten hat der in Abbildung 2.1 blau dargestellte Pfad $P_1 = (\,(v_1, v_4), (v_4, v_6), (v_6, v_7), (v_7, v_5)\,)$ zwischen den Knoten v_1 und v_5 die Länge 4.

Die **Distanzfunktion** $d_G : V \times V \to \mathbb{R}_0^+$ ordnet zwei Knoten $v_i, v_j \in V$ deren Distanz zu. Die Distanz $d_G(v_i, v_j)$ zwischen zwei Knoten v_i und v_j ist die Länge eines kürzesten Pfades zwischen diesen im Graphen G. Der kürzeste Pfad ist dabei nicht notwendigerweise eindeutig bestimmt. Der in dem vorliegenden Beispiel grün dargestellte, eindeutig bestimmte kürzeste Pfad zwischen den Knoten v_1 und v_5 ist der Pfad $P_2 = (\,(v_1, v_2), (v_2, v_5)\,)$. Dieser besitzt die Länge $|P_2| = d_G(v_1, v_5) = 2$.

Eine oft verwendete Kennzahl für Graphen ist der Durchmesser. Der **Durchmesser** diam_G eines Graphen G ist definiert als die Länge des längsten kürzesten Pfades. In dem Beispiel in Abbildung 2.1 ist der Durchmesser 5; dies ist beispielsweise die Länge eines kürzesten Pfades zwischen den Knoten v_1 und v_{11}.

Eine Metrik zur Bestimmung der Qualität eines Pfades ist der sogenannte **Stretch**, der die Verlängerung eines Pfades gegenüber dem jeweils kürzesten Pfad betrachtet. Der Stretch kann dabei multiplikativ oder additiv definiert werden: Für einen Pfad P zwischen zwei Knoten v_i, v_j ist der **multiplikative Stretch** definiert als

$$s_m = \frac{|P|}{d_G(v_i, v_j)}$$

Der Stretch des zuvor definierten Pfades P_1 zwischen den Knoten v_1 und v_5 ist damit

$$\frac{|P_1|}{d_G(v_1, v_5)} = \frac{|P_1|}{|P_2|} = \frac{4}{2} = 2.$$

Der **additive Stretch** für einen Pfad P zwischen zwei Knoten v_i, v_j ist definiert als

$$s_a = |P| - d_G(v_i, v_j)$$

Damit ergibt sich für den beispielhaft betrachteten Pfad P_1 zwischen den Knoten v_1 und v_5 ein additiver Stretch von

$$|P_1| - d_G(v_1, v_5) = |P_1| - |P_2| = 4 - 2 = 2$$

Während in diesem Beispiel zufälligerweise $s_m = s_a = 2$ ist, so ist der Unterschied zwischen beiden Formulierungen dennoch signifikant: Ein additiver Stretch von 2 bedeutet eine Verlängerung eines Pfades um 2 Übertragungsabschnitte, wohingegen ein multiplikativer Stretch von 2 eine Verdopplung der Pfadlänge bedeutet. Ist aus dem Kontext nicht eindeutig ersichtlich, welche Art des Stretch diskutiert wird, so wird der additive Stretch explizit mit s_a und der multiplikative Stretch mit s_m bezeichnet.

Ein Graph $G = (V, E)$ heißt **zusammenhängend**, wenn es zwischen jeweils zwei Knoten aus V einen Pfad in G gibt. Andernfalls ist G unzusammenhängend und besteht damit aus mehreren disjunkten Teilgraphen. Ein Teilgraph eines Graphen $G = (V, E)$ ist ein Graph $G' = (V', E')$, für den $V' \subseteq V$ und $E' \subseteq E$ gilt. Damit eine Kante $e = (v_i, v_j) \in E'$ Teil von G' ist, müssen die Knoten v_i und v_j Bestandteil von V' sein. Da der Zusammenhang eine Äquivalenzrelation ist, wird ein Graph dadurch eindeutig in maximal zusammenhängende Komponenten zerlegt, die **Zusammenhangskomponenten** genannt werden. Die Zusammenhangskomponente mit der größten Anzahl an Knoten wird im folgenden auch **größte Zusammenhangskomponente** genannt.

Der in den eingeführten Notationen enthaltene Bezug auf den jeweils betrachteten Graphen G entfällt, wenn G der gesamte Graph ist. Der Grad eines Knotens v kann mit dieser Konvention statt durch $\deg_G(v)$ als $\deg(v)$ notiert werden. Werden hingegen Teilgraphen von G betrachtet, wird der Bezeichner für den Teilgraphen in der Notation mitgeführt.

2.1.2 Distanz-Labels

Distanz-Labeling wird beispielsweise als Baustein in speichereffizienten Routing-Verfahren eingesetzt. Beispielhaft wird dazu hier das Konzept hinter Distanz-Labels sowie ein Verfahren zur Bestimmung und Verwendung von Distanz-Labels vorgestellt.

Das Ziel von Labeling-Verfahren im Allgemeinen ist es, Knoten eines Netzwerks derart mit Labels zu versehen, dass nur anhand der Labels zweier Knoten eine darin kodierte Information über die Beziehung der Knoten wiedergewonnen werden kann [11]. Im Falle von Distanz-Labels [12] kann man die Distanz zweier Knoten anhand derer Labels rekonstruieren. Dabei unterscheidet man zunächst zwischen Approximationsverfahren, die für eine kompaktere Speicherung der Labels eine gewisse Abweichung von der tatsächlichen Distanz in Kauf nehmen und sogenannten exakten Distanz-Labels. Im Folgenden werden nur exakte Distanz-Labels betrachtet und abschließend ein konkreter Algorithmus zur Konstruktion von exakten Distanz-Labels beispielhaft vorgestellt. Dieser vorgestellte Algorithmus ist Bestandteil eines später vorgestellten Routing-Verfahrens (siehe Abschnitt 2.2.1.5).

Das Verfahren zur Bestimmung der Distanz-Labels nennt man **Markierungsalgorithmus**; das Verfahren zur Bestimmung der Distanz aus zwei gegebenen Adressen ist der **Dekodieralgorithmus**. Für die Anwendung in Routing-Verfahren sind die Distanz-Labels üblicherweise auch Bestandteil des Paketkopfs. Daher ist es wichtig, dass die Distanz-Labels möglichst kurz sind. Für Bäume existieren dabei Verfahren mit einer Label-Länge in $O(\log^2 n)$ [13]. Dies ist vom Platzbedarf im Paketkopf realistisch, da in einem n-Knoten Netzwerk ja bereits die eindeutige Kodierung des Zielknotens einen Speicherbedarf von $O(\log n)$ benötigt. Das nachfolgend anhand eines Beispiels beschriebene exakte Distanz-Labeling Verfahren für Bäume [12] erzeugt dabei Distanz-Labels in $O(\log^2 n)$; dies wird dadurch ermöglicht, dass der Markierungsalgorithmus zentralisiert arbeitet und den vollständigen Baum als Eingabe erhält.

Abbildung 2.2 zeigt beispielhaft einen Baum, der nach diesem Verfahren mit Distanz-Labels versehen wurde. Durch eine rekursive Zerlegung des Baums über sogenannte Tree Separator wird dieser immer mindestens halbiert und auf jeder Separator-Ebene ein Teil-Label erzeugt.

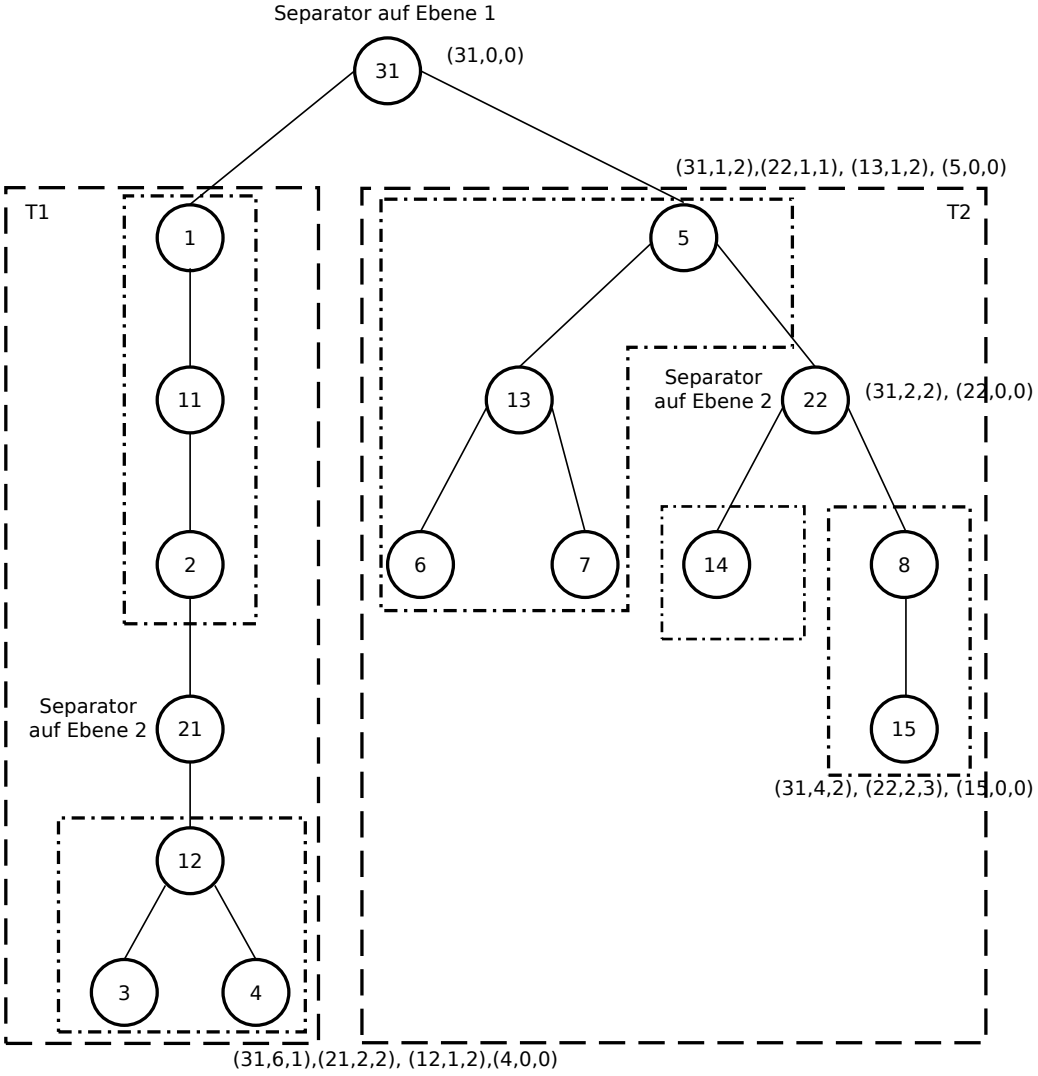


Abbildung 2.2: Beispiel für ein Distanz-Labeling Verfahren

Ein Tree Separator ist ein Knoten mit der Eigenschaft, dass ein Entfernen des Tree Separators aus dem Baum den Baum in Teilbäume zerlegt, die maximal halb so viele Knoten umfassen wie der ursprüngliche Baum. Ausgehend von einem Baum mit n Knoten entstehen durch das Entfernen des Separators also mindestens zwei Teilbäume mit maximal $n/2$ Knoten. Jeder der so entstandenen Teilbäume wird wiederum durch einen Tree Separator zerlegt bis nur noch ein einzelner Knoten verbleibt.

Bei jeder Teilung eines Baumes durch einen Tree Separator wird jedem Knoten in einem der Teilbäume ein Distanz-Label bezüglich dieses Tree Separators zugewiesen. Diese Distanz-Label bestehen aus drei Komponenten:

- dem Identifikator des Tree Separators
- der Distanz des Knotens zu diesem Tree Separator
- und einem eindeutigen Identifikator für den Teilbaum, in dem sich der Knoten befindet.

Das im Beispiel durch den Markierungsalgorithmus vergebene Distanz-Label von Knoten 5 bedeutet dann beispielsweise:

- (31,1,2): Knoten 5 ist in Teilbaum 2 von Separator 31 aus und einen Übertragungsabschnitt von diesem entfernt (Ebene 1).
- (22,1,1): Knoten 5 befindet sich in Teilbaum 1 von Separator 22, der Teilbaum T_2 zerlegt, aus und ist einen Hop von Separator 22 entfernt (Ebene 2).
- (13,1,2): Knoten 5 ist in Teilbaum 2 von Separator 13 aus und einen Übertragungsabschnitt von diesem entfernt (Ebene 3).
- (5,0,0): Knoten 5 selbst (Ebene 4).

Der Dekodierungsalgorithmus zur Bestimmung der Distanz zwischen zwei Knoten ermittelt anhand derer Distanz-Labels zunächst den Tree Separator auf der höchsten Ebene, den beide Distanz-Labels enthalten. Der Tree Separator auf dieser Ebene liegt dann zwischen den beiden Knoten und die Summe der in den Distanz Labels enthaltenen Distanzen zu dem gemeinsamen Tree Separator ist die Distanz zwischen den beiden Knoten.

Für die Knoten 4 und 5 ist gemeinsame Tree Separator auf der höchsten Ebene Knoten 31 und die Distanz zwischen den Knoten damit $6+1=7$. Für die Knoten 5 und 15 ist Knoten 22 der Tree Separator auf der höchsten Ebene und die Distanz der Knoten damit $1+2=3$.

Zusammenfassend ist es also möglich, Distanz-Labels zu konstruieren, die mit einer Länge von $O(\log^2 n)$ für einen Einsatz in Paketköpfen geeignet sind. Anhand der Distanz-Labels zweier Knoten ist es mit einem Dekodierungsalgorithmus möglich, die Distanz zwischen diesen Knoten zu bestimmen. Ein Problem des hier skizzierten Verfahrens mit einer optimalen Label-Länge, das auch für darauf basierende Routing-Verfahren gilt, ist jedoch, dass der Markierungsalgorithmus durch die rekursive Zerlegung der Bäume anhand der Tree Separator nicht ohne weiteres in einem verteilten Protokoll umgesetzt werden kann.

2.1.3 Einbettungen

Ein weiteres zentrales Konzept für die Entwicklung der hier betrachteten Routing-Verfahren sind sogenannte Einbettungen. Dazu werden zunächst metrische Einbettungen, also Abbildung von einem metrischen Raum auf einen weiteren metrischen Raum betrachtet. An-

schließend werden die für das Routing nach dem Greedy-Prinzip interessanten gierigen Einbettungen vorgestellt.

Ein **metrischer Raum** ist ein Paar (X, d), wobei X eine Menge und d eine Metrik auf dieser Menge ist. Die Metrik d ist eine symmetrische, nicht-negative Abbildung $d : X \times X \to \mathbb{R}_0^+$, die die Dreiecks-Ungleichung für drei beliebige Knoten $x, y, z \in X$ erfüllt:

$$d(x, y) + d(y, z) \geq d(x, z).$$

Ein ungerichteter Graph $G = (V, E)$ mit der Distanzfunktion d induziert einen metrischen Raum (V, d), der durch die Distanzmatrix repräsentiert werden kann. Für die in Abbildung 2.1 dargestellte Topologie ergibt sich mit Einheitsdistanzen beispielsweise die folgende Distanzmatrix:

$$M = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \end{array} \begin{array}{c} \begin{array}{ccccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} \end{array} \\ \left(\begin{array}{ccccccccccc} 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 & 4 & 5 \\ 1 & 0 & 1 & 2 & 1 & 2 & 2 & 2 & 3 & 3 & 4 \\ 1 & 1 & 0 & 1 & 2 & 2 & 3 & 3 & 4 & 4 & 5 \\ 1 & 2 & 1 & 0 & 2 & 1 & 2 & 3 & 3 & 4 & 4 \\ 2 & 1 & 2 & 2 & 0 & 1 & 1 & 1 & 2 & 2 & 3 \\ 2 & 2 & 2 & 1 & 1 & 0 & 1 & 2 & 2 & 3 & 3 \\ 3 & 2 & 3 & 2 & 1 & 1 & 0 & 1 & 1 & 2 & 2 \\ 3 & 2 & 3 & 3 & 1 & 2 & 1 & 0 & 2 & 1 & 2 \\ 4 & 3 & 4 & 3 & 2 & 2 & 1 & 2 & 0 & 2 & 1 \\ 4 & 3 & 4 & 4 & 2 & 3 & 2 & 1 & 2 & 0 & 1 \\ 5 & 4 & 5 & 4 & 3 & 3 & 2 & 2 & 1 & 1 & 0 \end{array} \right) \end{array}$$

Anhand dieser Matrix M können die drei Eigenschaften eines metrischen Raumes, also Symmetrie, Nicht-Negativität und die Dreiecksungleichung, leicht überprüft werden.

Für zwei metrische Räume (X, d_x) und (Y, d_y) ist eine **metrische Einbettung** eine Abbildung $f : X \to Y$, so dass für eine Skalierungskonstante $c > 0$ und für alle $x_i, x_j \in X$ gilt:

$$c \cdot d_x(x_i, x_j) \leq d_y(f(x_i), f(x_j)) \leq c \cdot D \cdot d_x(x_i, x_j)$$

D ist die sogenannte **Verzerrung** (englisch: Distortion). Für eine Einbettung $f : X \to Y$ ist die Verzerrung das kleinstmögliche D, das die vorangegangene Gleichung für eine Skalierungskonstante c erfüllt. Im Fall $D = 1$ erhält die Einbettung die Distanzen und wird auch **isometrische, also distanzerhaltende Einbettung** genannt. Ein Beispiel für eine solche isometrische Einbettung von Bäumen wird im nachfolgenden Abschnitt 2.2.2.2 vorgestellt.

Ausgehend von einem durch einen Graphen $G = (V, E)$ mit der Distanzmetrik d induzierten metrischen Raum sind dabei Einbettungen f von Interesse, die den Knoten $v \in V$ ein Bild $f(v)$ zuordnen, das platzeffizient, beispielsweise zur Verwendung in einem Paketkopf, dargestellt werden kann. Wie auch bereits bei den Distanz-Labels ist dies wünschenswert, da das Bild $f(v)$, das von nun an auch **virtuelle Koordinate** genannt wird, beispielsweise als Adresse oder Teil einer Adresse im Paketkopf mitgeführt werden muss.

Die **gierige Einbettung (englisch: greedy embedding)** [14] eines Graphen $G = (V, E)$ mit der Distanzfunktion d in einen metrischen Raum (X, d_x) ist eine Abbildung $f : V \to X$, so

dass für alle $x, y \in V; x \neq y$ gilt:

$$\exists z \in \Gamma(x) : d_x(f(z), f(y)) \leq d_x(f(x), f(y))$$

Dies bedeutet auf die spätere Anwendung in Routing-Verfahren bezogen, dass es für einen Knoten x, der ein Paket zu einem Knoten y weiterleiten soll, **immer** einen Nachbarn z von x gibt, über den das Paket in Richtung des Ziels weitergeleitet werden kann und damit durch jede Weiterleitung ein Fortschritt in Richtung des Ziels erzielt wird. Es gibt also immer einen Pfad zum Ziel und alle Pakete, die entlang einer gierigen Einbettung weitergeleitet werden, werden auch zugestellt.

Ein Verfahren zur Konstruktion einer solchen gierigen Einbettung von Spannbäumen wird in Abschnitt 2.2.2.2 vorgestellt. Spannbäume werden bei Einbettungen in der Praxis oft bevorzugt, da diese leichter einzubetten sind als allgemeine Graphen.

2.1.4 Internet-ähnliche Topologien

Im Verlauf der Arbeit werden drei Arten Internet-ähnlicher Topologien untersucht, die hier kurz vorgestellt werden. Dazu werden zunächst synthetisch erzeugbare Power-Law Topologien betrachtet. Die anschließend vorgestellten, aus Messungen des Internets gewonnenen Topologien modellieren das Internet auf Ebene autonomer Systeme beziehungsweise auf Ebene von Routern.

2.1.4.1 Synthetische Power-Law Topologien

Zur Untersuchung der im Rahmen dieser Arbeit entwickelten Ansätze werden unter anderem sogenannte Power-Law Random Graphs (PLRGs) eingesetzt. Diese Internet-ähnlichen Topologien können synthetisch mit Topologie-Generatoren erzeugt werden und ermöglichen über eine entsprechende Parametrisierung eine sehr breit angelegte Evaluierung der in dieser Arbeit entworfenen Verfahren.

Für Power-Law Graphen wird die folgende Definition [2] verwendet, die eine vereinfachte Formulierung von [15] darstellt. Ein Power-Law Graph $G = (V, E)$ ist ein ungerichteter, ungewichteter Graph, dessen Knotengradverteilung einem Potenzgesetz folgt. Für die Anzahl von Knoten y mit Grad k $y = |\{v \in V \mid \deg(v) = k\}|$ muss danach für eine Konstante $\beta \in \mathbb{R}^+$, den sogenannten Power-Law Parameter, gelten:

1. $y = \begin{cases} \lfloor c \rfloor - r & \text{für } k = 1 \\ \left\lfloor \frac{c}{k^\beta} \right\rfloor & \text{für } k = 2, \ldots, \left\lfloor c^{\frac{1}{\beta}} \right\rfloor \end{cases}$
2. $r = n - \sum_{k=1}^{\left\lfloor c^{\frac{1}{\beta}} \right\rfloor} \left\lfloor \frac{c}{k^\beta} \right\rfloor$
3. c minimiert unter den vorhergehenden Bedingungen $|n - r|$

Ein β-PLRG $G_\beta = (V, E)$, $|V| = n$, ist ein zufällig aus der Menge aller Power-Law Graphen mit Power-Law Parameter β und n Knoten ausgewählter Graph. Da diese Graphen nicht notwendigerweise verbunden sind, wird im Folgenden G_β mit der jeweils größten Zusammenhangskomponente des Graphen gleichgesetzt und entsprechend auch später so in den Simulationen verwendet.

Für diese Graphen wurden in Abhängigkeit vom Power-Law Parameter β die folgenden Eigenschaften gezeigt [15]:

1. Für alle β besitzt G_β eine eindeutige größte Zusammenhangskomponente während alle anderen Komponenten eine maximale Größe in $O(\log n)$ haben.
2. Für $0 < \beta < 2$ existiert in der größten Zusammenhangskomponente ein dichter Kern (englisch: Core) mit einem Durchmesser von maximal 3. Um den Kern herum befinden sich baum-ähnliche Fortsätze endlicher Länge.
3. Für $2 < \beta < 4$ besteht die größte Zusammenhangskomponente aus drei Schichten: einem dicht verbundenen Kern, einer Zwischen-Ebene und den baum-ähnlichen Fortsätzen, jeweils mit Durchmesser in $O(\log n)$
4. Für alle β enthält der Kern von G_β die Knoten mit dem höchsten Grad.

Diese Eigenschaften bieten sich einerseits an, um darauf basierend angepasste Routing-Protokolle zu entwickeln. Andererseits ist es durch die Variation des Parameters β einfach möglich, das Verhalten eines Protokolls auf einen breiten Spektrum von Power-Law Topologien zu untersuchen und in Abhängigkeit von β optimale Parametrisierungen zu finden. Daher werden diese Topologien später auch zur Evaluierung der entwickelten Ansätze verwendet. Ein für die Erzeugung solcher Topologien zur Verfügung stehender Topologie-Generator, der nach dem PLRG-Modell arbeitet und auch zur Erzeugung der hier verwendeten PLRG-Graphen verwendet wurde, ist unter `http://digg.cs.tufts.edu/` verfügbar.

2.1.4.2 DIMES AS-Level Topologie

Das Projekt DIMES [16] erforscht die Struktur und Topologie des Internets und platziert dazu sogenannte Agenten verteilt auf der ganzen Welt bei freiwilligen Helfern. Diese Agenten versenden Traceroute- und Ping-Pakete und liefern die resultierenden Ergebnisse an das Projekt zurück. Anhand der aggregierten Sicht wird dann die Topologie des Internets auf Ebene autonomer Systeme konstruiert und seit Dezember 2006 in monatlichen Abständen veröffentlicht.

Der Power-Law Parameter der DIMES AS-Level Topologie ist etwa $\beta = 2.12$ [17]. Dieses Ergebnis wird auch durch anderweitig erhobene Datensätze bestätigt: Für Skitter [18] wird von $\beta = 2.25$ berichtet, während durch das Projekt Routeviews erhobene Daten ein $\beta = 2.16$ ergeben [19].

2.1.4.3 CAIDA Router-Level Topologie

Die Cooperative Association for Internet Data Analysis (CAIDA) untersucht makroskopische Aspekte der Internet-Infrastruktur und erfasst dazu über aktive und passive Messungen Daten über das Routing im Internet. Basierend auf diesen Daten wurde unter anderem die Topologie des Internets auf Ebene der Router frei verfügbar gemacht[1]. Da diese Topologie auch in vielen anderen Arbeiten verwendet wird und damit eine gute Vergleichbarkeit der Ergebnisse ermöglicht, wird diese auch hier in der Evaluierung verwendet.

[1] `http://www.caida.org/data/router-adjacencies/itdk0304_rlinks_undirected.gz`

Diese CAIDA Topologie des Internets auf Router-Ebene besitzt einen Power-Law Exponenten von $\beta = 2.14$ [20].

2.2 Routing

Routing (englisch: Streckenplanung, Wegbestimmung) in einem Kommunikationsnetzwerk beschreibt die Bestimmung von Pfaden, entlang derer Pakete zwischen den Knoten des Netzwerkes zugestellt werden. Im Gegensatz zu leitungsvermittelten Systemen erfolgt die Weiterleitung von Paketen derart, dass Knoten anhand von Informationen im Paketkopf und ihrer lokalen Routing-Tabelle bestimmen, an welchen Knoten Pakete jeweils als nächstes weitergeleitet werden. Die Festlegung der Wege erfolgt durch die Bestimmung der Routing-Tabellen über einen Routing-Algorithmus. Routing-Algorithmen bestimmen anhand einer vom jeweiligen Algorithmus abhängigen Eingabe die Routing-Tabellen für alle Knoten des Netzes.

Dabei kann man zunächst unterscheiden, ob ein Routing-Algorithmus eine **globale** Sicht auf die Topologie benötigt, wie dies beispielsweise bei Link-State Routing-Protokollen und auch vielen Arbeiten aus dem Bereich des kompakten Routings (siehe Abschnitt 2.2.1) der Fall ist, oder ob ein Routing-Algorithmus auf einer **eingeschränkten** Sicht auf die Topologie arbeitet.

Die Ausführung von Routing-Algorithmen kann auf der einen Seite **zentralisiert** erfolgen. Dazu müssen die von einem Routing-Algorithmus als Eingabe erwarteten Daten auf einem zentralen System vorliegen, auf dem dann die Berechnung der Routing-Tabellen erfolgt [21, 22]. Die resultierenden Routing-Tabellen können dann anschließend auf die jeweiligen Knoten verteilt werden. Im Gegensatz zur zentralisierten Ausführung können Routing-Algorithmen auch **verteilt** auf allen Knoten eines Netzes ausgeführt werden. In diesem Fall ist sowohl der Zustand als auch die Kontrolle über die Knoten verteilt.

Während des Betriebs eines Netzes kann es zum Beispiel durch Änderung der Topologie oder veränderte Verkehrsmuster erforderlich werden, die Routing-Tabellen anzupassen. Im Gegensatz zu **statischem** Routing, bei dem die Routing-Tabellen nach eine initialen, beispielsweise administrativen Festlegung nicht angepasst werden, wird bei **dynamischem** Routing eine entsprechende Anpassung der Routing-Tabellen durchgeführt.

In der Praxis werden meist verteilte, dynamische Routing-Algorithmen eingesetzt, wobei Systeme initial nur über eine lokale Sicht verfügen. Dies bedeutet, dass ein Knoten anfangs nur seine direkte Nachbarschaft kennt und zunächst die als Eingabe für den Routing-Algorithmus erforderlichen Informationen akquirieren muss. Dieser Informationsaustausch wird über Routing-Protokolle definiert.

Abbildung 2.3 zeigt ein konzeptionelles Modell eines Routers. Routing-Protokolle tauschen im sogenannten **Kontrollpfad** Routing-Nachrichten aus, um die notwendigen Eingaben für den Routing-Algorithmus zu erhalten. Anhand der durch den Routing-Algorithmus bestimmten, lokalen Routing-Tabelle erfolgt dann die Weiterleitung eingehender Daten-Nachrichten im **Datenpfad**.

Das Routing im Internet basiert üblicherweise darauf, optimale (also z. B. kürzeste) Pfade

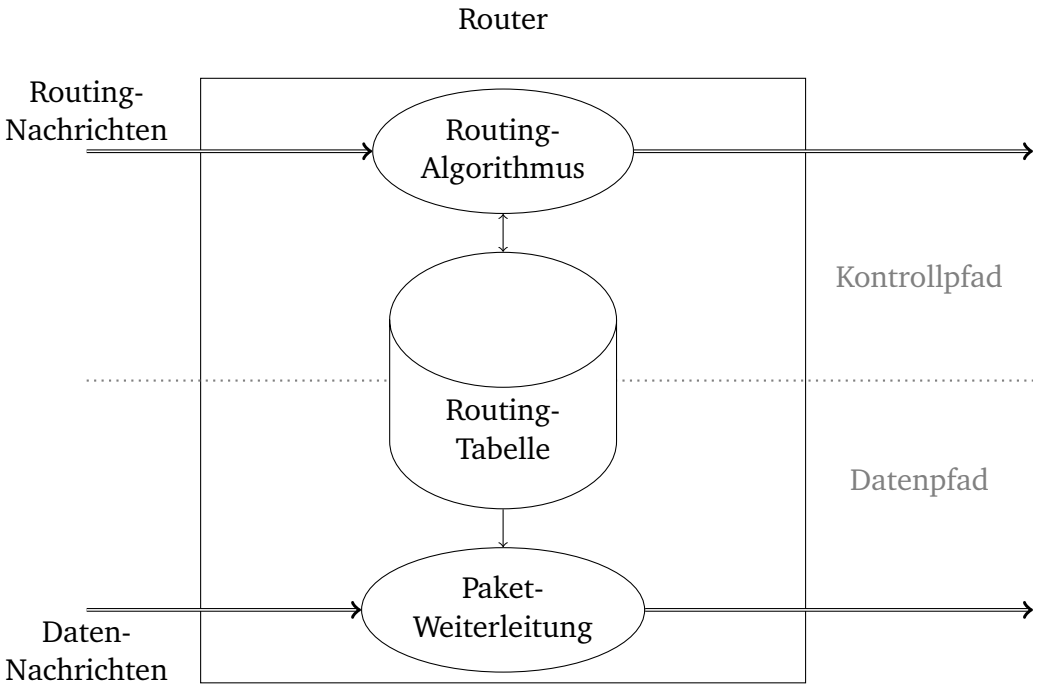


Abbildung 2.3: Konzeptionelles Modell eines Routers

bezüglich einer oder mehrerer Metriken zu wählen. Dies hat jedoch im Allgemeinen zur Folge, dass ein Router für jedes mögliche Ziel einen Eintrag in seinen Routing-Tabellen ablegen muss. Um optimale Pfade zu erhalten, ist also ein lineares Wachstum der Routing-Tabellen mit der Anzahl der Ziele notwendig. Für große Netze oder ressourcenbeschränkte Geräte ist dies aber nicht immer möglich oder erwünscht und daher beschäftigen sich viele Arbeiten mit der Entwicklung **skalierbarer** Routingprotokolle, die ein **sublineares** Wachstum der Routing-Tabellen erreichen. Von besonderer Bedeutung ist dabei der notwendige Kompromiss zwischen der Größe der Routing-Tabellen und der Qualität der Pfade, die üblicherweise über den Stretch gemessen wird.

Als Grundlage für die im Rahmen dieser Arbeit entwickelten Lösungen sind dabei insbesondere das kompakte Routing sowie gieriges Routing von Bedeutung. Diese werden nun konzeptionell und anhand prominenter Vertreter vorgestellt.

2.2.1 Kompaktes Routing

Die Wurzeln kompakter Routingverfahren reichen zurück bis ins Jahr 1988, als Peleg und Upfal erstmalig den Kompromiss zwischen der Größe der Routing-Tabellen und der Qualität der Pfade eines Routing-Schemas, gemessen durch den maximalen Stretch, formulierten [23, 24]. Als **Routing-Schema** wird ein verteilter Algorithmus bezeichnet, der die Zustellung von Paketen zwischen beliebigen Knoten eines Netzes erlaubt. Ein Routing-Schema umfasst dazu einerseits einen Algorithmus zur Vorberechnung der lokal auf jedem Knoten zu speichernden Information sowie eine Funktion, um darauf basierend Pakete weiterleiten zu können. Die üblicherweise globalen Routing-Schemata, die also von einer globalen Sicht auf die Topo-

logie ausgehen, können, wie im folgenden Abschnitt, weiter unterteilt werden. Nach einer kurzen Vorstellung bekannter Schranken werden kompakte Routing-Verfahren mit maximalem multiplikativem Stretch 3 und ein Routing-Protokoll, das auf diesen Ansätzen aufbaut, vorgestellt. Abschließend wird ein kompaktes Routing-Verfahren für Power-Law Graphen vorgestellt, das eine additive Stretch-Schranke bietet.

2.2.1.1 Klassifizerung kompakter Routing-Verfahren

Kompakte Routing-Verfahren können anhand verschiedener Kriterien unterteilt werden, von denen hier einige grundlegende vorgestellt werden.

In einem **labeled** Routing-Verfahren ist es möglich, Knoten für das Routing günstige Adressen zuzuweisen, die beispielsweise topologische Information enthalten. Dem stehen sogenannte **name-independent** Routing-Verfahren gegenüber, bei denen Knoten mit festen Adressen, also beispielsweise deren Identifikator, bezeichnet werden müssen. Dadurch ist es im Gegensatz zu labeled Routing-Schemata nicht mehr möglich, Informationen zur Unterstützung der Wegewahl in den Adressen zu kodieren. Name-independent Routing-Verfahren werden oft auf Basis von labeled Routing-Verfahren konstruiert. Über einen entsprechend konstruierten Namensdienst erhält man dann ausgehend vom Identifikator eines Zielknotens dessen Adresse und kann Pakete an diesen über das labeled Routing-Verfahren weiterleiten.

Des weiteren unterscheidet man zwischen **universellen** Routing-Schemata, die auf allen Arten von Graphen einsetzbar sind und **spezialisierten** Routing-Schemata, die für die Konstruktion der Routing-Tabellen Eigenschaften des Graphen, wie beispielsweise eine Baumstruktur, nutzen und nur auf diesen einsetzbar sind.

2.2.1.2 Bekannte Schranken

In ihrer grundlegenden Arbeit formulierten Peleg und Upfal den ersten Kompromiss zwischen maximalem multiplikativen Stretch und der Größe der Routing-Tabellen [24]. Sie bewiesen, dass in einem Netzwerk mit n Knoten für einen maximalen multiplikativen Stretch von $s \geq 1$ Routing-Tabellen mit einer Gesamtgröße von $\Omega(n^{1+\frac{1}{2s+4}})$ notwendig sind. Die resultierende untere Schranke für Routing entlang kürzester Pfade ($s = 1$) von $\Omega(n^{1+\frac{1}{2+4}})$ wurde dann in späteren Arbeiten weiter präzisiert. Für Routing-Schemata mit einem Stretch von $s < 2$ wurde ein Gesamt-Speicherbedarf von $\Omega(n^2)$ gezeigt, wobei auf einigen Routern lokale Routing-Tabellen der Größe $\Theta(n \log n)$ anfallen können [25]. Gavoille und Gengler zeigten später, dass sogar für jedes Routing-Schema mit einem maximalen multiplikativen Stretch von $s < 3$ mindestens Routing-Tabellen mit einer Gesamtgröße von $\Omega(n^2)$ erforderlich sind [26]. Für Routing-Schemata mit einem Stretch $s < 5$ ist eine Gesamtgröße der Routing-Tabellen von $\Omega(n^{1/2})$ erforderlich [27]. Abbildung 2.4 fasst diese Ergebnisse anschaulich zusammen.

2.2.1.3 Kompaktes Routing mit maximalem multiplikativem Stretch 3

Cowen entwickelte das erste universelle Routing-Schema [28], das bei einem maximalen multiplikativen Stretch von 3 mit $\tilde{O}(n^{2/3})$ sublineare Routingtabellen auf jedem Knoten

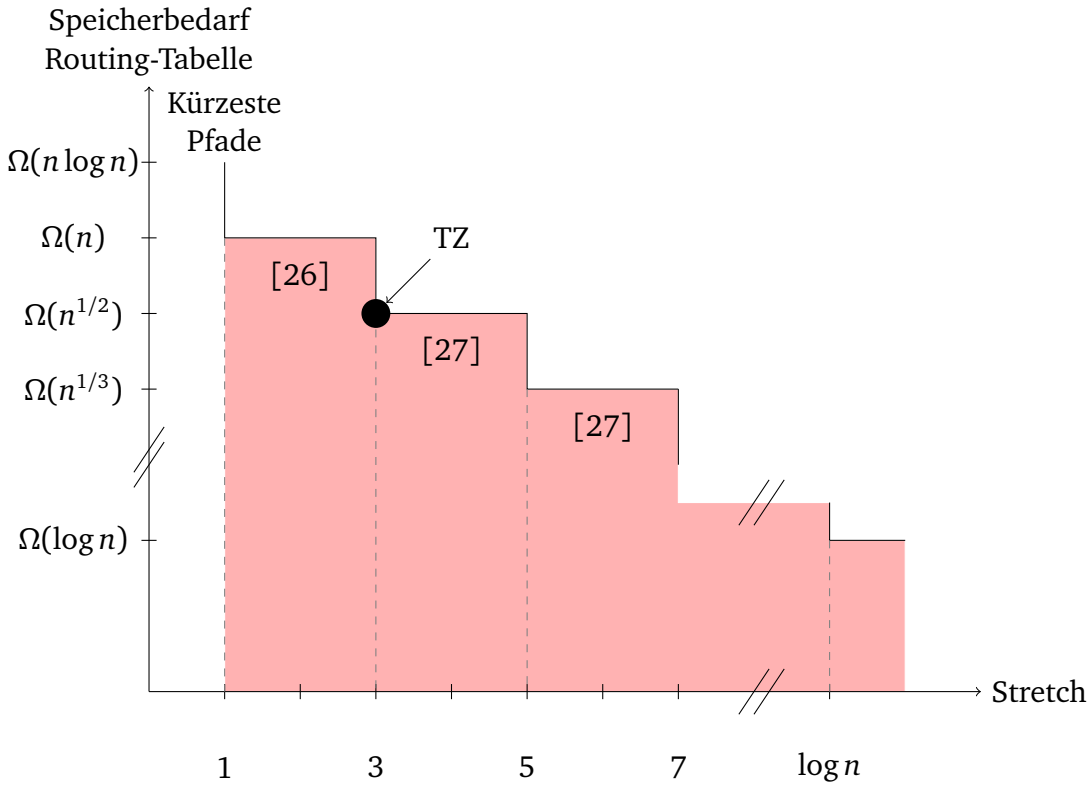


Abbildung 2.4: Untere Schranken für die Routingtabellengröße für universelle kompakte Routing-Verfahren in Abhängigkeit vom maximalen multiplikativen Stretch

erreicht[2]. Es basiert auf global bekannten Orientierungspunkten, sogenannten **Landmarks** und lokalen Nachbarschaften, sogenannten **Clusters**. Jeder Knoten kennt also die Landmarks und über die Clusters Knoten in einer gewissen Umgebung um sich herum, zu denen Pakete auf dem kürzesten Pfad weitergeleitet werden können. Zu weit entfernten Knoten werden Pakete zunächst auf dem kürzesten Pfad zu deren nächstem Landmark und anschließend von dort aus auf dem kürzesten Pfad zu dem jeweiligen Knoten weitergeleitet. Befindet sich das Ziel innerhalb des Clusters eines weiterleitenden Knotens, kann das Paket von dort über den kürzesten Pfad zugestellt werden.

Dieses Routing-Schema wurde von Thorup und Zwick (TZ) dahingehend verbessert, dass bei einem gleichbleibenden maximalen Stretch von $s = 3$ der erforderliche Zustand auf $\tilde{O}(n^{1/2})$ verbessert wurde. Damit ist das TZ Routing-Schema (siehe Abbildung 2.4) bis auf polylogarithmische Faktoren optimal als dass es bei einem maximalen Stretch von $s = 3$, der Voraussetzung für ein sublineares Routing-Tabellenwachstum ist, den minimalen erforderlichen Zustand benötigt. Da dieses TZ Routing-Schema auch als Routing-Protokoll realisiert wurde und stellenweise als Vergleich dient, wird dieses nun zunächst theoretisch vorgestellt und im folgenden Abschnitt 2.2.1.4 das entsprechendes Protokoll beschrieben.

Ausgehend von den Landmarks werden zunächst die Clusters definiert. Sei $A \subseteq V$ eine Menge von Landmarks und für einen Knoten $v \in V$ sei $d(A, v) = min\{d(u, v)|u \in A\}$. Für jeden Knoten $w \in V$ wird dann dessen Cluster $C(w)$ in Abhängigkeit von A definiert als

$$C(w) = \{v \in V | d(w, v) < d(A, v)\}$$

und damit für alle $a \in A$ $C(a) = \emptyset$.

Davon ausgehend speichert nun jeder Knoten w in seiner Routing-Tabelle für jedes Landmark $a \in A$ dessen Identifikator sowie den Ausgangsport port(w, a), der vom Knoten w auf kürzestem Wege zum Landmark a führt. Damit kann jeder Knoten Pakete zu allen Landmarks entlang des kürzesten Wegs weiterleiten. Ebenso wird für jeden Knoten $c \in C(w)$ der Identifkator und der Ausgangsport port(w, c) gespeichert, so dass diese auf dem kürzesten Weg erreichbar sind.

Die Anzahl der Einträge in der Routing-Tabelle auf einem Knoten w ist damit $O(|A| + |C(w)|)$; dies bedeutet, dass für eine kleine Routing-Tabelle einerseits die Anzahl der Landmarks und andererseits die Größe der Cluster klein sein muss. Das TZ Routing-Schema verwendet zur Bestimmung der Landmarks einen iterativen Algorithmus, der einerseits dafür sorgt, dass die Größe jedes Clusters sowie die Anzahl der Landmarks beschränkt ist durch $\tilde{O}(\sqrt{n \log n})$ und erreicht damit seinen bis auf logarithmische Faktoren optimalen Speicherbedarf der Routing-Tabellen.

Einem Knoten v wird dann die Adresse label(v) bestehend aus dessen Identifikator, seinem Landmark sowie dem Port vom Landmark zu Knoten v zugewiesen: label(v) = $(v, \text{lm}(v), \text{port}(\text{lm}(v), v))$.

Die Weiterleitung eines Paketes auf Knoten w zu Knoten t erfolgt dann, wie bei Cowen, anhand der Routing-Tabellen und der im Paketkopf enthaltenen Adresse des Ziels label(t). Ist der Weiterleitungsknoten der Zielknoten, so wird das Paket direkt ausgeliefert. Ist der Zielknoten t im Cluster $C(w)$ enthalten oder ein Landmark und damit port(w, t) Teil der

[2] $\tilde{O}$ unterdrückt polylogarithmische Faktoren

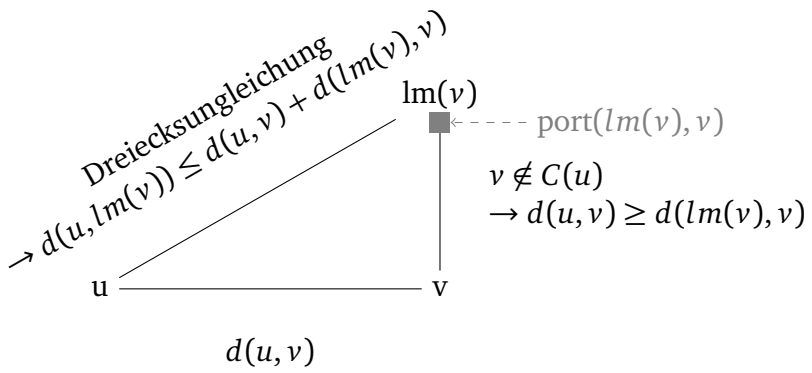


Abbildung 2.5: Skizze für Beweis des maximalen multiplikativen Stretch von $s = 3$ für das TZ Routing-Schema.

Routing-Tabelle, so wird das Paket auf dem kürzesten Weg in Richtung t weitergeleitet. Ansonsten wird das Paket zunächst in Richtung des Landmarks $\text{lm}(t)$ weitergeleitet, das als Teil der Adresse aus dem Paketkopf bekannt ist. Von dort aus über den ebenfalls aus dem Paket bekannten $\text{port}(\text{lm}(t), t)$ in Richtung von t.

Damit ist Routing von u nach v mit maximalen multiplikativen Stretch von $s = 3$ möglich. Ist v im Cluster $C(u)$ enthalten, so kann die Weiterleitung des Pakets auf dem kürzesten Pfad zu v erfolgen. Ansonsten wird das Paket zunächst auf dem kürzesten Pfad zum Landmark $\text{lm}(v)$ und anschließend von dort auf dem kürzesten Pfad zu v weitergeleitet. Abbildung 2.5 skizziert den Beweis für die Einhaltung des maximalen multiplikativen Stretchs bei der Zustellung eines Pakets von u nach v über dessen Landmark $\text{lm}(v)$. Dabei gilt unter Ausnutzung von Symmetrie und der Dreiecksungleichung

$$\begin{aligned} & d(u, lm(v)) && +d(lm(v), v) \\ \leq\ & d(u,v) + d(lm(v), v) && +d(lm(v), v) \\ \leq\ & d(u,v) + d(u,v) && +d(u,v) \\ \leq\ & 3d(u,v) \end{aligned} \tag{2.1}$$

Es gibt mehrere Routing-Protokolle, die auf dem universellen TZ Routing-Schema basieren [29, 30, 31], wovon Distributed Compact Routing [31] im nächsten Abschnitt exemplarisch vorgestellt wird.

2.2.1.4 Routingprotokoll mit maximalem multiplikativem Stretch 3

Distributed Compact Routing (Disco) [31] setzt sich zusammen aus einem auf Lokatoren basierten kompakten Routing-Protokoll NDDisco und einem verteilten Namensdienst. Im Folgenden wird dabei nur das labeled Routing-Protokoll NDDisco betrachtet, auf dessen Basis auch Disco realisiert ist.

NDDisco geht davon aus, dass die Größe n des Netzes jedem Knoten bekannt ist oder durch Schätzungsverfahren gewonnen werden kann. Um die gemäß dem TZ Routing-Schema benötigten $\Theta(\sqrt{n \log n})$ Landmarks verteilt zu bestimmen, wählt jeder Knoten

lokal unabhängig eine Zahl p gleichverteilt aus dem Intervall $[0,1]$ und wird zum Landmark, falls $p < \sqrt{(\log n)/n}$. Der Erwartungswert für die Anzahl von Landmarks ist damit $n \cdot \sqrt{(\log n)/n} = \sqrt{n \log n}$ und es wurde gezeigt, dass mit hoher Wahrscheinlichkeit $\Theta(\sqrt{n \log n})$ Landmarks entstehen. Diese Landmarks werden, wie später auch die Knoten der Nachbarschaften, die den Clusters entsprechen, über ein Pfad-Vektor Protokoll bekanntgemacht. Darüber lernt jeder Knoten einen Pfad zu jedem der Landmarks kennen. Zusätzlich lernen die Knoten nun ihre Nachbarschaft bestehend aus den $\sqrt{n \log n}$ nächstgelegenen Knoten kennen und es wurde gezeigt, dass dies dazu führt, dass jeder Knoten wiederrum mit hoher Wahrscheinlichkeit ein Landmark in seiner Nachbarschaft hat. Die Adresse eines Knotens v bestimmt sich dann aus dem Identifikator des Landmarks $\mathrm{lm}(v)$ sowie der Source-Route vom Landmark zu v. Die Weiterleitung von Paketen, deren Zielknoten in der Nachbarschaft liegt, erfolgt dann entlang des Pfades in den Routing-Tabellen; ansonsten wird das Paket über den allen Knoten bekannten Pfad zum Landmark des Zielknotens weitergeleitet und von dort aus gemäß der im Paket enthaltenen Source-Route zum Ziel zugestellt. In Kombination mit einer DHT-inspirierten Namensauflösung wird damit ein Identifikator-basiertes kompaktes Routing ermöglicht, das mit hoher Wahrscheinlichkeit einen maximalen multiplikativen Stretch 3 erreicht.

Die bisher betrachteten universellen Routing-Schemata und -Protokolle sind auf allen Topologien einsetzbar, während es aber auch spezialisierte Ansätze für beispielsweise Bäume oder Power-Law Graphen gibt. Die Klasse von Power-Law Graphen ist hier deshalb interessant, da man annimmt, dass die Knotengradverteilung des Internets einem Power-Law mit einem Exponenten von etwa 2 folgt [32, 33, 34, 35]. Eine Anpassung des TZ Routing-Schemas auf Power-Law Topologien [36] stellt analytisch eine Beziehung zu dem Power-Law Parameter und dem notwendigen Zustand her. Dabei wurde gezeigt, dass für Power-Law Graphen anstelle der $\tilde{O}(\sqrt{n})$ im allgemeinen TZ Routing-Schema in Abhängigkeit vom Power-Law Parameter $\beta \in (2,3)$ Routing-Tabellen mit einer erwarteten Größe von $O(n^{\gamma} \log n)$ konstruiert werden können mit $\gamma = \frac{\beta-2}{2\beta-3}$. Ein zentraler Aspekt dabei ist, dass die Landmarks nicht wie bei Thorup und Zwick zufällig ausgewählt werden, sondern die n^{γ} Knoten mit dem höchsten Grad als Landmarks genutzt werden.

2.2.1.5 Spezialisierte kompakte Routing-Verfahren für Power-Law Topologien

Neben TZ-basierten Ansätzen für Power-Law Graphen wurde von Brady und Cowen (BC) auch ein spezialisiertes kompaktes Routing-Schema für ungewichtete, ungerichtete Power-Law Topologien entwickelt, das einen maximalen additiven Stretch d garantiert und dabei einen, von einem konstruktionsbedingten Parameter e_F abhängigen, lokalen Zustand von $e_F \cdot \log^2(n)$ benötigt [2]. Das BC Routing-Schema wird an dieser Stelle detailliert vorgestellt, da in dem späteren Kapitel 5 mit Sprinkles ein darauf basiertes, verteiltes Routing-Protokoll vorgestellt wird. Das BC Routing-Schema basiert auf den in Abschnitt 2.1.4.1 vorgestellten Eigenschaften von Power-Law Topologien [15] und nutzt aus, dass Power-Law Topologien einen dicht vermaschten Kern besitzen, während die nicht nicht zum Kern gehörigen Komponenten des Graphen eine beschränkte Größe und Tiefe haben. Das BC Routing-Schema

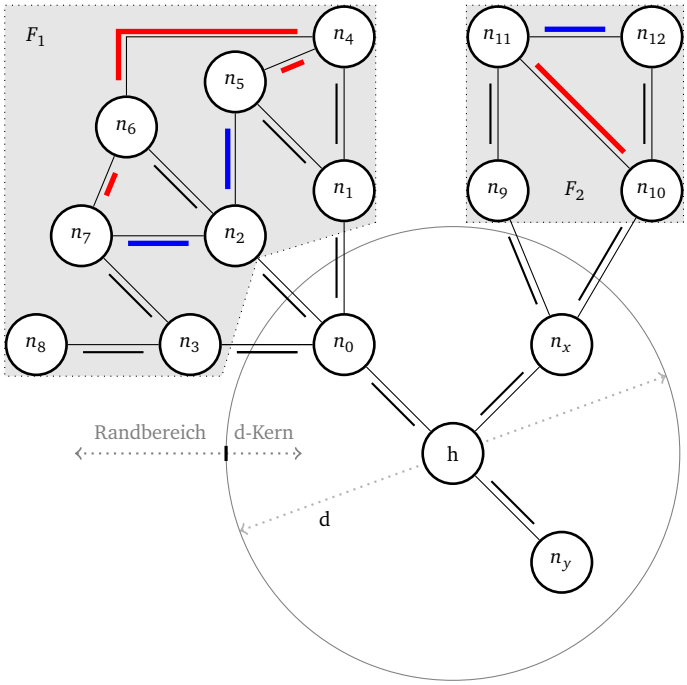


Abbildung 2.6: Funktionsweise des kompakten Routing-Schemas von Brady und Cowen [2].

verwendet daher unterschiedliche Strategien für den Kern und die Randbereiche des Graphen.

Aufbau der Routing-Tabellen Dazu wird der Graph G, wie in Abbildung 2.6 dargestellt, zunächst in einen Kern C (englisch: Core) mit Durchmesser d und den Randbereich F (englisch: Fringe) unterteilt. Da die Knoten mit dem höchsten Grad und insbesondere der Knoten h mit dem höchsten Grad jeweils im Kern eines Power-Law Graphen enthalten sind, werden alle Knoten im Abstand $d/2$ sowie die Kanten zwischen diesen als zum Kern C gehörig definiert. Die verbleibenden Teile des Graphen bilden den nicht notwendigerweise verbundenen Randbereich F, der sich aus $|F|$ in sich verbundenen Randbereichen F_i, $1 \leq i \leq |F|$ zusammensetzt. Im Folgenden werden anhand dieser Einteilung Spannbäume derart erzeugt, dass schließlich ein additiver Stretch von d erreicht wird. Innerhalb der Spannbäume werden Adressen der Länge $O(\log^2 n)$ vergeben, so dass einerseits die Bestimmung des Abstands zwischen zwei Knoten innerhalb eines Spannbaums anhand derer Adressen möglich ist [12] und Pakete entlang des Spannbaums weitergeleitet werden können [37]. Das Verfahren, das hier zur Bestimmung des Abstandes verwendet wird, ist das in Abschnitt 2.1.2 beschriebene Distanz-Label Verfahren. Nun wird über G ausgehend vom höchstgradigen Knoten h ein Spannbaum T_h, der sogenannte Hauptspannbaum, erzeugt (schwarze Parallelmarkierungen in Abbildung 2.6). Während sich über T_h bereits alle Knoten gegenseitig erreichen können, so sind für die Einhaltung der Stretch-Schranke weitere Spannbäume notwendig. Zunächst werden dafür die Randbereiche F_i bzw. deren Schnitt $F_i \cap T_h$ mit dem Hauptspannbaum betrachtet. Ist der Schnitt $F_i \cap T_h$ nicht verbunden, so werden diesem so lange Kanten hinzugefügt, bis für jeden Randbereich F_i ein Randspannbaum T_{F_i} bestimmt wurde. Diese Randspannbäume T_F stellen sicher, dass sich Knoten innerhalb

eines Randbereichs auch entlang eines Pfades innerhalb dieser Randbereiche erreichen können und Pakete nicht notwendigerweise über den Hauptspannbaum und damit über den Kern weitergeleitet werden müssen. In Abbildung 2.6 sind die in diesem Schritt hinzugefügten Kanten blau markiert. Schließlich wird noch eine dritte Kategorie von Spannbäumen benötigt, die für die Einhaltung der Stretchschranke notwendig ist und die ausgehend von sogenannten Extrakanten konstruiert wird. Die Extrakanten eines Graphen H sind dabei diejenigen Kanten, die von H entfernt werden müssen, damit H zyklenfrei ist. Die Anzahl von Extrakanten im Randbereich F sei im Folgenden e_F. Die für die Konstruktion relevanten Extrakanten $E' = \{(u,v)|(u,v) \in E; (u,v) \notin T_F; u,v \in F\}$ ($|E'| \leq e_F$) sind, wie beispielsweise die in Abbildung 2.6 rot markierten, Bestandteil eines Randbereichs, aber weder im Hauptspannbaum noch im jeweiligen Randspannbaum enthalten. Für jede solche Extrakante $e = (u,v)$ wird ein zusätzlicher Spannbaum auf G mit Wurzel u oder v aufgebaut, der die Kante e enthält.

Ein Knoten u ist damit maximal enthalten in

1. dem Hauptspannbaum T_h
2. einem Randspannbaum aus T_F (nur Knoten aus dem Randbereich)
3. e_F Extrakanten-Spannbäumen T_E.

Da die Adresslänge pro Baum beschränkt ist durch $O(\log^2 n)$, ergibt sich sowohl für die Routing-Tabellen eines Knotens als auch die Paketkopfgröße ein Speicherbedarf von $O(e_F \log^2 n)$. Zur Weiterleitung von Paketen von einem Sender u zu einem Zielknoten t wird dann in allen Bäumen T, die die Knoten u und v gemeinsam haben, aus der Adresse deren Distanz bestimmt. Die Weiterleitung erfolgt dann entlang des Baumes, der den besten Weg bietet.

Einhaltung der Stretch-Schranke Für den Beweis für die Einhaltung des maximalen additiven Stretch von d werden nun die folgenden Fälle unterschieden:

1. $u,v \in$ Kern: $d_{T_h}(u,v) \leq d_{T_h}(u,h) + d_{T_h}(h,v) = d(u,h) + d(h,v) \leq d/2 + d/2 = d$. Da $d(u,v) \geq 1$, ist $d_T(u,v) \leq d(u,v) + (d-1) < d(u,v) + d$.
2. $u \in$ Kern, $v \in$ Randbereich: $d_T(u,v) \leq d_T(u,h) + d_T(h,v) = d(u,h) + d(h,v) \leq d/2 + d(h,v) \leq d/2 + d(u,h) + d(u,v) \leq d/2 + d/2 + d(u,v) = d(u,v) + d$
3. $u,v \in$ Randbereich: Ist ein kürzester Pfad zwischen u und v Teil des Randspannbaums, so erfolgt die Weiterleitung entlang des kürzesten Pfades. Ist hingegen $d_{T_F}(u,v) > d(u,v)$, so muss es eine Kante (u',v') geben, die im kürzesten Pfad P enthalten, aber nicht Bestandteil von T_F ist.
   a) $u',v' \in$ Randbereich: Ist eine Kante (u',v') in der Randbereich nicht Bestandteil von T_F, so muss laut Konstruktionsvorschrift ein Extra-Spannbaum T von u' (oder v') aus konstruiert worden sein, der diese Kante enthält. Damit ist $d_T(u,v) = d_T(u,u') + d_T(u',v) = d(u,u') + d(u',v)$. Da u' auf einem kürzesten Pfad zwischen den Knoten u und v liegt, muss daher entsprechend $d_T(u,v) = d(u,v)$ sein.
   b) u'oder $v' \in$ Kern: Ist der kürzeste Pfad nicht vollständig im Randbereich enthalten, so können Pakete über den Hauptspannbaum T_h weitergeleitet werden. Da mindestens ein Knoten u' des kürzesten Pfades im Kern liegt, gilt

$d(u,u') + d(u',v) = d(u,v)$. Damit folgt $d(u,v) = d(u,u') + d(u',v) \leq d_{T_h}(u,h) + d_{T_h}(h,v) \leq d_{T_h}(u,x) + d/2 + d/2 + d_{T_h}(x,v) \leq d(u,v) + d$.

Da das Routing innerhalb der Spannbäume ohne Stretch erfolgt und es, wie gezeigt, immer einen Baum T gibt, entlang dessen eine Weiterleitung mit maximalem additivem Stretch d möglich ist, hat das BC Routing-Schema einen maximalen additiven Stretch von d.

Zusammenfassung Das BC Routing-Schema erreicht insbesondere im für Internet-ähnliche Topologien relevevanten Bereich $\beta \in \{2.0, 2.1, 2.2\}$ einen besseren durchschnittlichen Stretch als das TZ Routing-Schema und ist damit für die Praxis von Interesse. Auch die Beschränkung des maximalen Stretch macht das BC Routing-Schema für die Praxis attraktiv. Ein Problem für einen praktischen Einsatz ist jedoch, dass das BC Routing-Schema zentralisiert arbeitet und für die Bestimmung der Adressen und Routing-Tabellen globales Wissen, beispielsweise zur Bestimmung der Distanz-Labels, erfordert. Eine Konstruktion der BC Routing-Tabellen durch ein verteiltes Routing-Protokoll ist daher nicht ohne weiteres möglich.

2.2.2 Gieriges Routing

Gierige Routing-Verfahren leiten Pakete immer an den Nachbarn weiter, der die Distanz zum Ziel minimiert und haben den Vorteil, dass die Routing-Tabellen nur aus den Adressen der Nachbarn bestehen. Aus diesen Adressen und der Adresse des Ziels kann dann bestimmt werden, welcher Nachbar die Distanz zum Ziel minimiert.

Ein intuitives Beispiel eines solchen gierigen Routing-Verfahren ist das geographische Routing. Mit dem Aufkommen und steigender Verbeitung von GPS-Empfängern enstand die Idee, die damit auf den Knoten zur Verfügung stehenden Koordinaten für das Routing zu verwenden [38]. Kennt ein Knoten nun die Koordinaten seiner Nachbarn sowie des Ziels, kann er daraus gierig den Weiterleitungsknoten bestimmen.

Allgemein kann man die Weiterleitungsvorschrift für gierige Routing-Verfahren folgendermaßen formulieren:

Ein Knoten v mit den Nachbarn $\Gamma_G(v)$ bestimmt bei gieriger Paketweiterleitung den besten Weiterleitungsknoten w für ein Paket mit Zielknoten t durch

$$w := \arg\min_{u \in \Gamma_G(v)} \{d(v,u) + d(u,t)\}. \tag{2.2}$$

Damit diese Art der Weiterleitung erfolgreich ist, muss also immer ein Nachbar existieren, der die Distanz zum Ziel weiter reduziert.

Ein Problem, das bei gierigem Routing auftreten kann, sind **lokale Minima**, wie in Abbildung 2.7 beispielhaft skizziert. Knoten v_1 prüft anhand von Gleichung 2.2, über welchen Nachbarn er ein Paket an den Zielknoten v_7 weiterleiten muss. In Abbildung 2.7 sind dazu die euklidischen Abstände der Nachbarn zum Zielknoten v_7, wie sie beispielsweise durch GPS-Koordinaten bestimmt werden könnten, durch grün gestrichtelte Linien eingetragen. Da $d(v_1,v_3) + d(v_3,v_7)$ auf einer Geraden liegt, ist dies hier die minimale Distanz und das Paket wird an Knoten v_3 weitergeleitet. Prüft nun Knoten v_3, über welchen Nachbarn das Paket

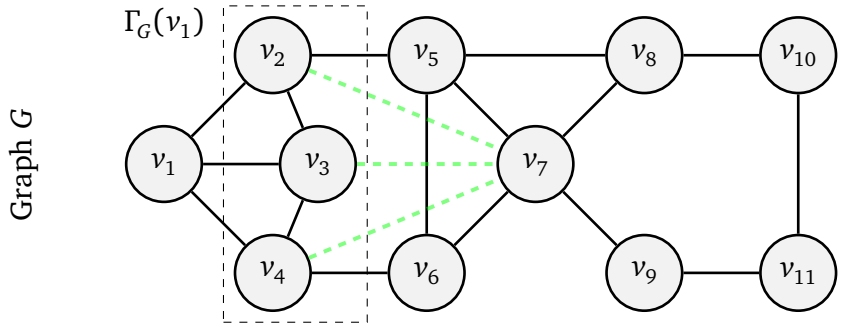


Abbildung 2.7: Knoten v_3 ist ein lokales Minimum bei gieriger Paketweiterleitung von Knoten v_1 an Zielknoten v_7

weitergeleitet werden muss, so gibt es keine Möglichkeit, das Paket näher an das Ziel heranzuführen. Knoten v_3 ist damit ein lokales Minimum. Ein Paket, das in ein lokales Minimum gerät, muss entweder verworfen werden oder durch eine gesonderte Weiterleitungs-Strategie behandelt werden. Für planare Graphen können Pakete beispielsweise über das sogenannte Face Routing [39] in Gegenwart lokaler Minima garantiert zugestellt werden.

Ein anderer Ansatz ist es, statt Verfahren zur Behandlung lokaler Minima zu verwenden, dafür Sorge zu tragen, dass eine gierige Weiterleitung stets erfolgreich möglich ist. Dies ist beispielsweise anhand der in Abschnitt 2.1.3 vorgestellten gierigen Einbettungen möglich. In verteilten Umgebungen haben Spannbäume sich als besonders geeignete Strukturen für gierige Einbettungen herausgestellt und werden daher im folgenden Abschnitt betrachtet. Schleifen und lokale Minima sind in Abwesenheit von Fehlern dabei nicht möglich, da sich die Distanz zum Ziel mit jeder eindeutig bestimmten Weiterleitung in dessen Richtung reduziert.

2.2.2.1 Spannbaum-basierte gierige Einbettungen

Es gibt viele Ansätze für Routing-Verfahren, die auf der gierigen Einbettung von Spannbäumen basieren [40, 41, 42, 14, 43, 44]. Darauf basierende Routing-Protokolle werden im folgenden der **Klasse spannbaum-basierter gieriger Routing-Verfahren** zugeordnet. Die grundlegende Idee bei dieser Klasse von Routing-Protokollen ist es, nicht etwa den zugrundeliegenden Graphen G gierig einzubetten sondern stattdessen einen einfacher einzubettenden Spannbaum T über G. Im Rest der Arbeit sind diese Spannbäume immer Bäume kürzester Pfade und enthalten also einen kürzesten Pfad von jedem Knoten des Spannbaums zu dessen Wurzel.

Auf der Beispiel-Topologie in Abbildung 2.8 wird dafür zunächst ein einzelner Spannbaum T_1 konstruiert und dieser anhand einer Abbildung f_{T_1} gierig eingebettet (siehe Abschnitt 2.1.3). Für eine kompaktere Schreibweise der virtuellen Koordinate eines Knotens u wird $f_{T_1}(u)$ im Folgenden als u^{T_1} notiert. Für zwei beliebige Knoten $u, v \in T$ kann dann anhand derer Koordinaten und der Distanzmetrik $d_T(u^T, v^T)$ die Entfernung der Knoten entlang des Baumes T bestimmt werden.

Werden Pakete entlang von T_1 weitergeleitet, so kann gegenüber einer Weiterleitung entlang

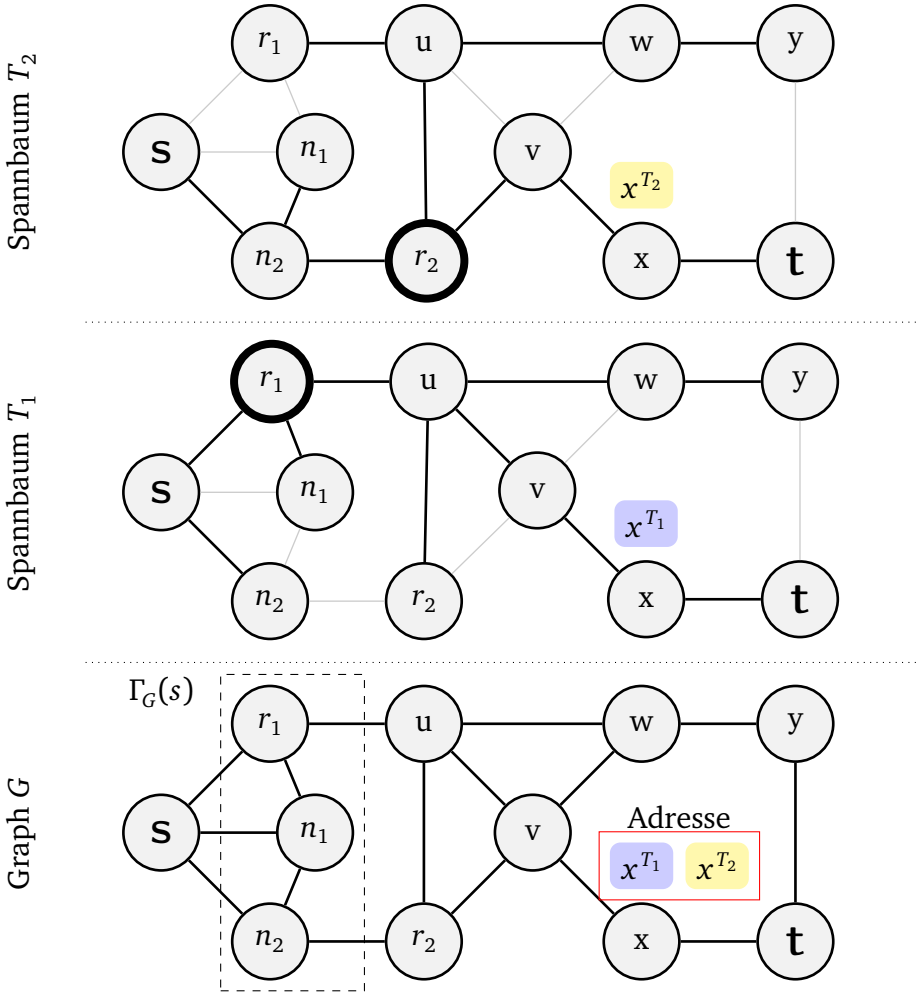


Abbildung 2.8: Konstruktion von Routing-Tabellen basierend auf spannbaumbasierten gierigen Einbettungen

des kürzesten Pfades in dem zugrundeliegenden Graphen G ein hoher Stretch entstehen. Um die Wahrscheinlichkeit für das Zustandekommen von hohem Stretch zu reduzieren, werden bei gieriger Paketweiterleitung auch Nicht-Spannbaumkanten zur Weiterleitung von Paketen genutzt, wenn der entsprechende Nachbar die Distanz zum Ziel minimiert. Dies ist möglich, da jeder Knoten die Adressen aller Nachbarn, also nicht nur der Nachbarn in einem gewissen Spannbaum, kennt. Nicht-Spannbaumkanten, die einen kürzeren Weg zum Ziel ermöglichen als eine Weiterleitung entlang des Spannbaums, werden auch als **Abkürzungen** bezeichnet. Eine weitere Möglichkeit zur Reduktion des Stretch ist die Verwendung weiterer gierig eingebetteter Spannbäume, wie in Abbildung 2.8 das Hinzufügen von T_2. Ebenso kann der Aufbau des Spannbaumes, also welche Kanten in diesem enthalten sind und welche nicht, Einfluss auf den Stretch haben [45].

In dem Beispiel in Abbildung 2.8 hat Spannbaum T_1 den Knoten r_1 zur Wurzel und Spannbaum T_2 hat Knoten r_2 als Wurzel. Knoten x wurde in Spannbaum T_1 die virtuelle Koordinate x^{T_1} und in Spannbaum T_2 die virtuelle Koordinate x^{T_2} zugewiesen. Die Konkatenation dieser beiden virtuellen Koordinaten ist schließlich die Adresse von Knoten x.

Um ein Paket von Knoten s an den Zielknoten t über einen Baum T weiterzuleiten, wird der Weiterleitungsknoten w auf Knoten u folgendermaßen nach dem Greedy-Prinzip bestimmt:

$$w := \arg\min_{v \in \Gamma_G(u)} \left\{ d(u,v) + d_T(v^T, t^T) \right\} \tag{2.3}$$

arg min liefert dabei das Argument zurück, für das die Funktion ihr Minimum annimmt.

Stehen mehrere Bäume zur Weiterleitung von Paketen zur Verfügung, so wird unter diesen derjenige gewählt, in dem die Distanz zum Ziel minimiert werden kann.

Sei τ die Menge aller eingebetten Spannbäume. Die Distanz $d_\tau(u,v)$ zweier Knoten ist dann die minimale Distanz aus allen Bäumen, in denen sowohl u als auch v eingebettet sind.

$$d_\tau(u,v) = \min_{T \in \tau} \left\{ d_T(u^T, v^T) \right\} \tag{2.4}$$

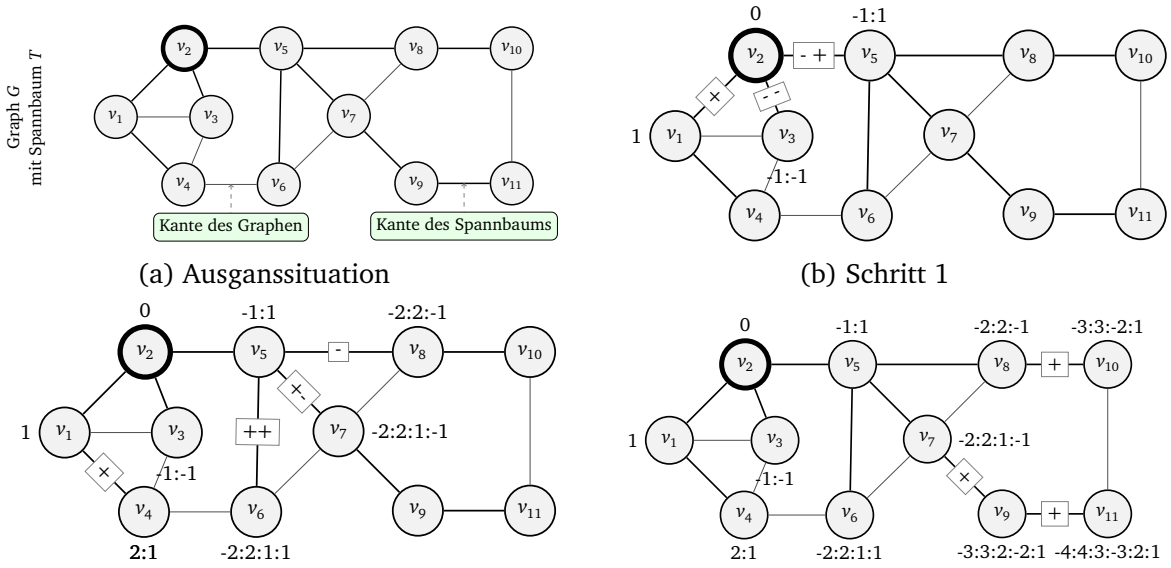
Ersetzt man nun d_T in Gleichung 2.3 mit dem allgemeineren d_τ aus Gleichung 2.4, so erhält man die folgende Gleichung zur Bestimmung des Weiterleitungsknotens:

$$w := \arg\min_{v \in \Gamma_G(u)} \{ d(u,v) + d_\tau(v,t) \} \tag{2.5}$$

Während in diesem Abschnitt die allgemeine Idee spannbaum-basierter gieriger Routing-Verfahren skizziert wurde, stellt der nachfolgende Abschnitt mit Practical Isometric Embedding ein solches Routing-Protokoll vor.

2.2.2.2 Practical Isometric Embedding

Practical Isometric Embedding (PIE) [44] ist ein skalierbares gieriges Routingprotokoll für ungerichtete Topologien. PIE basiert auf der Konstruktion mehrerer Spannbäume, von denen jeder isometrisch (und damit gierig) eingebettet wird, und entlang derer Pakete dann gierig weitergeleitet werden können. Durch den Verzicht auf eine nicht-triviale Beschränkung des maximalen Stretch erzielt PIE ein polylogarithmisches Wachstum der Routing-Tabellen in



(a) Ausganssituation (b) Schritt 1

(c) Schritt 2 (d) Schritt 3 und Schritt 4

Abbildung 2.9: Koordinatenvergabe durch die isometrische Baum-Einbettung nach PIE.

Abhängigkeit von der Anzahl Knoten in der Topologie. Zudem erreicht PIE auf den untersuchten Power-Law Topologien einen niedrigen durchschnittlichen Stretch $s_m \leq 1.25$ auf ungewichteten Topologien.

Im Folgenden wird zunächst die isometrische Einbettung von Bäumen eingeführt und anschließend die Konstruktion der einzubettenden Spannbäume sowie die Weiterleitung von Paketen erläutert.

Isometrische Einbettung von Bäumen Die isometrische Einbettung eines Spannbaums durch PIE erfolgt anhand der durch die Maximumsnorm induzierte Metrik. Die Einbettung in diese Metrik sowie die Distanzbestimmung anhand zweier virtueller Koordinaten werden hier anhand eines anschaulichen Beispiels unter Verwendung von Abbildung 2.9 erklärt. Kanten des Spannbaums T sind dabei, wie in Abbildung 2.9a annotiert, dick gezeichnet und Kanten des Graphen G, die nicht Teil des Spannbaums sind, sind dünn dargestellt.

Die Einbettung des Baumes wird durch die Wurzel initiiert, die sich selbst die virtuelle Koordinate {0} zuordnet (siehe Abbildung 2.9b). Die Wurzel v_2 hat 3 Kindknoten in T und ordnet jedem seiner Kindknoten einen eindeutigen binären präfixfreien Code ($v_1 \rightarrow \{+\}$, $v_3 \rightarrow \{--\}$, $v_5 \rightarrow \{-+\}$) zu. Diesen präfixfreien Code sendet v_2 inklusive der eigenen virtuellen Koordinate an das jeweilige Kind.

Empfängt ein Knoten nun von seinem Elternknoten ein Tupel bestehend aus der virtuellen Koordinate des Elternknotens sowie präfixfreiem Code, bestimmt der Knoten daraus zunächst seine eigene virtuelle Koordinate. Ausgehend von Einheitskantengewichten inkrementiert er dazu erhaltene positive Koordinaten und dekrementiert erhaltene negative Koordinaten; die Wurzelkoordinate {0} ist dabei ein Sonderfall und kann für eine Verkürzung der Adresslängen

ignoriert werden. An die so verarbeitete Koordinate des Vorgängers hängt jeder Knoten anschließend eine 1 für ein empfangenes + und eine -1 für ein empfangenes - an. Damit ergeben sich die in Abbildung 2.9b dargestellten virtuellen Koordinaten für die Knoten v_1, v_3 und v_5.

Nach der Bestimmung der eigenen virtuellen Koordinate fährt jeder Knoten, wie zuvor beschrieben, mit der Vergabe von Koordinaten an seine eigenen Kinder fort, d.h. auch er bestimmt wieder einen binären präfixfreien Code und sendet diesen an seine Kinder, solange bis jeder Knoten im Baum über eine virtuelle Koordinate verfügt. Abbildung 2.9c zeigt entsprechend die Vergabe eines binären präfixfreien Codes durch die vorherigen Kindknoten v_1, v_3 und v_5 an deren Kindknoten sowie die resultierenden Koordinaten dieser Kindknoten. Abbildung 2.9d stellt die verbleibenden Schritte dar und zeigt die Topologie nach erfolgter Koordinatenvergabe.

Die so erhaltenen virtuellen Koordinaten werden anschließend allen Nachbarn bekanntgemacht, so dass Pakete zwischen direkten Nachbarn immer direkt zugestellt werden können und damit später eine gierige Paketweiterleitung möglich ist. Die gierige Paketweiterleitung anhand der virtuellen Koordinaten ist möglich, da eine isometrische Einbettung eines Baumes, in dem es per Definition ja nur einen Weg zwischen zwei beliebigen Knoten gibt, immer auch eine gierige Einbettung ist.

Die Bestimmung der Distanz zwischen zwei virtuellen Koordinaten ist nun über die durch die Maximumsnorm induzierte Metrik $||a-b||_\infty$ möglich, wobei $||x||_\infty = max_i|x_i|$. Für beispielsweise die Knoten v_9 und v_4 bestimmt sich die Distanz dadurch folgendermaßen:

$$\begin{array}{rrrrrrrr} & -3 & 3 & 2 & -2 & 1 & & (v_9) \\ - & 2 & -1 & & & & & (v_4) \\ \hline || & -5 & 4 & 2 & -2 & 1 & ||_\infty = \mathbf{5} & \end{array}$$

Konstruktion der Spannbäume Verwendet man, wie in dem bei der Vergabe der Koordinaten angeführten Beispiel, nur einen Spannbaum, so kann, wie bereits in Abschnitt 2.2.2.1 angedeutet, ein hoher Stretch resultieren. Dies ist für die Knoten v_1 und v_6 der Fall. Bei einer Weiterleitung entlang des Spannbaums werden 4 Übertragungsabschnitte benötigt, während für den kürzesten Pfad nur 2 Übertragungsabschnitte erforderlich sind. Damit resultiert ein multiplikativer Stretch von 2.

Um das Auftreten von hohem Stretch zu reduzieren, können mehrere Spannbäume über dem Graphen aufgebaut werden, von denen jeder individuell eingebettet wird. Die Adresse eines Knotens setzt sich dann zusammen aus der Konkatenation der virtuellen Koordinaten in den einzelnen Spannbäumen. Pakete können damit in jedem der Spannbäume in Richtung des Ziels weitergeleitet werden, was die Auswahl des günstigsten Spannbaums und eine Reduktion des Stretch ermöglicht. Während die Nutzung mehrerer Spannbäume den Stretch reduziert, erhöht sich dadurch auch die Länge der Adressen.

Um die Länge der Adressen kurz zu halten, führt PIE sogenannte **Baum-Ebenen** ein und konstruiert durch eine Partitionierung des Graphen lokale Spannbäume, wie in Abbildung 2.10 dargestellt. Werden insgesamt l Baum-Ebenen konstruiert, so werden auf Baum-Ebene i, $0 \leq i < l$, je 2^i Spannbäume aufgebaut. Knoten, die keine Wurzel auf einer Baum-Ebene

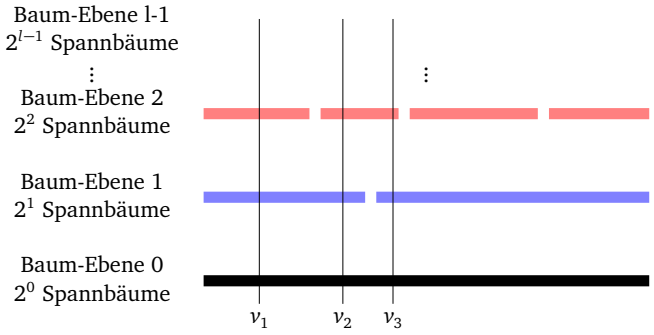


Abbildung 2.10: Schematische Darstellung der PIE Baum-Ebenen.

sind, schließen sich auf dieser Ebene dem Spannbaum an, zu dessen Wurzel sie die kleinste Entfernung haben.

Da es auf Ebene 0 nur eine Wurzel gibt, entsteht ein globaler Spannbaum, der die Konnektivität zwischen allen Knoten sicherstellt. Spannbäume auf höheren Ebenen decken im Gegensatz dazu nicht den ganzen Graphen ab. Deren Aufgabe ist im Gegensatz zum Baum auf Ebene 0 nicht die Sicherstellung der Konnektivität, sondern die Bereitstellung zusätzlicher Pfade, entlang derer Pakete zu einem Zielknoten weitergeleitet werden können.

Wie in Abbildung 2.10 angedeutet, ist die Weiterleitung von Paketen zwischen den Knoten v_1 und v_2 damit neben dem Spannbaum auf Ebene 0 nur in dem blauen Baum auf Ebene 1 möglich, in dem beide Knoten enthalten sind. Für die Weiterleitung von Paketen zwischen den Knoten v_2 und v_3 kann neben dem Spannbaum auf Ebene 0 der gemeinsame rote Baum verwendet werden. Für die Bestimmung der Wurzeln der Spannbäume wird ein von der Netzgröße abhängiger Zufallsprozess vorgeschlagen, der mit hoher Wahrscheinlichkeit in der gewünschten Anzahl von Wurzeln bzw. Spannbäumen pro Baum-Ebene resultiert. Für jede Baum-Ebene $0 \leq i < l$ wird jeder Knoten basierend auf einer rein lokalen Entscheidung mit einer Wahrscheinlichkeit von $2^l/n$ Wurzel, woraus eine erwartete Anzahl von 2^l Spannbäumen resultiert.

Es wird als gute Abwägung zwischen Stretch und Adresslänge empfohlen, die Anzahl der Baum-Ebenen l in Abhängigkeit von der Netzgröße n zu wählen als $l = \lfloor \log_2(n) - 7 \rfloor$. Abbildung 2.11 stellt die resultierende Treppenfunktion für l in Abhängigkeit von der Netzgröße dar. Für das heutige Internet auf Router-Ebene wären nach dieser Empfehlung 11 Baum-Ebenen zu verwenden.

Der **Aufbau der Spannbäume** bei PIE erfolgt über den *Tree Maintainer*, der einem vereinfachten und leicht angepassten Spanning Tree Protocol [46] entspricht. Der Tree Maintainer aus Sicht eines Knotens u ist in Algorithmus 1 dargestellt.

Während der Initialisierung von Knoten u (Zeilen 2–5) werden zunächst die zur Verwaltung der Baumstruktur notwendigen Felder initialisiert. Die *rootID* wird bei der Bestimmung des globalen Spannbaumes auf Baum-Ebene 0 über den Grad von Knoten u initialisiert und zur Unterscheidung zwischen Wurzeln mit gleichem Knotengrad eine Zufallszahl addiert. Die Höhe *height* beschreibt den Abstand eines Knotens zur Wurzel des Baumes und wird mit 0 initialisiert. Da Knoten zu Beginn keinen Elternknoten haben, wird *parent* mit $\emptyset$ initialisiert.

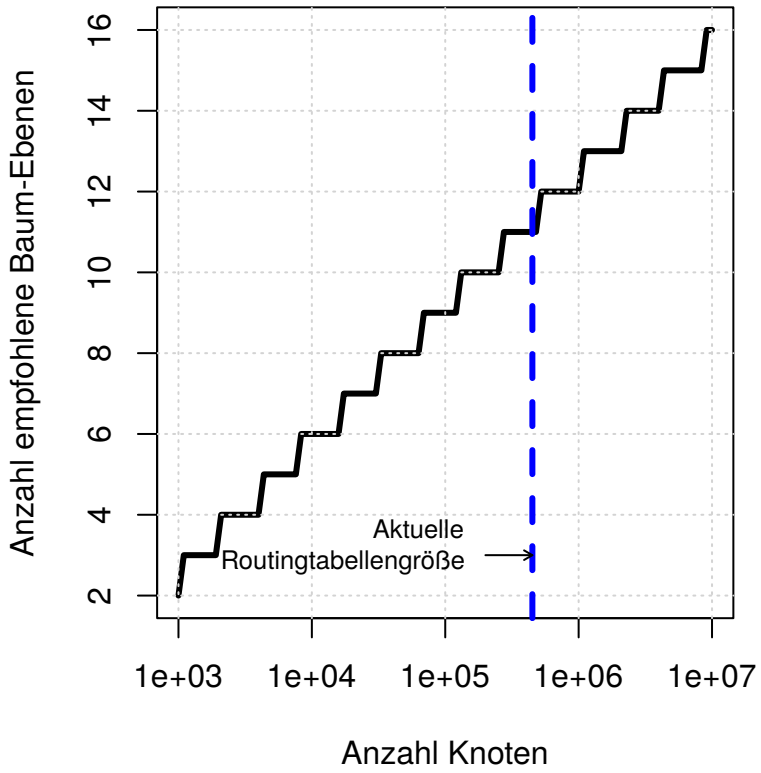


Abbildung 2.11: Empfohlene Anzahl an Baum-Ebenen für PIE in Abhängigkeit von der Netzgröße

Algorithmus 1 PIE Tree Maintainer auf Knoten u

```
1:  Init:
2:  u.rootID ← Knotengrad von u (+ Zufallszahl aus Intervall [0,1[)        ▷ Wurzel
3:  u.height ← 0                                          ▷ Speichert Abstand zur Wurzel
4:  u.parent ← ∅                                                ▷ Elternknoten im Baum
5:  u.children ← ∅                                              ▷ Menge von Kindknoten
6:
7:  Periodisch:
8:  sende Baum-Nachricht(u.rootID, u.height, u.parent) an Nachbarn
9:
10: Bei Empfang einer Baum-Nachricht msg von Nachbar v:
11: w_{v,u} ← w(v,u)                          ▷ Kosten für letzten Übertragungsabschnitt
12: if (msg.rootID > u.rootID) or
          (msg.rootID == u.rootID and msg.height + w_{v,u} < u.height) then
13:     u.parent ← v
14:     u.height ← msg.height + w_{v,u}
15:     u.rootID ← msg.rootID
16: end if
17: if msg.parent == u and v ∉ u.children then
18:     add(u.children, v)                  ▷ Füge Knoten v zu Kindknoten hinzu
19: end if
20: if msg.parent ≠ u and v ∈ u.children then
21:     remove(u.children, v)               ▷ Knoten v aus Kindknoten entfernen
22: end if
```

Zum Zeitpunkt der Initialisierung wurde ein Knoten noch von keinem anderen Knoten als Elternknoten bestimmt und daher wird die Menge *children* als leere Menge initialisiert.

Nach erfolgter Initialisierung werden periodisch Baum-Nachrichten versendet, die die rootID, height und den Elternknoten von Knoten u enthalten (Zeile 8).

Beim Empfang einer Baum-Nachricht (Zeilen 11–21) wird geprüft, ob der annoncierte Baum besser ist als der bisher bekannte. Dieser Fall tritt unter zwei Bedingungen ein:

1. Es wird eine Nachricht von einer bisher unbekannten Wurzel mit höherem Knotengrad empfangen.
2. Es wird eine Nachricht von der aktuell gewählten Wurzel über einen kürzeren Pfad empfangen.

Ist dies der Fall, wechselt Knoten u seinen Baum und schließt sich dem Baum mit der besseren Wurzel msg.rootID an (Zeilen 13–15).

Abschließend wird noch die Menge von Kindknoten aktualisiert (Zeilen 17–21). Hat sich ein Knoten laut einer empfangenen Baum-Nachricht Knoten u als Elternknoten gewählt und ist noch kein Kindknoten von u, so wird dieser als Kind eingetragen. Umgekehrt werden Knoten aus der Menge der Kindknoten entfernt, wenn diese inzwischen einen anderen Elternknoten gewählt haben.

Bei l Baum-Ebenen muss dieser Algorithmus leicht modifiziert werden und zu jedem der nun zu verwaltenden Bäume zusätzlich die Baum-Ebene gespeichert werden. Auf Baum-Ebenen außer Baum-Ebene 0 ändert sich zudem die Bedingung, unter der ein Baumwechsel stattfindet. Ein Knoten schließt sich dem Spannbaum auf Baum-Ebene k an, zu dessen Wurzel (aus ingesamt 2^k) er die kleinste Distanz hat. Damit ergeben sich die in Abbildung 2.10 bereits skizzierten Baum-Ebenen.

Weiterleitung von Paketen Die Weiterleitung von Paketen erfolgt bei PIE nach dem Greedy-Prinzip. Gleichung 2.2, die die grundsätzliche gierige Weiterleitung von Paketen beschreibt, muss für PIE bei Verwendung von l Baum-Ebenen leicht angepasst werden. Für die Weiterleitung von Paketen, können bei Verwendung von l Baum-Ebenen mindestens der Baum auf Ebene 0 und im Idealfall alle l Baum-Ebenen verwendet werden. In die Bestimmung des Weiterleitungsknotens dürfen daher nur die für die Weiterleitung verwendbaren Bäume, in denen also der Weiterleitungsknoten und das Ziel enthalten sind, einbezogen werden:

$$w := \underset{u \in \Gamma_G(v)}{\arg\min} \quad \min_{\substack{l : \exists T^l \text{ mit} \\ ||(t^{T^l} - u^{T^l})||_\infty < ||(t^{T^l} - v^{T^l})||_\infty}} \left\{ d(v,u) + d_{T^l}(u^{T^l}, t^{T^l}) \right\}. \tag{2.6}$$

2.3 Zusammenfassung der Notationen

Abschließend werden zur besseren Übersicht in Tabelle 2.1 die in diesem Kapitel eingeführten Notationen nochmals zusammengefasst.

Tabelle 2.1: Zusammenfassung der eingeführten Notationen.

Abkürzung	Bedeutung
G	Ein Graph
V	Die Knoten eines Graphen
E	Die Kanten eines Graphen
n	Die Anzahl von Knoten $\|V\|$ eines Graphen
m	Die Anzahl von Kanten $\|E\|$ eines Graphen
$\Gamma_G(v)$	Die Nachbarschaft des Knoten v im Graphen G
$\deg_G(v)$	Der Knotengrad, also die Anzahl von Nachbarn des Knotens v im Graphen G
h	Der Knoten mit dem höchsten Grad eines Graphen
$d_G(x,y)$	Die Länge eines kürzesten Pfades zwischen den Knoten x und y im Graphen G
$diam_G$	Der Durchmesser des Graphen G, also der längste kürzeste Pfad
s_a	Additiver Stretch
s_m	Multiplikativer Stretch
β	Der Power-Law Exponent
d	Kerndurchmesser und additive Stretch-Schranke des BC Routing-Verfahrens
C	Der Kern mit Durchmesser d um den höchstgradigen Knoten h
F	Die Randbereiche, also $G \setminus C$
T_h	Der vom höchstgradigen Knoten h aus konstruierte Hauptspannbaum
T_F	Die Randspannbäume, die einzelne Randbereiche intern verbinden
e_F	Anzahl Extrakanten im Randbereich
E'	Nach Konstruktion der T_F nicht durch eine Spannbaumkante abgedeckte Kanten im Randbereich; $\|E'\| \leq e_F$
T_E	Extrakanten-Spannbäume die den gesamten Graphen umfassen
$f_T(u)$	Die Einbettung von Knoten u im Spannbaum T
u^T	Kompaktere Notation für $f_T(u)$
τ	Die Menge aller eingebetteten Spannbäume
$d_\tau(x,y)$	Distanzberechnung über alle eingebetteten Spannbäume τ, entlang derer eine Weiterleitung von x nach y möglich ist
$\|\|x-y\|\|_\infty$	Von der Maximumsnorm induzierte Metrik; $\|\|x\|\|_\infty = max_i\|x_i\|$
l	Anzahl von Baum-Ebenen

KAPITEL 3

Netzvirtualisierung

Während das Internet in den letzten Jahrzehnten massiv gewachsen ist, so hat sich seine grundlegende Architektur nur in verhältnismäßig geringem Umfang weiterentwickelt. Während es zwar deutliche Weiterentwicklungen beispielsweise bei den zugrundeliegenden Kommunikationstechnologien hin zu immer höheren Bandbreiten gab oder neue innovative Anwendungen entwickelt wurden, so fanden diese Innovationen meist ohne eine Änderung der zentralen Protokolle des Internets statt.

Gerade diese grundlegende Architektur des Internets steht aber vor vielfältigen Herausforderungen beispielsweise im Hinblick auf Skalierbarkeit, Sicherheit, die Unterstützung mobiler Dienste und Dienstgüte, die nicht vollständig auf Basis der heutigen Internet-Architektur umgesetzt werden können [47]. Während es viele innovative Ansätze zur Lösung dieser Probleme gibt [48, 49, 50, 51], so sind diese oft nicht mit der heutigen Internet-Architektur kompatibel. Sind für die Ausbringung eines solchen Ansatzes damit beispielsweise Hardware-Änderungen erforderlich, so ist dies einerseits durch die schiere Größe des Internets schwierig. Zusätzlich wäre eine enge Synchronisation zwischen den durch verschiedene Parteien betriebenen Teilnetzen notwendig, um solche Änderungen zeitgleich einzuführen. Selbst die Einführung des immer dringender benötigten IPv6 verlief bisher und verläuft immer noch schleppend. So waren selbst im Jahr 2013, gut 15 Jahre nach der initialen Standardisierung [52], nur etwa 3% der zehntausend meistbesuchten Webseiten per IPv6 erreichbar und nur etwa 1% der Zugriffe auf Google erfolgte über IPv6 [53].

Ein Ausweg aus diesem inflexiblen, oft auch als Verknöcherung (englisch: Ossification) bezeichneten Zustand und hin zu einem flexibleren Netz ist die sogenannte *Netzvirtualisierung* [54]. Ziel der Netzvirtualisierung als neues grundlegendes Paradigma eines künftigen Internets ist es, den gleichzeitigen Betrieb vieler neuartiger und innovativer Netzarchitekturen nebeneinander zu ermöglichen. Dazu werden gegeneinander isolierte Netze mit jeweils eigenen Netzarchitekturen und -protokollen auf Basis einer gemeinsamen Infrastruktur, dem sogenannten *Substrat*, betrieben. Ähnlich zur Systemvirtualisierung werden dabei jedem *vir-*

tuellen Netz Ressourcen zur Nutzung zugewiesen. Dies umfasst dabei einerseits Ressourcen auf End- und Zwischensystemen wie beispielsweise Routern aber auch Netzressourcen wie Links. Für die Virtualisierung von Links stehen dabei je nach betrachteter Schicht verschiedene Möglichkeiten bereit. Auf der Bitübertragungsschicht können virtuelle Links beispielsweise durch Zeit- oder Frequenzmultiplex realisiert werden; auf der Sicherungsschicht können bei Verwendung von Ethernet zum Beispiel VLANs verwendet werden. Auf der Vermittlungsschicht können beispielsweise Pseudo-Wires [55] verwendet werden. Die grundlegenden Techniken für die Virtualisierung von Netzen sind damit also bereits verfügbar sind.

Während eine zentrale Motivation von Netzvirtualisierung der parallele Betrieb neuer innovativer Architekturen und Protokolle ist, bietet die Einführung von Netzvirtualisierung weitere Vorteile. Betreiber von Hardware, die als Grundlage für virtuelle Netze dient, können ihre *Ressourcen effizient nutzen*, indem sie zwischen verschiedenen virtuellen Netzen statistisches Multiplexing betreiben. Mit der durch Netzvirtualisierung eingeführten Indirektion zwischen virtuellen Ressourcen und physischen Ressourcen ist es beispielsweise auch möglich, ein virtualisiertes System auf andere Hardware zu migrieren [56]. Durch die so erhöhte Flexibilität können *Ausfallzeiten vermieden oder reduziert werden*.

In diesem Kapitel wird die im Rahmen des Projekts 4WARD [1] entwickelte Architektur für Netzvirtualisierung vorgestellt [4, 57], die im Gegensatz zu existierenden Ansätzen nicht nur für den Einsatz in Testumgebungen gedacht ist, sondern die unter Rücksichtnahme auf wirtschaftliche Randbedingungen die notwendigen Rollen und Schnittstellen definiert. Im Projekt 4WARD wurden sogenannte *Clean Slate*-Ansätze [47] untersucht und eine Vision des Projekts war der Betrieb verschiedenartiger Netzarchitekturen und Kommunikationsparadigmen, wie er beispielsweise durch das ebenfalls am Institut für Telematik entwickelte NENA-Rahmenwerk[2] ermöglicht wird, parallel zum Internet [58, 59] . Die im Folgenden beschriebene Architektur für Netzvirtualisierung bildet dafür die Grundlage.

Im Folgenden werden zunächst die identifizierten Rollen mit einer kurzen Motivation und einer Beschreibung ihrer Funktion vorgestellt. Anschließend werden anhand eines Beispiel-Szenarios die erforderlichen Schnittstellen zwischen den Rollen identifiziert. In einem ausführlichen Vergleich werden verwandte Arbeiten aufgearbeitet und daran die Vorteile der hier vorgeschlagenen Architektur aufgezeigt bevor das Kapitel mit einer Zusammenfassung schließt.

3.1 Rollen

Im heutigen Internet unterscheidet man üblicherweise zwischen Internetanbietern und Dienstanbietern. Internetanbieter (englisch: Internet Service Provider, ISP), wie beispielsweise AT&T oder die Deutsche Telekom, stellen die benötigte Hardware zur Verfügung, tragen Sorge für die Konnektivität zu anderen Netzen und bieten Endanwendern Zugang zum Internet an. Dienstanbieter, wie beispielsweise Google, stellen Angebote für Endanwender zur Verfügung. Allein bei diesem einfachen Zwei-Rollen-Modell ist bereits implizit eine dritte Rolle enthalten: Trennt man die Tätigkeit innerhalb eines Internetanbieters weiter auf, so

[1] `http://www.4ward-project.eu`
[2] `http://nena.intend-net.org`

kann man weiter unterscheiden in einerseits den Infrastruktur-Provider, der die Hardware bereitstellt und verwaltet, und den Konnektivitäts-Provider, der für die strategische Ausrichtung und Planung des Netzes zuständig ist. Während diese beiden Rollen zwar in der Praxis oft innerhalb derselben Firma platziert sind, so finden sie sich dort üblicherweise in verschiedenen Abteilungen. Diese Aufteilung in einen Infrastruktur-Provider und einen Konnektivitäts-Provider wird so auch von weiteren Ansätzen propagiert [60, 61].

In Kombination mit der durch Netzvirtualisierung zusätzlich eingeführten Abstraktionsebene, wurden für die hier entwickelte Architektur die folgenden vier Rollen eingeführt [4]:

Infrastruktur-Provider (InP) Der Infrastruktur-Provider besitzt und verwaltet physische Hardware. Mit entsprechenden Techniken kann der InP diese physischen Ressourcen in gegenseitig isolierte virtuelle Ressourcen unterteilen und diese als Dienstleistung anbieten.

Virtual Network Provider (VNP) Der Virtual Network Provider ist für den Aufbau virtueller Netze verantwortlich und kann dazu Ressourcen von einem oder mehreren InPs zu einem virtuellen Netz zusammenfügen.

Virtual Network Operator (VNO) Der Virtual Network Operator ist für den Betrieb und die Wartung eines virtuellen Netzes nach dessen Aufbau durch den VNP zuständig und kann beispielsweise die für eine bestimmte Netzarchitektur notwendige Software installieren und konfigurieren.

Service Provider (SP) Der Service Provider (oder auch Dienstanbieter) nutzt ein virtuelles Netzwerk, um einen Mehrwert-Dienst wie beispielsweise Video-Streaming bereitzustellen.

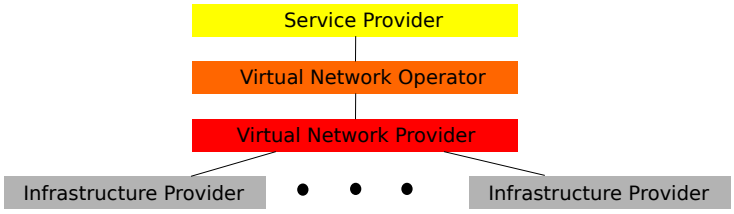


Abbildung 3.1: Rollen der Netzvirtualisierungsarchitektur

Wie bereits zuvor erwähnt, kann ein einzelnes Unternehmen dabei auch mehrere Rollen gleichzeitig einnehmen. Da die Rollen aber üblicherweise unternehmensintern von unterschiedlichen Abteilungen wahrgenommen werden, wurde diese feingranulare Aufteilung vorgenommen. Ein weiterer Vorteil dieser Aufteilung ist auch, dass damit gemäß der „Design for Tussle“-Philosophie keine vorweggenommenen Einschränkungen durch die Architektur gemacht werden, sondern die Architektur sich an praktischen Bedürfnissen und Gegebenheiten orientieren und an diese anpassen kann [62]. Abbildung 3.1 fasst die identifizierten Rollen sowie deren Beziehungen untereinander zusammen.

3.2 Schnittstellen der Architektur

Mit den zuvor eingeführten Rollen ergeben sich die in Abbildung 3.2 dargestellten Schnittstellen zur Erzeugung, Wartung und Nutzung des virtuellen Netzes [63]. Abbildung 3.2 stellt beispielhaft ein Substrat mit drei Infrastruktur-Providern (InP 1, InP 2, InP 3) dar, von denen jeder über virtualisierbare Knoten (Kreise) und Kanten (Verbindungen) verfügt und diese Ressourcen potentiellen Kunden anbietet.

Zur Erzeugung des oberhalb des Substrats abgebildeten virtuellen Netzes übergibt der Service Provider zunächst seine Anforderungen an den Virtual Network Operator. An dieser Schnittstelle ist die Beschreibung des Netzes noch nicht formalisiert sondern besteht üblicherweise aus Beschreibungen der Ziele des Service Providers. Daraus erarbeitet der Virtual Network Operator eine den Anforderungen entsprechende Topologie und beschreibt diese mit detaillierten Randbedingungen, wie beispielsweise den gewünschten Points of Presence, Bandbreiten- und Latenzanforderungen oder Kostenbeschränkungen, in einer gemeinsamen Ressourcenbeschreibungssprache. Dazu kann beispielsweise Flexible Resource Descriptions (FLeRD) [64] verwendet werden.

Diese Beschreibung des gewünschten virtuellen Netzes wird dann dem Virtual Network Provider übergeben (Schnittstelle 1), dessen Aufgabe es ist, von den verfügbaren Infrastruktur-Providern die entsprechenden Ressourcen anzumieten, so dass die Anforderungen erfüllt werden können. Dazu kann der Virtual Network Provider verfügbare Ressourcen von Infrastruktur-Providern abfragen (Schnittstelle 2) und beispielsweise in Abhängigkeit von der geographischen Präsenz der Infrastruktur-Provider das angefragte virtuelle Netz zwischen diesen aufteilen. Dabei muss der Virtual Network Provider das sogenannte Virtual Network Embedding Problem lösen und entscheiden, wie virtuelle Ressourcen auf Substrat-Ressourcen abgebildet werden [65].

Anschließend werden die virtuellen Ressourcen bei den involvierten Infrastruktur-Providern angefordert (Schnittstelle 2). Die Infrastruktur-Provider reservieren die erforderlichen Ressourcen innerhalb ihrer Infrastruktur (Schnittstelle 3) und tragen ebenfalls Sorge dafür, dass die erforderlichen virtuellen Links zwischen den Infrastruktur-Providern aufgebaut werden (Schnittstelle 4).

Nachdem das virtuelle Netz aufgebaut ist, erhält der Virtual Network Operator über einen Out-of-Band Kontrollkanal (Schnittstelle 5) Zugriff auf die einzelnen Ressourcen des virtuellen Netzes. Darüber kann dieser erforderliche Software installieren, diese konfigurieren und während der ganzen Lebenszeit des virtuellen Netzes dessen virtuelle Ressourcen überwachen.

Ist die Konfiguration des virtuellen Netzes abgeschlossen, können sich über die verschiedenen Points of Presence des virtuellen Netzes Endanwender an dieses Netz anbinden (Schnittstelle 6). An dieser Stelle können optional Authentifzierung und Authorisierung der Nutzer durchgeführt und Accounting-Daten erfasst werden.

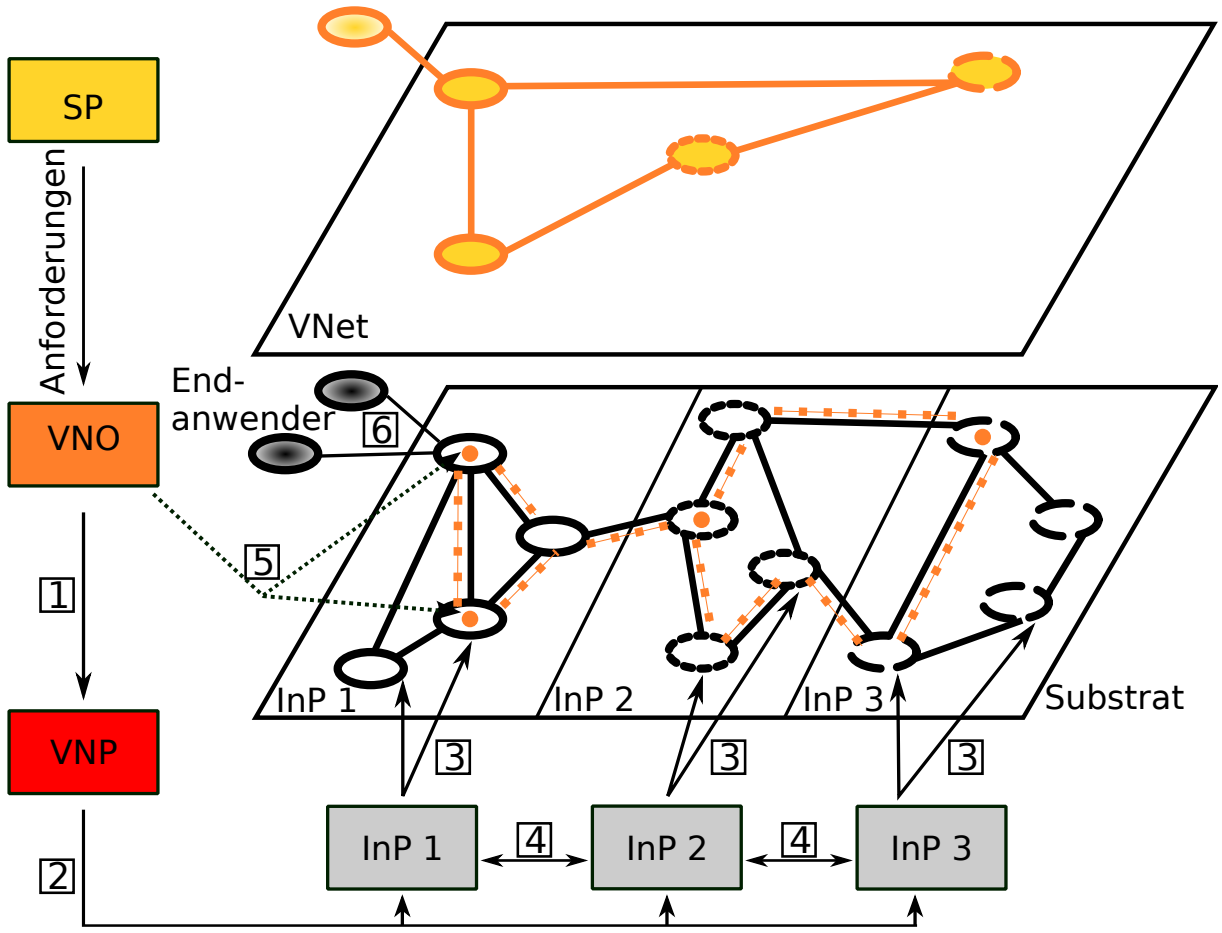


Abbildung 3.2: Beispiel eines virtuellen Netzes und Schnittstellen der Netzvirtualisierungsarchitektur

3.3 Verwandte Arbeiten

Der Entwurf von Architekturen für Netzvirtualisierung war ein sehr aktiver Forschungsbereich der letzten Jahre. Diese Architekturen können grob in zwei Klassen unterschieden werden:

- Architekturen für den Einsatz in Testumgebungen und
- Architekturen für den Einsatz in Produktivumgebungen.

Architekturen für Testumgebungen können dabei teilweise als Vorläufer und Wegbereiter für Architekturen für den Produktiveinsatz betrachtet werden. Da sich virtualisierte Netze in Testumgebungen als extrem nützlich herausgestellt hatten, führen Architekturen für den Produktiveinsatz den nächsten logischen Schritt durch und untersuchen, wie virtualisierte Netze in einem großen Maßstab verfügbar gemacht werden können.

3.3.1 Netzvirtualisierung für Testumgebungen

In den vergangenen Jahren wurden viele Forschungsarbeiten im Bereich künftiger Netze durchgeführt, die teilweise völlig neue Paradigmen für die Kommunikation untersuchen [66]. Diese neuen Ansätze brechen an verschiedenen Stellen mit der Architektur des heutigen Internets und sind daher nicht evolutionär auf Basis des Internets einsetzbar. Stattdessen werden dafür isolierte, vollständig konfigurierbare Netze benötigt. Um die zur Evaluierung der Ansätze notwendige teure Infrastruktur effizient nutzen zu können und zwischen Forschern zu teilen, wurden Architekturen zur Virtualisierung von Testumgebungen entwickelt, von denen im Folgenden einige prominente Vertreter vorgestellt werden.

3.3.1.1 PlanetLab

PlanetLab [67, 68] stellt ein sehr erfolgreiches Beispiel einer großen verteilten Testumgebung dar. Institutionen, die Zugang zu PlanetLab erhalten, stellen eigene Ressourcen zur Verfügung und geben die Kontrolle über diese zunächst an PlanetLab ab. Im Gegenzug erhalten diese Institutionen Zugang zum gesamten PlanetLab und können sich für einen gewissen Zeitraum über ihre eigenen Ressourcen hinaus ein *Slice* genanntes Stück von PlanetLab reservieren und für Experimente nutzen. Institutionen agieren also einerseits als Infrastrukturbetreiber und als Nutzer.

PlanetLab basiert auf einem hierarchischen Vertrauensmodell und die sogenannte *PlanetLab Central* erhält vollständige Kontrolle über die durch eine Institution für PlanetLab bereitgestellten Ressourcen. Während ein solches zentralisiertes Vertrauensmodell für PlanetLab als Testumgebung funktioniert und zur Robustheit beiträgt, ist dieses Modell für den Einsatz in globalen Produktivumgebungen nicht geeignet. Da durch PlanetLab nur eine containerbasierte Virtualisierung bereitgestellt wird und der Netzwerk-Stack selbst nicht virtualisiert wird, stellt PlanetLab keine vollständige Netzvirtualisierung dar.

Vini VINI [69] ist eine Testumgebung, die im Vergleich zu PlanetLab eine tiefergehende Virtualisierung erlaubt. VINI erlaubt die Virtualisierung von Routern und die Verbindung

dieser Router über virtuelle Links. Dadurch wird es möglich, neuartige Netzarchitekturen auszubringen und auf echter Router-Software unter Produktivbedingungen beispielsweise bezüglich des Verkehrs zu evaluieren. VINI unterstützt die parallele Durchführung von Experimenten mit beliebiger Topologie aus Basis einer geteilten Infrastruktur.

Zur Verwaltung der Infrastruktur setzt VINI auf dem zuvor vorgestellten PlanetLab auf und übernimmt damit desse Vor- und Nachteile. VINI erweitert PlanetLab um eine Schnittstelle zur Konfiguration virtueller Links. Eine erste, auf User-Mode Linux [70] basierte Implementierung von VINI bot allerdings nur eine eingeschränkte Leistungsfähigkeit bei der Weiterleitung von Paketen.

Trellis Trellis [71] bietet als Weiterentwicklung von VINI noch eine höhere Leistung bei der Weiterleitung von Paketen. Die Verwaltung ändert sich gegenüber VINI jedoch konzeptionell nicht.

3.3.1.2 Emulab

Emulab [72] ist ebenfalls eine sehr verbreitet eingesetzte Testumgebung. Während zwar sehr gute Werkzeuge zur Verwaltung von virtualisierten Netzen über deren ganze Lebenszeit zur Verfügung gestellt werden, verfügt Emulab über keine Architektur wie die zuvor vorgestellten Ansätze. Emulab ermöglicht die Konfiguration von Topologien anhand von ns-2 [73] Konfigurationsdateien und unterstützt grundlegende Virtualisierungsmechanismen über container-basierte Systemvirtualisierung.

3.3.1.3 GENI

GENI [74] ist eine sehr große amerikanische Forschungsinitiative mit dem Ziel, föderierte virtualisierte Testumgebungen für künftige Netzarchitekturen zu entwickeln. Bei GENI werden alle Aktionen durch ein zentrales *GENI Clearing House* genehmigt und verwaltet, das in etwa dem Virtual Network Provider der hier vorgeschlagenen Architektur entspricht. Während die Unterstützung föderierter Clearing Houses ein Ziel der Initiative ist, wurden dafür noch keine detaillierten Ansätze vorgestellt.

3.3.1.4 Zusammenfassung

Die zuvor vorgestellten Architekturen für Testumgebungen gehen davon aus, dass es eine zentrale, vertraute Entität wie beispielsweise die Planet Lab Central oder die GENI Clearing Houses gibt. Diese Annahme ist so für eine kommerzielle, möglicherweise globale Architektur zur Netzvirtualisierung nicht erfüllt. Wie auch im heutigen Internet muss es für Firmen möglich sein, die vollständige Kontrolle über ihre eigene Infrastruktur zu behalten und ihre Richtlinien innerhalb ihrer eigenen administrativen Bereiche wie auch gegenüber Wettbewerbern umzusetzen. Um diesen Anforderungen gerecht zu werden, wurden in der hier vorgeschlagenen Architektur diese Wettbewerbsverhältnisse berücksichtigt. Entlang der definierten Schnittstellen ist es den Firmen möglich, interne Informationen wie beispielsweise

ihre genaue Topologie oder vorhandene Ressourcen zu verstecken. Eine Beschreibungssprache und entsprechende Algorithmen, die dies ermöglichen, wurden im Rahmen einer weiteren Dissertation [64] entwickelt.

3.3.2 Netzvirtualisierung für Produktivumgebungen

Als Ansätze für Produktivumgebungen wurden CABO und das darauf aufbauende Cabernet entwickelt.

CABO [75] hat zum Ziel, das Ausbringen neuartiger Protokolle und Netzarchitekturen zu beschleunigen, indem es den Betrieb mehrerer paralleler virtueller Netze ermöglicht. Dazu nimmt CABO eine Unterteilung in Infrastrukturanbieter und Dienstanbieter vor. Während Infrastrukturanbieter die Substrat-Ressourcen verwalten, können Dienstanbieter darauf basierend virtuelle Netze aufbauen und darin ihre Dienste betreiben. Sollen ein virtuelles Netz aus den Ressourcen mehrerer Infrastrukturanbieter zusammengesetzt werden, so muss der Dienstanbieter selbst Verträge mit diesen abschließen und sich um die Konnektivität zwischen den Infrastrukturanbietern bemühen.

Cabernet [76] verfeinert diese Idee weiter und führt eine sogenannte Konnektivitätsebene (englisch: Connectivity Layer) ein, die sich zwischen dem Infrastrukturanbieter und Dienstanbieter befindet. Auf dieser Konnektivitätsebene baut ein Konnektivitäts-Provider dann das virtuelle Netz gegebenenfalls aus den Ressourcen mehrerer Infrastrukturanbieter auf und stellt dieses dem Dienstanbieter als ein virtuelles Netz zur Verfügung. Damit muss ein Dienstanbieter nicht mehr Verträge mit mehreren Infrastrukturanbietern abschließen, was die Einstiegshürde für Dienstanbieter erniedrigt.

3.3.2.1 Zusammenfassung

Während die von Cabernet vorgenommene Aufteilung der Rollen zu der hier vorgeschlagenen Architektur vergleichbar ist, ist die hier vorgenommene Aufteilung der Rollen flexibler. Der Konnektivitäts-Provider und der Dienstanbieter von Cabernet wurde hier in die drei Rollen Virtual Network Provider, Virtual Network Operator und Dienstanbieter unterteilt. Dadurch wird die in Unternehmen übliche Unterteilung in die strategische Ausrichtung eines Netzes und den operativen Betrieb auch in den Rollen sichtbar. Während es möglich ist, dass diese Rollen durch ein und dasselbe Unternehmen wahrgenommen werden können und diese Rollen damit zusammenfallen können, ermöglicht die hier Rollenaufteilung der hier vorgestellten Architektur eine größtmögliche Flexibilität.

3.4 Zusammenfassung

In diesem Kapitel wurde eine Architektur zur Netzvirtualisierung vorgestellt. Im Gegensatz zu verwandten Arbeiten wurde dabei eine feingranulare Einteilung der beteiligten Entitäten in die Rollen

- Infrastruktur-Provider

- Virtual Network Provider
- Virtual Network Operator
- Service Provider

vorgenommen. Die Architektur definert dabei die Beziehungen zwischen den beteiligten Entitäten, nimmt aber beispielsweise nicht vorweg, wie diese intern aufgebaut sind. Jede Entität kann also ihre eigenen physischen oder virtuellen Ressourcen nach Belieben verwalten während sich wirtschaftliche Interessenskonflikte entlang der durch die Architektur identifizierten Schnittstellen abspielen können. Die vorgestellte Architektur folgt damit der „Design for Tussle“-Philosophie.

Für diese provider-übergreifende Architektur wurden im Rahmen einer Dissertation [64] eine Beschreibungssprache für Ressourcen und Algorithmen zur effizienten Platzierung virtueller Ressourcen entwickelt. In einer weiteren Arbeit [77] werden virtuelle Netze zur Verbesserung der Kontrolle und Verwaltung von Netzen eingesetzt. Basierend auf dem IETF Next Steps in Signaling (NSIS) Rahmenwerk wurde ein Virtual Link Setup Protokoll (VLSP) entwickelt, das den authentifizierten Aufbau virtueller Links unter Berücksichtigung von Dienstgüteaspekten ermöglicht [78, 79]. Zudem wurde diese Architektur auch von weiteren Forschungsprojekten, wie beispielsweise G-LAB [80] und SECCRIT [81], als Grundlage für die dort durchgeführten Arbeiten verwendet.

Netzvirtualisierung ist eine vielversprechende Möglichkeit, die im Folgenden entwickelten Routing-Ansätze einerseits zu testen und schließlich auch zum praktischen Einsatz zu bringen.

KAPITEL 4

Ein Rahmenwerk zur Untersuchung von Routingverfahren

Da die praktische Evaluierung von Routing-Protokollen, insbesondere auf den hier im Fokus stehenden großen Topologien, einen sehr hohen Aufwand bedeutet oder gar nicht realisierbar ist, hat sich die simulative Untersuchung von Routing-Protokollen in der Praxis als Mittel der Wahl durchgesetzt. Zur Untersuchung von Protokollen und insbesondere von Routingprotokollen gibt es eine Reihe von Simulationsumgebungen, die teils einen sehr spezifischen Fokus aufweisen. Für die detaillierte Simulation des Verhaltens des Border Gateway Protocols kann beispielsweise C-BGP [82] verwendet werden. Zur Untersuchung von Overlay-Netzen steht mit OverSim [83] einen verbreitet eingesetzter Simulator zur Verfügung.

In dieser Arbeit ist zur Evaluierung der im Folgenden entwickelten Ansätze vor allem das algorithmische Verhalten der untersuchten Routing-Algorithmen, beispielsweise in Bezug auf den Stretch oder Nachrichtenaufwand, von Interesse. Bei der Betrachtung des Aufbaus der Routing-Tabellen oder der durch ein Routing-Protokoll bestimmten Pfade spielen unterhalb der Vermittlungsschicht angesiedelte Aspekte, wie die realistische Simulation des Verhaltens von Kommunikationskanälen oder des Medienzugriffs, nur eine untergeordnete Rolle. Da die Simulation dieser Aspekte damit nur beschränkt relevant ist und gleichzeitig einen erheblichen Aufwand in Simulationen bedeutet, wird zur Evaluierung der hier entwickelten Routing-Protokolle ein Simulator mit Fokus auf die Vermittlungsschicht benötigt.

Im Folgenden werden zunächst allgemeine Anforderungen an den Simulator formuliert und anschließend existierende Ansätze anhand dieser Anforderungen bewertet. Da sich bei dieser Analyse ergab, dass keiner der existierenden Ansätze die Anforderungen zufriedenstellend erfüllte, wurde mit RoutingSim [7] ein neues Rahmenwerk zur Untersuchung von Routing-Protokollen entwickelt, um die für diese Arbeit notwendigen Untersuchungen durchführen zu können. Im weiteren Verlauf des Kapitels wird die Architektur von RoutingSim und grundlegende Entwurfsentscheidungen vorgestellt und diskutiert. Das Kapitel schließt mit einer Zusammenfassung.

4.1 Anforderungen

Zunächst werden die an einen Routing-Simulator gestellten Anforderungen aufgestellt. Diese werden unterschieden in *notwendige Anforderungen* an den Routing-Simulator (RS) und *optionale* Anforderungen an den Routing-Simulator (RSO). Anhand dieser Anforderungen werden im folgenden Abschnitt existierende Ansätze bewertet.

RS 1 Um die Untersuchung von umfangreichen Topologien in der Größenordnung der Topologie des Internets auf Router-Ebene zu ermöglichen, muss der Routing-Simulator bezüglich seines Ressourcenverbrauchs in Form von Hauptspeicher und Rechenzeit skalierbar sein.

RS 2 Da der Fokus der Untersuchungen auf dem algorithmischen Verhalten von Routing-Protokollen liegt, soll zugunsten einer erhöhten Skalierbarkeit auf dafür nicht relevante Aspekte, wie beispielsweise die ausführliche Simulation des Medienzugriffs oder gar der Bitübertragungsschicht, verzichtet werden.

RS 3 Die Untersuchung der Routing-Protokolle soll auf Paket-Ebene erfolgen.

RS 4 Der Import gängiger Topologieformate muss unterstützt werden.

RS 5 Zur Untersuchung des Verhaltens eines Routing-Protokolls unter Dynamik muss es möglich sein, eine existierende Topologie zur Laufzeit zu ändern, indem Knoten und Kanten aktiviert oder deaktiviert werden.

RS 6 Die Erfassung routing-relevanter Statistiken wie beispielsweise des Stretch oder von Topologie-Eigenschaften soll durch den Routing-Simulator bereitgestellt werden.

RSO 1 Die Implementierung soll als Open Source Software veröffentlicht sein und idealerweise weit verbreitet eingesetzt werden, um auf ausgereiftem Code aufbauen zu können.

RSO 2 Das Vorhandensein einer graphischen Oberfläche zur Konfiguration, Simulation und Analyse ist wünschenswert aber nicht notwendig

RSO 3 Der Routing-Simulator soll einfach und mit wenig Aufwand über eine schlanke Schnittstelle um neue Protokolle erweiterbar sein.

4.2 Existierende Ansätze

An dieser Stelle werden zunächst existierende Routing-Simulatoren betrachtet. Nach der Feststellung, dass keiner der untersuchten Ansätze die gestellten Anforderungen erfüllt, werden zur Implementierung eines neuen Routing-Simulators ereignisdiskrete Simulationsumgebungen verglichen und schließlich die ausgewählte Simulationsumgebung OMNeT++ im Detail beschrieben.

4.2.1 Existierende Routing-Simulatoren

Als mögliche Routing-Simulatoren wurden der Graph-Theoretic Network Analyzer [84], DRMSim [85] und das INET-Rahmenwerk [86] identifiziert. Diese Ansätze werden hier

knapp vorgestellt und anhand von Tabelle 4.1 bezüglich der zuvor formulierten Anforderungen bewertet.

Graph-Theoretic Network Analyzer (GTNA) ist ein in Java realisiertes modulares Rahmenwerk zur Analyse von Netzen. GTNA besteht aus den vier Modulen *Network*, *Series*, *Metrics* und *Plot*. Das Network-Modul kann verschiedene Arten von Topologien erzeugen und gewisse Topologieformate einlesen. Ebenso wird die zentralisierte Erzeugung der Routinginformationen einiger bekannter P2P Netze wie beispielsweise CAN, Chord und Pastry ermöglicht. Eine Simulation von Routing-Protokollen auf Nachrichten-Ebene ist nicht vorgesehen. Ebenso ist die Analyse von Netzen unter Dynamikeinflüssen nicht Bestandteil von GTNA. Das Series-Modul ermöglicht die Aggregation mehrer Simulationsläufe und das Metrics-Modul stellt grundlegende Metriken, wie beispielsweise den durchschnittlichen Grad oder die Länge der kürzesten Pfade, über das Netz zur Verfügung. Das Plot-Modul ermöglicht über GNU-Plot die Visualisierung der Ergebnisse. Während GTNA gute Möglichkeiten zur Analyse von Topologien bietet, ist es für die hier benötigte Simulation von Routing-Protokollen nicht geeignet.

DRMSim ist ein in Java implementierter Simulator zur Untersuchung von Routing-Verfahren. DRMSim basiert auf dem schlanken, ereignisorientierten Simulator Mascsim [87]. DRMSim ist, wie das im Rahmen dieser Arbeit entwickelte RoutingSim auf die Untersuchung von Routing-Protokollen zugeschnitten und erfüllt alle notwendigen Anforderungen. Durch einen starken Fokus auf die Vermittlungsschicht ist DRMSim sehr skalierbar und ermöglicht über eine schlanke Schnittstelle die Implementierung neuer Routing-Protokolle unter verschiedenen Metriken. Dazu ist es möglich, die Topologie zur Laufzeit zu verändern und damit die Auswirkungen auf das Routing zu untersuchen. Während DRMSim alle notwendigen Anforderungen erfüllt, gibt es kleine Einschränkungen bei den optionalen Anforderungen, da beispielsweise keine graphische Oberfläche zur Verfügung gestellt wird. Zudem basiert DRMSim mit Mascsim auf einem nur eingeschränkt verbreiteten Simulationskern. DRMSim wäre zunächst ein idealer Kandidat für die im Rahmen dieser notwendigen Untersuchungen gewesen, war aber nach Korrespondenz mit den Autoren im Juni 2011 leider noch nicht für die Allgemeinheit verfügbar.

Das **INET-Rahmenwerk** ist ein auf OMNeT++ basiertes, in C++ implementiertes Rahmenwerk zur Untersuchung von im Internet im Einsatz befindlichen Protokollen aller Schichten. Das Rahmenwerk umfasst dabei Implementierungen von Netzwerkgeräten, verschiedenen Medienzugriffsverfahren und erlaubt unter anderem über die Einbindung der Quagga Suite auch die realitätsnahe Simulation von Routing-Protokollen. Durch seine Generalität ist das INET-Rahmenwerk für die hier beabsichtigten Untersuchungen jedoch leider nicht skalierbar genug. Zudem gibt es erst mit dem kürzlich eingeführten Lebenszyklusmodell für Knoten eine anfängliche Unterstützung für die Umsetzung von dynamischen Topologien. Durch das zugrundeliegende OMNeT++ kann das INET-Rahmenwerk von dessen umfangreicher Werkzeug-Unterstützung in Form einer IDE und graphischen Oberflächen für die Konfiguration, Beobachtung der Simulation sowie die statistische Auswertung der Ergebnisse profitieren.

Wie anhand von Tabelle 4.1 ersichtlich ist DRMSim der einzige Routing-Simulator, der die zuvor gestellten Anforderungen umfänglich erfüllt. Während DRMSim inzwischen auch verfügbar gemacht wurde, stand es zum Zeitpunkt der Durchführung dieser Arbeit noch

Tabelle 4.1: Vergleich von Simulatoren für Routingprotokolle anhand der in Abschnitt 4.1 formulierten notwendigen und optionalen Anforderungen

	GTNA	DRMSim	INET/Quagga	RoutingSim
RS 1	✓	✓	×	✓
RS 2	✓	✓	×	✓
RS 3	×	✓	✓	✓
RS 4	✓	✓	✓	✓
RS 5	×	✓	×	✓
RS 6	✓	✓	✓	✓
RSO 1	(✓)	(×)	✓	✓
RSO 2	×	×	✓	✓
RSO 3	(✓)	✓	(✓)	✓

nicht zur Verfügung. Daher wurde zur Evaluierung der in dieser Arbeit entwickelten Ansätze das auf die obigen Anforderungen zugeschnittene, skalierbare Rahmenwerk RoutingSim entwickelt.

Ein Ziel beim Entwurf von RoutingSim war es, so weit wie möglich erprobtem Code wiederzuverwenden. Dazu wurde zunächst die Entscheidung getroffen auf einem existierenden, ereignisdiskreten Simulator aufzusetzen. Die Auswahl dieses ereignisdiskreten Simulators wird im folgenden Abschnitt beschrieben.

4.2.2 Ereignisdiskrete Simulation

Zur Umsetzung eines Rahmenwerks, das allen obengenannten Anforderungen gerecht wird, wurde im Sinne der Skalierbarkeitsanforderung (RS 1) die Entwurfsentscheidung getroffen, auf einem schlanken ereignisdiskreten Simulator aufzusetzen. Ereignisdiskret bedeutet dabei, dass der Zustand der Simulation sich nur durch das Auftreten von Ereignissen wie beispielsweise dem Empfang einer Nachricht in diskreten Zeitschritten ändert. Der gewählte Simulator sollte idealerweise zusätzlich frei verfügbar sein (RSO 1).

Unter diesen Kriterien stehen einige, verbreitet eingesetzte Simulatoren wie beispielsweise die C++-basierten ns-2 [73], ns-3 [88], OMNeT++ [89] oder etwa das Java-basierte JiST [90] zur Verfügung. Eine Vergleichsstudie zwischen diesen ereignisdiskreten Simulatoren [91] hat an einem in allen Simulatoren implementierten Modell gezeigt, dass JiST einen bis zu doppelt so hohen Speicherbedarf wie die anderen Ansätze haben kann. Da erwartet wird, dass der Hauptspeicher in großen Simulationen eine kritische Ressource ist, wurde JiST nicht weiter betrachtet. ns-2 benötigte zur Berechnung des Modells deutlich mehr Zeit als die Simulatoren OMNeT++ und ns-3; daher wurde dieses ebenfalls nicht weiter betrachtet. OMNeT++ und ns-3 dagegen verhielten sich in der Studie sowohl bezüglich der benötigten Simulationszeit als auch bezüglich des Speicherbedarfs vergleichbar gut. Aufgrund vorhandener Erfahrung fiel die Entscheidung zwischen diesen beiden Simulatoren zugunsten des

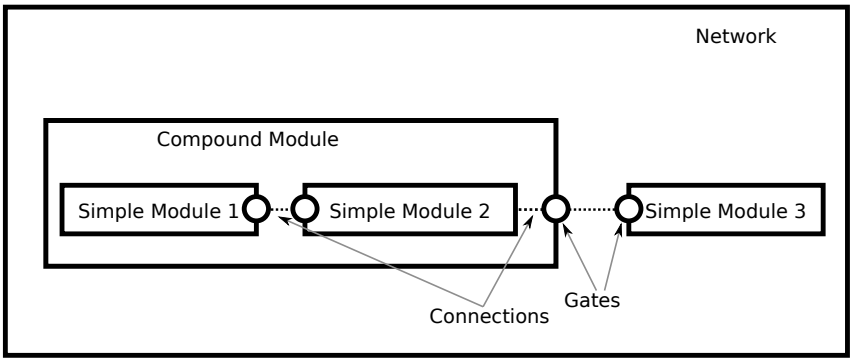


Abbildung 4.1: Grundlegende Abstraktionen von OMNeT++

Objective Modular Network Testbeds OMNeT++ aus, das im Folgenden genauer beschrieben wird.

4.2.3 OMNeT++

Die grundlegenden Abstraktionen, aus denen sich eine Simulation mit OMNeT++ zusammensetzt, stellt Abbildung 4.1 dar. *Simple Modules* stellen die kleinsten aktiven Elemente der Simulation dar und können ihrerseits zu komplexeren *Compound Modules* zusammengesetzt werden. Im Folgenden werden sowohl Simple Modules als auch Compound Modules einfach als Module bezeichnet, wenn keine Unterscheidung notwendig ist. *Gates* stellen die Ein- und Ausgänge für die Kommunikation zwischen Modulen dar und können durch *Connections* verbunden werden. Wie Module untereinander verbunden sind, definiert das *Network*.

Anhand der OMNeT++-eigenen Sprache Network Definition Language (NDL) können die Verbindungen der Module untereinander, also das Network, beschrieben werden. Über diese Verbindungen können *Nachrichten* zwischen den Modulen ausgetauscht werden. Der Austausch von Nachrichten erfolgt dabei, wie für ereignisorientierte Simulation üblich, durch das Einfügen von Nachrichten in Verbindung mit deren Fälligkeitszeitpunkt in eine Liste künftiger Ereignisse, das sogenannte Future Event Set (FES). In der Ereignisschleife der Simulation werden dann die Nachrichten im Future Event Set entsprechend ihres Fälligkeitszeitpunktes abgearbeitet und können während ihrer Abarbeitung beispielsweise den Zustand von Modulen beeinflussen oder das Hinzufügen weiterer Ereignisse zum FES auslösen. Zeitgeber werden in OMNeT++ beispielsweise realisiert, indem Knoten eine Nachricht mit dem gewünschten Fälligkeitszeitpunkt an sich selbst senden.

OMNeT++ ist aus mehreren Gründen eine geeignete Grundlage zur Realisierung des hier entwickelten RoutingSim: Durch die schlanken Basisabstraktionen ist OMNeT++ einerseits sehr flexibel und andererseits skalierbar. Der Overhead, den OMNeT++ gegebenenfalls gegenüber einem hochspezialisierten, neu zu entwickelnden ereignisorientierten Simulator verursacht, wird durch die umfangreiche Werkzeugunterstützung bei der Entwicklung kompensiert.

Ein Grund für die Auswahl von OMNeT++ als Grundlage von RoutingSim war dessen ausgezeichnete Unterstützung des Protokollentwurfs mit Werkzeugen, von der auch RoutingSim di-

rekt profitieren kann. Während der Implementierungsphase unterstützt eine Eclipse-basierte Entwicklungsumgebung den Programmierer. Nach der Definition neuer Nachrichtenformate in einer intuitiven Beschreibungssprache kann der zum Umgang mit diesen Nachrichten notwendige Code automatisch generiert werden. Die Beschreibung der Struktur der Simulation erfolgt über die NDL und es wird auch eine graphische Bearbeitung der Struktur ermöglicht. Die Konfiguration der Modulparameter sowie Simulationen erfolgt über Konfigurationsdateien, die ebenfalls graphisch editiert werden können. Simulationen können dann in der Entwicklungsphase in einer graphischen Oberfläche durchgeführt werden, die eine Beobachtung des Ablauf ermöglicht. Für umfangreiche Simulationen empfiehlt sich nach abgeschlossener Entwicklung die Verwendung des konsolenbasierten Modus, der zugunsten einer erhöhten Leistung auf die aufwändige graphische Darstellung verzichtet. Mit Skalaren, Vektoren und Histogrammen stellt OMNeT++ standardisierte Container zur Statistikerfassung bereit und kann die darin gesammelten Ergebnisse auch auswerten und visualisieren. Für umfangreiche Simulationen und große Datenmengen empfiehlt sich jedoch der Einsatz spezialisierter Software wie beispielsweise dem in dieser Arbeit verwendeten GNU R [92]. OMNeT++ wurde über die letzten Jahre kontinuierlich weiterentwickelt und ist so zu einem hohen Reifegrad gelangt.

4.3 Architektur

Die Architektur und die Komponenten von RoutingSim werden nun anhand von Abbildung 4.2 vorgestellt. Das im unteren Teil der Abbildung dargestellte simulierte Netz besteht aus *Nodes*, die einzelne Router mit ihren Links modellieren und als OMNeT++-Module realisiert sind. Die Nodes entsprechend Routern, die beliebige Routing-Protokolle ausführen können. Vor der Vorstellung der Nodes in Abschnitt 4.3.3 werden aber zunächst die im oberen Teil von Abbildung 4.2 dargestellten Komponenten von RoutingSim beschrieben. Diese umfassen den ebenfalls als OMNeT++-Modul realisierten *NetworkManager*, der für die Verwaltung des erzeugten Netzes während dessen Lebenszeit verantwortlich ist und das Modul *Statistics*, das für die Erfassung von Ergebnissen zuständig ist.

4.3.1 Network Manager

Der *Network Manager* ist für die Verwaltung des zu untersuchenden Netzwerks zuständig. Seine Funktionalität ist dabei unterteilt in die zwei Module *Network Topology* und *Network Dynamics*. Das Modul *Network Topology* erzeugt und verwaltet die zu untersuchende Topologie. Das Modul *Network Dynamics* kann für eine bestehende Topologie Knoten- und Kantenausfälle nach verschiedenen Wahrscheinlichkeitsverteilungen und Lebenszeitmodellen erzeugen.

4.3.1.1 Network Topology

Das Modul *Network Topology* vereint alle zur Verwaltung der Topologie notwendigen Funktionalitäten in sich. Dabei hält das Modul zwei Repräsentationen des Netzwerks:

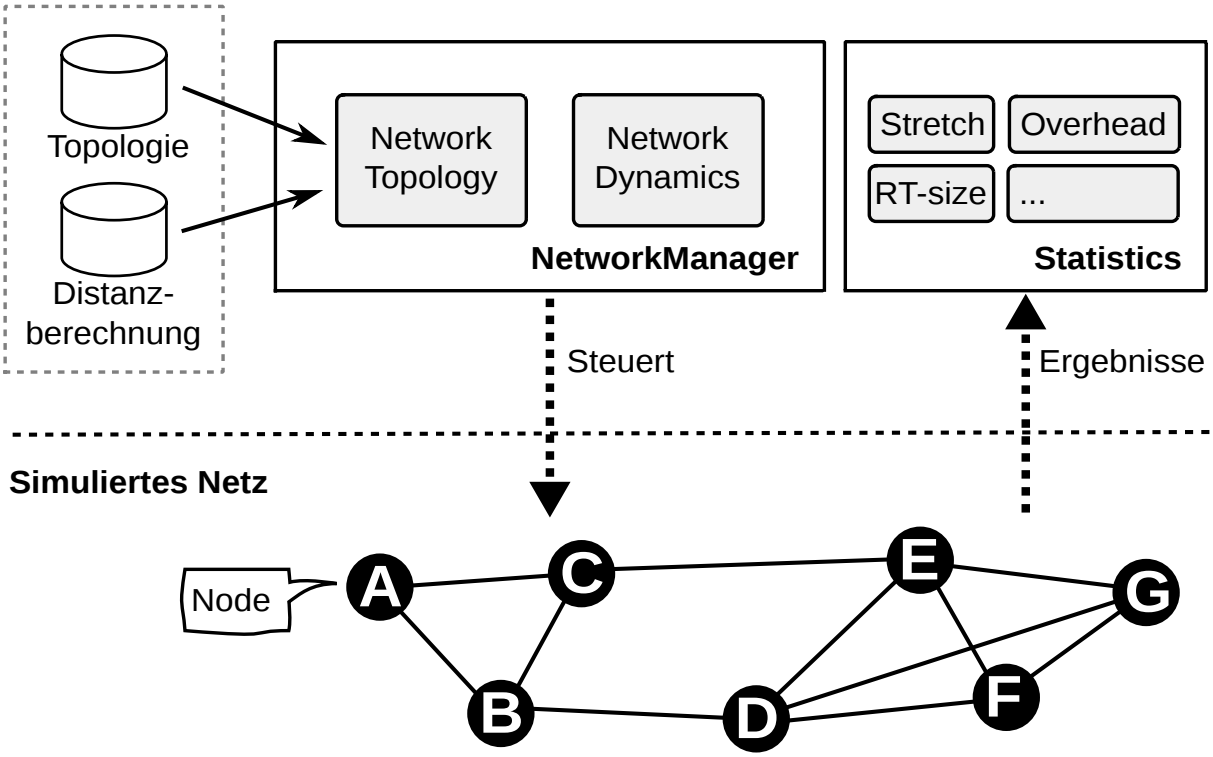


Abbildung 4.2: Übersicht über RoutingSim-Komponenten

- Eine abstrakte Darstellung der Topologie zur Anwendung von Graphenalgorithmen
- und eine damit synchron gehaltene Repräsentation des Netzes aus OMNeT++-Modulen zur Ausführung der Simulation.

Die Datenstruktur für die abstrakte Darstellung der Topologie wurde auf Basis der Boost Graph Library [93] implementiert und ermöglicht die Nutzung der durch die Bibliothek bereitgestellten umfangreichen Algorithmensammlung beispielsweise zur Berechnung kürzester Pfade oder zur Bestimmung von Graphenmetriken. Zwischen der abstrakten Darstellung und dem aus OMNeT++-Modulen, -Gates und -Verbindungen bestehenden zu untersuchenden Netz besteht eine 1:1-Beziehung, die aus beiden Richtungen initialisiert werden kann:

- Wurde das zu untersuchende Netz durch eine Beschreibung in der OMNeT++-eigenen NDL erzeugt, so werden zu Beginn der Simulation die benötigten Module, Gates und Verbindungen durch OMNeT++ erzeugt. NetworkTopology iteriert danach über das Netz und die darin enthaltenen Module und Verbindungen und baut daraus die abstrakte Darstellung auf.
- Wird hingegen eine Topologiebeschreibung durch den NetworkManager aus einer Datei eingelesen, wird daraus zunächst die abstrakte Darstellung initialisiert und anschließend in RoutingSim-eigenen Methoden das zu untersuchende Netz konstruiert.

Zu diesem Zeitpunkt repräsentiert die abstrakte Darstellung die Topologie des zu untersuchenden Netzes. Mit den durch das Modul NetworkTopology bereitgestellten Methoden können nun beide Repräsentationen desselben Netzes synchron manipuliert werden. Wird darüber beispielsweise ein Knoten deaktiviert, so führt dies zu einer Deaktivierung des Kno-

tens in der abstrakten Darstellung wie auch in der laufenden Simulation. Dadurch liefert beispielsweise die Berechnung eines kürzesten Pfades auf der geänderten Topologie die aktuelle Distanz zwischen zwei Knoten und eine eventuell darauf basierende Berechnung des Stretch verwendet damit den korrekten Bezugspunkt. Daneben stellt NetworkTopology viele weitere Funktionen zur Verfügung, die jeweils auf der aktuellen Topologie arbeiten und dadurch die Untersuchung von Routing-Protokollen bei einer sich ändernden Topologie ermöglichen. Beispiele dafür sind die Bestimmung der Knotengradverteilung, ein Sampling von Knoten oder die Bestimmung der Zusammenhangskomponenten.

RoutingSim unterstützt das Einlesen von Topologien, die im DIMES-Format vorliegen sowie ein eigenes Dateiformat; die Erweiterung um weitere Dateiformate ist möglich.

4.3.1.2 Network Dynamics

Das Modul *NetworkDynamics* kann anhand verschiedener Modelle eine bestehende Topologie beeinflussen. Dazu sendet es eine Beschreibung des jeweiligen Dynamik-Ereignisses, also beispielsweise der (De-)Aktivierung eines Knotens oder einer Kante, an das NetworkTopology-Modul, das diese Änderungen entsprechend umsetzt. NetworkDynamics erzeugt die Dynamik-Ereignisse anhand des gewählten Modells und fügt diese zum dadurch bestimmten Zeitpunkt in das Future Event Set ein. Zum Fälligkeitszeitpunkt werden diese dann an das NetworkTopology-Modul ausgeliefert.

Für die im Rahmen dieser Arbeit durchgeführten Untersuchungen wurden zwei Arten von Dynamik betrachtet:

- Der Ausfall eines Anteils von zufällig mit gleicher Wahrscheinlichkeit bestimmten Knoten
- und der Ausfall eines Anteils von zufällig mit gleicher Wahrscheinlichkeit bestimmten Kanten.

Diese beiden Dynamik-Modelle sind innerhalb des Dynamik-Moduls *MixedFailure* realisiert, das neben der Bestimmung der Ausfallszeitpunkte und -dauern die Festlegung des Anteils der zu deaktivierenden Knoten und Kanten ermöglicht.

Daneben stellt RoutingSim weitere Dynamik-Module bereit, die beispielsweise koordinierte Ausfälle oder Churn modellieren.

4.3.2 Statistics

Das Modul *Statistics* ermöglicht eine aggregierte Erhebung von Statistiken. Für die Untersuchung von großen Netzen kann es zur Reduktion der resultierenden Datenmenge wünschenswert sein, die relevanten Metriken nur zu periodischen Sampling-Zeitpunkten zu erheben anstatt jede einzelne Änderung eines Wertes zu erfassen. Das Statistics-Modul bietet diese Funktionalität und sammelt zu jedem Sampling-Zeitpunkt die aggregierten Ergebnisse von registrierten Modulen ein.

Zusätzlich stellt das Modul Statistics eine Schnittstelle zur Bestimmung des Stretch bereit. Beim Versenden von Datenpaketen werden diese beim Statistics-Modul registriert. Wird ein registriertes Paket durch ein Routing-Protokoll zugestellt, so veranlasst das Statistics-Modul

die Berechnung des kürzesten Pfades und bestimmt daraus in Kombination mit der Länge des tatsächlich gewählten Pfades den Stretch des Pakets.

4.3.3 Node und Protocol

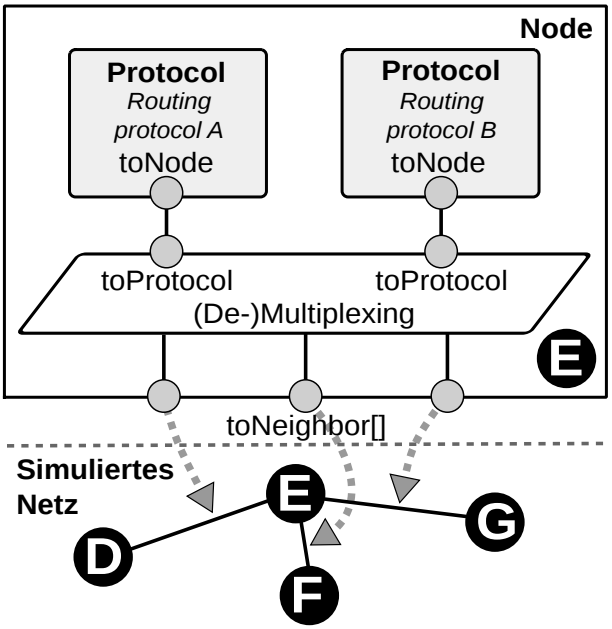


Abbildung 4.3: Abstraktion eines Routers durch das Node-Modul

Die Realisierung eines Routers erfolgt in einem *Node*-Modul. Ein Node-Modul verfügt über einen eindeutigen Identifikator (nodeid_t) und ist, wie in Abbildung 4.3 dargestellt, über den Vektor von Gates *toNeighbors[]* mit weiteren Node-Modulen, also benachbarten Routern, verbunden. Im Beispiel sind dies für Node E die Node-Module D, F und G. Die so verbundenen Node-Module können untereinander Nachrichten austauschen.

Ein Node-Modul wird vom Modul NetworkManager verwaltet, der jederzeit beispielsweise Node-Module sowie einzelne Verbindungen eines Node-Moduls deaktivieren oder wieder aktivieren kann. Innerhalb eines Node-Moduls können verschiedene Routing-Protokolle ausgeführt werden. Routing-Protokolle müssen dazu die Protocol-Schnittstelle implementieren und können dann als Modul innerhalb der Node-Module ausgeführt werden. Die Verbindung zwischen Node- und Protocol-Modulen erfolgt über die Gates *toProtocol* beziehungsweise *toNode*.

Die Protocol-Schnittstelle umfasst die folgenden Funktionen:

void onInitialize() ermöglicht die Initialisierung der Datenstrukturen des Routing-Protokolls

void onDisconnect(nodeid_t nid) Funktion, die bei der Deaktivierung einer Kante zum bisher benachbarten Knoten mit dem Identifikator nid aufgerufen wird und dem Routing-Protokoll die Behandlung dieses Ereignisses ermöglicht.

void onConnect(nodeid_t nid) diese Funktion eines Protocol-Moduls wird aufgerufen, wenn eine deaktivierte Kante zum Nachbarn mit dem Identifkator nid wieder aktiviert wird. Das Routing-Protokoll kann dieses Ereignis dann wie erforderlich behandeln.

void sendToNode(nodeid_t nid, NodePkt* pkt) versendet das zu übergebende Paket an den benachbarten Knoten mit dem Identifikator nid.

void onMessage(NodePkt* pkt) sobald eine Nachricht für das Protocol-Modul beim Node-Modul eintrifft, wird diese über einen Aufruf der onMessage-Methode zur weiteren Verarbeitung übergeben.

void reset() wird beim Ausfall eines Knotens aufgerufen und ermöglicht das Zurücksetzen des Zustandes des Protocol-Moduls ähnlich zu einem Neustart des Knotens.

Diese schlanke Schnittstelle erfüllt zwei Hauptfunktionen:

Nach erfolgter Initialisierung (onInitialize) zu Beginn der Simulation kann ein Protocol-Modul beziehungsweise das darin implementierte Routing-Protokoll beliebig geformte Nachrichten mit Nachbarn austauschen (sendToNode und onMessage). Das Routing-Protokoll kann also seine reguläre Aufgabe wahrnehmen.

Zur Behandlung von Dynamik müssen drei Funktionen implementiert werden, die jederzeit durch den NetworkManager, beispielsweise auf Veranlassung des Moduls NetworkDynamics, aufgerufen werden können. Dazu gehört zum Einen die (De-)aktivierung einzelner Nachbarn oder Kanten (onInitialize, onConnect). Der Totalausfall eines Knotens wird durch ein Deaktivieren aller Kanten des Knotens realisiert und es kann zum Zurücksetzen des Zustands des Knotens zusätzlich die Funktion reset entsprechend implementiert werden.

Durch die leichtgewichtige Protocol-Schnittstelle ist eine zeiteffiziente Implementierung neuer Routing-Protokolle möglich. Diese können durch das RoutingSim-Rahmenwerk unter vergleichbaren Bedingungen und sogar unter dem Einfluss verschiedener Dynamik-Modelle untersucht werden.

4.4 Zusammenfassung

Mit RoutingSim wurde eine skalierbare Simulationsumgebung für Routing-Protokolle entwickelt, die die Untersuchung von großen Netzen mit bis zu 200000 Knoten erlaubt. Auch die im Rahmen dieser Arbeit erforderliche Untersuchung von Knoten- oder Kantenausfällen auf eine Topologie beziehungsweise Routing-Protokolle wird durch RoutingSim ermöglicht. Durch die leichtgewichtige Protocol-Schnittstelle können neue Protokolle zeiteffizient implementiert werden und die umfangreiche Funktionalität des RoutingSim Rahmenwerks zu deren Untersuchung genutzt werden. Da keine vergleichbare Simulationsumgebung zur Verfügung stand, wurde RoutingSim als Open Source Software veröffentlicht[1].

[1] `www.routingsim.org`

KAPITEL 5

Skalierbares Routing mit additiver Stretch-Schranke

Vor dem Hintergrund des kontinuierlichen Routing-Tabellenwachstums stellen kompakte Routing-Verfahren einen interessanten Ansatz für skalierbares Routing dar. Kompakte Routing-Verfahren beschränken die Größe der Routing-Tabellen, so dass diese nur sublinear mit der Anzahl der Ziele n im Netz wachsen. Während dadurch eine Verschlechterung der Pfade in Kauf genommen werden muss, beschränken kompakte Routing-Verfahren den maximalen Stretch durch eine Konstante.

Eine Untersuchung des kompakten Routing-Verfahrens von Brady und Cowen (siehe Abschnitt 2.2.1.5) ist hier besonders interessant, da es ein auf Power-Law Topologien spezialisiertes kompaktes Routing-Verfahren ist. Diese Power-Law Topologien umfassen, wie auch in Abschnitt 2.1.4 erläutert, die Topologie des Internets auf Ebene autonomer Systeme und von Routern.

Das kompakte Routing-Verfahren von Brady und Cowen (BC Routing-Verfahren) ermöglicht durch die Vorgabe des Kerndurchmesser genannten Parameters d eine konfigurierbare additive Beschränkung des maximalen Stretch. Ist ein optimaler Pfad beispielsweise k Übertragungsabschnitte lang, so beträgt die maximal mögliche Pfadlänge bei Verwendung des BC Routing-Verfahrens $k + d$ Übertragungsabschnitte. Das Wachstum der in Abhängigkeit vom Kerndurchmesser d konstruierten Routing-Tabellen ist beim BC Routing-Verfahren beschränkt durch $e_F \cdot \log^2 n$. Dieses Wachstum ergibt sich zum Einen aus einer in Abhängigkeit vom Kerndurchmesser d maximal zu konstruierenden Anzahl von Spannbäumen e_F. Zum Anderen werden den Knoten für jeden dieser Spannbäume zwei Labels der Länge $O(\log^2 n)$ zugewiesen, die die Adresse der Knoten bilden. Das eine Label wird verwendet, um bei der Weiterleitung von Paketen den Baum mit der kürzesten Distanz zum Ziel zu bestimmen (Distanz-Label, siehe Abschnitt 2.1.2), und das andere Label wird zur Weiterleitung von Paketen entlang des bestimmten Baumes genutzt.

Während das BC Routing-Verfahren mit der Spezialisierung auf Power-Law Topologien, der konfigurierbaren additiven Beschränkung des Stretch und der beschränkten Größe der

Routing-Tabellen über positive Eigenschaften verfügt, wurde es nicht als verteiltes Routing-Protokoll entworfen. Wie auch andere kompakte Routing-Verfahren nimmt das BC Routing-Verfahren an, dass die Berechnung der Routing-Tabellen durch eine zentrale Entität vorgenommen wird, die eine vollständige Sicht auf die Topologie des Netzes besitzt und die so berechneten Routing-Tabellen dann auf den Routern installiert werden. Dies steht jedoch im Gegensatz zur Praxis, in der die Routing-Tabellen üblicherweise durch verteilt arbeitende Routing-Protokolle aufgebaut werden.

Während der Aufbau der erforderlichen Spannbäume und die Bestimmung der Labels im zentralisierten Fall einfach ist, gestaltet sich dies im verteilten Fall schwierig und führt damit zu der bereits in der Einleitung aufgeworfenen Frage

V	Kann das zentralisierte Routing-Verfahren von Brady und Cowen als verteiltes Routing-Protokoll umgesetzt werden?

Daran werden hier zwei Folgefragen zur Bewertung des Nutzens einer möglichen Lösung angeschlossen:

V-1	Ist damit auf Internet-ähnlichen Topologien eine Verbesserung des maximalen Stretch gegenüber alternativen aktuellen Ansätzen erreichbar?
V-2	Was sind die Einschränkungen eines solchen Routing-Protokolls?

Mit dem in diesem Kapitel vorgestellten Sprinkles [8] wurde basierend auf dem kompakten Routing-Verfahren für Power-Law Topologien von Brady und Cowen ein Routing-Protokoll entworfen, das die zur Umsetzung der additiven Stretch-Schranke notwendigen Routing-Tabellen verteilt konstruiert. Sprinkles ersetzt die beiden zentralisierten Algorithmen zur Bestimmung der Labels des BC Routing-Verfahrens durch ein verteiltes Verfahren. Während dadurch die Beschränkung von $O(\log^2 n)$ der Label-Länge aufgegeben wird, erreicht das gewählte verteilte Verfahren in der Praxis auf Internet-ähnlichen Topologien eine Länge der zugewiesenen Labels von $O(\log^3 n)$. Dadurch wird die Forderung nach sublinear wachsenden Routing-Tabellen nicht berührt.

In Abschnitt 5.1 werden zunächst Entwurfsziele für das in dieser Arbeit entwickelte Routing-Protokoll Sprinkles formuliert. Die verteilte Konstruktion der Routing-Tabellen durch Sprinkles erläutert Abschnitt 5.2, in dem darauf aufbauend auch Optimierungen zur Reduktion der Adresslänge vorgestellt werden. Für die optimierte Version von Sprinkles wird der Beweis erbracht, dass selbst dann die Einhaltung der Stretch-Schranke immer gegeben ist. In der in Abschnitt 5.3 beschriebenen Evaluierung wird Sprinkles auf synthetischen und realistischen Internet-ähnlichen Topologien mit bis zu zweihunderttausend Knoten untersucht sowie eine Bewertung der erreichbaren Stretch-Schranke vorgenommen. Das Kapitel schließt mit einer Zusammenfassung, in der die eingangs formulierten Fragestellungen wieder aufgegriffen und abschließend beantwortet werden.

5.1 Zielsetzung

Sprinkles soll als verteiltes Routing-Protokoll die additive Stretch-Schranke des kompakten Routing-Verfahrens von Brady und Cowen in die Praxis überführen. Der Fokus ist nicht darauf gerichtet, aktuell im Einsatz befindliche Protokolle wie OSPF oder BGP zu ersetzen, sondern zu untersuchen, inwiefern eine verteilte Umsetzung überhaupt möglich ist und welche Kompromisse dazu notwendig sind.

Die Ziele bei der Entwicklung von Sprinkles umfassten dabei die folgenden Punkte:

- Die Konstruktion von Routing-Tabellen soll ausgehenend von einem uninitialisierten Netz erfolgen (Bootstrapping). Knoten kennen zunächst nur ihren eigenen Identifikator, den ihrer direkten Nachbarn und eine netzweit vorgegeben Konstante d, auf die der additive Stretch beschränkt werden soll.
- Da den Knoten während des Aufbaus der Routing-Tabellen lokationsabhängige Adressen zugewiesen werden, die einerseits in den Routing-Tabellen und andererseits in den Paketköpfen vorkommen, ist es wichtig, die Länge dieser Adressen möglichst klein zu halten. Verfahren zur Reduktion der Adresslänge dürfen jedoch die in den Adressen gespeicherte Information nur insoweit komprimieren als dass dadurch eine Verletzung der Stretch-Schranke ausgeschlossen ist.
- Neben einer Beschränkung des maximalen Stretch ist auch ein niedriger durchschnittlicher Stretch wünschenswert.
- Der beim Aufbau der Routing-Tabellen durch Protokoll-Nachrichten verursachte Nachrichtenaufwand soll möglichst gering sein.
- Beim Aufbau der Routing-Tabellen wird zunächst ein statisches Netz betrachtet, das frei von Knoten- und Kantenausfällen ist. Eine Möglichkeit zur Behandlung von Dynamik, die nicht nur auf Sprinkles sondern auf die gesamte Klasse spannbaum-basierter gieriger Routing-Protokolle (siehe Abschnitt 2.2.2.1) anwendbar ist, stellt das folgende Kapitel 6 vor.
- Die Behandlung von Richtlinien für Routing wird im Rahmen dieser Arbeit nicht untersucht.
- Die additive Stretch-Schranke d kann nicht beliebig reduziert werden, da andernfalls die Größe der Routing-Tabellen stark wächst. Daher müssen für ein breites Spektrum von Topologien praktikable Parameterbereiche für d identifiziert werden. Darauf basierend kann dann für ein gegebenes Netz später auch eine gute Parametrisierung für d bestimmt werden.

5.2 Funktionsweise von Sprinkles

Die Herausforderungen auf dem Weg zu einem verteilten Routing-Protokoll lassen sich zunächst in zwei Teilbereiche zerlegen. Einerseits müssen die für die Einhaltung der Stretch-Schranke notwendigen Spannbäume ohne eine zentralisierte globale Sicht auf den Graphen aufgebaut werden. Auf der anderen Seite ist es erforderlich, innerhalb dieser Spannbäume die für eine Weiterleitung von Paketen notwendigen Adressen zu vergeben. Dies umfasst im

zentralisierten Fall die Konstruktion von zwei Label der Länge $O(\log^2 n)$ für jeden Knoten in jedem Spannbaum. Das eine Label erlaubt dabei die Bestimmung der exakten Distanz zwischen zwei beliebigen Knoten innerhalb eines Spannbaums, während das andere für die Weiterleitung innerhalb eines Baumes genutzt wird. Damit kann beim Sender über das Distanz-Label der Spannbaum mit der geringsten Entfernung zum Ziel bestimmt werden und ein Paket anschließend entlang dieses Baumes zum Empfänger zugestellt werden.

Im Folgenden wird zunächst eine Analyse des Baum-Aufbaus durch das BC Routing-Verfahren durchgeführt. Danach werden die beiden Teilprobleme des Spannbaum-Aufbaus und der Adressvergabe betrachtet und der jeweilige Lösungsansatz von Sprinkles vorgestellt. Anschließend wird eine Erweiterung für Sprinkles zur Reduktion der Adresslänge entwickelt und gezeigt, dass damit die additive Stretch-Schranke stets eingehalten wird.

5.2.1 Analyse der Baumkonstruktion durch das BC Routing-Verfahren

Bevor die Konstruktion der notwendigen Spannbäume durch Sprinkles vorgestellt wird, wiederholt die folgende Zusammenfassung kurz den Aufbau der Spannbäume im BC Routing-Verfahren (siehe Abschnitt 2.2.1) und verweist auf die entsprechende Lösung durch Sprinkles. Bei gegebenem Graphen $G = (V, E)$ und additiver Stretch-Schranke d erfolgt der Aufbau folgendermaßen:

1. Partitionierung des Graphen in einen d-Kern C und den Rand F (siehe Abschnitt 5.2.2.1).
2. Konstruktion des Hauptspannbaums T_h vom Knoten h mit dem höchsten Knotengrad aus (siehe Abschnitt 5.2.2.1). Der Hauptspannbaum umfasst den gesamten Graphen G; es sind also alle Knoten V Teil von T_h.
3. $T_h \cap F$ wird so lange um Kanten erweitert, bis auf jedem Randbereich F_i von F ein Spannbaum entstanden ist, der diesen vollständig umfasst (siehe Abschnitt 5.2.2.2).
4. In dem resultierenden Wald T_F werden nun die Extrakanten E' identifiziert, also Kanten des Graphen, die im Randbereich F liegen, aber nicht Teil von T_F sind. Für jede Kante $e = (u, v)$ in E' wird ein Spannbaum mit einem der beiden Knoten als Wurzel über G konstruiert (siehe Abschnitt 5.2.2.3).
5. Als Heuristik zur Reduktion des durchschnittlichen Stretchs werden beim Vorhandensein von weniger als 5 Extrakanten-Spannbäumen bis zu 5 Extrakanten-Spannbäume mit einer zufällig im Randbereich gewählten Wurzel erzeugt (siehe Abschnitt 5.2.2.4).

Eine erste Beobachtung, die sich anhand des Aufbaus machen lässt, ist die Abhängigkeit der durch das BC Routing-Verfahren konstruierten Spannbäume:

- Die auf den Randbereichen aufgebauten Spannbäume T_F sind über den Kernspannbaum T_h und den Rand F definiert.
- Die für die Konstruktion der Extrakanten-Spannbäume entscheidenden Extrakanten $E' = \{(u, v) | (u, v) \in E; (u, v) \notin T_F; u, v \in F\}$ sind wiederum in Abhängigkeit von T_F definiert.

Auch in einer verteilten Konstruktion ist diese Reihenfolge durch die Abhängigkeit daher erforderlich.

Eine weitere Beobachtung betrifft die beim Aufbau der Routing-Tabellen durch das BC Routing-Verfahren zu Beginn vorgenommene Partitionierung des Graphen in Kern und Rand. Diese Partitionierung kann auch beim Aufbau des Spannbaums T_h vorgenommen werden. Über den Abstand zur Wurzel h des Spannbaums T_h und den netzweit bekannten Kerndurchmesser d können Knoten entscheiden, ob sie im Kern oder im Rand des Graphen liegen.

In Abhängigkeit von der Topologie kann die Wahl eines kleinen Kerndurchmessers zu hohen Adresslängen führen, und es ist wichtig, die Adresslänge wo möglich zu reduzieren. Für die Einhaltung der Stretch-Schranke in Randbereichen ist es nicht erforderlich, dass Extrakanten-Spannbäume wie beim BC-Routing-Verfahren den gesamten Graphen G umfassen. Stattdessen genügt es, die Extrakanten-Spannbäume auf den Randbereich zu beschränken, in dem sie ihre Wurzel haben. Eine Motivation, die Extrakanten-Spannbäume über den gesamten Graphen aufzubauen, war vermutlich die Reduktion des durchschnittlichen Stretch, wie auch aus der verwendeten Heuristik ersichtlich. Um die Reduktion des durchschnittlichen Stretch von der Konstruktion der Extrakanten-Spannbäume zu entkoppeln, werden für Sprinkles unabhängig von der Einhaltung der Stretch-Schranke konstruierbare *zusätzliche Spannbäume* eingeführt.

Ein Knoten kann damit maximal in den folgenden Spannbäumen enthalten sein, wobei zwischen den Bäumen die zuvor identifizierten Abhängigkeiten gelten:

1. dem Hauptspannbaum T_h
2. einem Randspannbaum aus T_F
3. allen Extrakanten-Spannbäumen T_E
4. zusätzlichen Spannbäumen

Im folgenden Abschnitt wird die verteilte Partitionierung des Graphen und der Aufbau der Spannbäume beschrieben.

5.2.2 Aufbau der Spannbäume

Sprinkles definiert zwei Betriebs-Modi, die sich nur in der Konstruktion der Extrakanten-Spannbäume unterscheiden:

Dense Mode Der Dense Mode konstruiert die Extrakanten-Spannbäume vergleichbar zum BC Routing-Verfahren. Allerdings wird hier, wie zuvor motiviert, darauf verzichtet, die Extrakanten-Spannbäume über den gesamten Graphen aufzubauen. Stattdessen umfassen diese zur Reduktion der Adresslängen nur den Randbereich, in dem ihre jeweilige Wurzel liegt.

Sparse Mode Der Sparse Mode verfolgt eine weitere Reduktion der Adresslänge und reduziert dazu unter Einhaltung der Stretch-Schranke die Anzahl der Extrakanten-Spannbäume.

Die Unterschiede zwischen beiden Modi werden bei der Konstruktion der Extrakanten-Spannbäume in Abschnitt 5.2.2.3 beschrieben; zunächst aber wird das Bootstrapping von Sprinkles ausgehend von einem uninitialisierten Netz beschrieben.

Um die Abhängigkeiten zwischen den Spannbäumen zu handhaben, konstruiert Sprinkles diese in der zuvor identifizierten Reihenfolge und trennt die Konstruktionen durch **zeitliche Schutzabstände** voneinander. Die zeitlichen Schutzabstände sorgen dabei dafür, dass der jeweils vorausgesetzte Baum konstruiert wurde, bevor die Konstruktion der davon abhängigen Bäume begonnen wird. Die Realisierung dieser Schutzabstände erfolgt durch einen Zeitgeber, der (wieder-)gestartet wird, wenn sich ein vorausgesetzter Baum ändert. Findet während des zeitlichen Schutzabstandes keine Änderung mehr statt und läuft daher der Zeitgeber ab, so wird die Konstruktion des abhängigen Baumes gestartet. Die Dauer der Schutzabstände ist abhängig von der Topologie des Netzes beziehungsweise des betroffenen Bereichs des Netzes sowie der erwarteten Latenz des betrachteten Netzes zu wählen.

Im folgenden wird entsprechend der Abhängigkeiten zunächst die Konstruktion des Hauptspannbaumes und die damit einhergehende Partitionierung des Graphen in Kern und Rand beschrieben. Anschließend erfolgt die Konstruktion der Randspannbäume in Abschnitt 5.2.2.2 und die Konstruktion der Extrakanten-Spannbäume in Abschnitt 5.2.2.3. Nach einer Beschreibung des Aufbaus der zusätzlichen Bäume in Abschnitt 5.2.2.4 schließt dieser Abschnitt mit einer Zusammenfassung.

5.2.2.1 Hauptspannbaum

Der Hauptspannbaum dient der Sicherstellung der gegenseitigen Erreichbarkeit aller Knoten und wird von Sprinkles auch dazu verwendet, die Einteilung des Graphen in den Kernbereich C und den Rand F vorzunehmen.

Der Aufbau des Hauptspannbaums T_h vom höchstgradigen Knoten h aus wird unter Anpassung des in Abschnitt 2.2.2.2 beschriebenen Tree Maintainers von PIE vorgenommen. Um zwischen den bei Sprinkles unterschiedenen vier Baumtypen differenzieren zu können, wird in jeder Baum-Nachricht die Art des Baumes (hier: Hauptspannbaum) und zusätzlich der Knoten-Identifikator der Wurzel mitgeführt.

Im hier betrachteten Bootstrapping-Szenario nimmt zunächst jeder Knoten an, dass er selbst die Wurzel von T_h sei und versendet entsprechende Baum-Nachrichten an seine Nachbarn. Da als Root-ID der Knotengrad verwendet wird und höhere Root-IDs bevorzugt werden, setzt sich schließlich h als eindeutige Wurzel durch und der Spannbaum T_h ist aufgebaut.

Der durch diese Lösung verursachte Nachrichtenaufwand ist aus zwei Gründen nicht sehr hoch:

- Der Durchmesser Internet-ähnlicher Topologien wächst logarithmisch mit der Anzahl der Knoten. Dies bedeutet, dass spätestens nach einer logarithmisch beschränkten Anzahl von Übertragungsabschnitten jeder Knoten h bzw. T_h kennt. Damit setzt sich der im Kern des Netzes liegende Knoten h mit dem höchsten Grad schnell gegenüber Knoten mit einem kleineren Grad durch. Selbst bei der größten hier betrachteten Topologie mit 200000 Knoten, die einen Durchmesser von 26 Übertragungsabschnitten aufweist, würde sich der im schlechtesten Fall am Netzrand liegende höchstgradige Knoten bei angenommenen 100ms Latenz pro Übertragungsabschnitt nach 2.6s durchsetzen.
- In Internet-ähnlichen Topologien gibt es üblicherweise einen Knoten, der sich durch einen besonders hohen Grad auszeichnet. Auf der DIMES-Topologie, die das Internet

auf Ebene autonomer Systeme modelliert, haben die höchstgradigen fünf Knoten je 5308, 4027, 2860, 2812 bzw. 2515 Nachbarn. Dadurch weist der höchstgradige Knoten in Internet-ähnlichen Topologien eine gewisse Stabilität gegenüber beispielsweise temporären Link-Ausfällen auf, die sonst zu einer wiederkehrenden Neubestimmung führen könnten.

Wenn während eines Schutzzeitraums δ_h keine Änderungen des Hauptspannbaums T_h mehr aufgetreten sind, so können Knoten aufgrund einer rein lokalen Entscheidung annehmen, dass T_h der korrekte Hauptspannbaum ist. Davon ausgehend kann jeder Knoten durch Kenntnis des Kerndurchmessers d sowie des eigenen Abstands h bestimmen, ob er Teil des Kerns oder Teil des Randbereichs ist. Damit kann gleichzeitig die Konstruktion der Rand-Spannbäume beginnen.

5.2.2.2 Rand-Spannbäume

Auch mit nun bekanntem Rand, Kern und Hauptspannbaum ist die Konstruktion der Randspannbäume, wie durch das BC Routing-Verfahren beschrieben, verteilt nicht praktikabel. Die dort vorgenommene Konstruktion von T_F durch Erweiterung des Schnitts $T_h \cap F$ zu Spannbäumen würde ohne globale Sicht eine aufwändige Schleifenerkennung erfordern.

Da die Aufgabe der Rand-Spannbäume lediglich die Bereitstellung von Konnektivität innerhalb der verbundenen Randbereiche ist, wird durch Sprinkles zu diesem Zweck ein neuer Spannbaum pro Randbereich erzeugt, der sich nur über den jeweiligen Randbereich erstreckt.

Dazu muss verteilt in jedem Randbereich F_i eine eindeutige Wurzel des Spannbaums T_{F_i} bestimmt werden. Um dies zu bewerkstelligen und die Anzahl möglicher Wurzeln zur Reduktion des Nachrichtenaufwandes möglichst gering zu halten, sind in Sprinkles all diejenigen Knoten eines Randbereichs F_i Kandidaten für die Wurzel von T_{F_i}, die in F_i liegen, aber zusätzlich über eine Kante in den Kern verfügen. Es werden als nur Knoten von F_i am Rand zum Kern C berücksichtigt.

Diese Wurzelkandidaten initiieren nach Ablauf von δ_h den Aufbau der Randspannbäume T_F. Der Typ des Spannbaums (Randspannbaum) wird wieder in die Baum-Nachrichten aufgenommen und die Nachrichten nur innerhalb eines Randbereichs weitergeleitet. Dies ist möglich, da jeder Knoten weiß, ob er Teil des Randes oder Teil des Kerns ist. Von diesen Knoten setzt sich über die Root-ID wiederum der Knoten mit dem höchsten Grad durch, und es entsteht ein verbundener Spannbaum T_{F_i} pro verbundenem Randbereich F_i.

5.2.2.3 Extrakanten-Spannbäume

Bevor die Konstruktion der Extrakanten-Spannbäume beginnt, muss sichergestellt werden, dass der vorangegangene Aufbau des jeweiligen Rand-Spannbaums erfolgreich abgeschlossen wurde. Dazu wird erneut ein Schutzzeitraum δ_F verwendet. Beobachtet ein Knoten in einem Randbereich F_i während eines Zeitraums δ_F keine Veränderung des Randspannbaums T_{F_i}, so nimmt er an, dass der Randspannbaum erfolgreich bestimmt wurde und beginnt mit der Konstruktion der Extrakanten-Spannbäume.

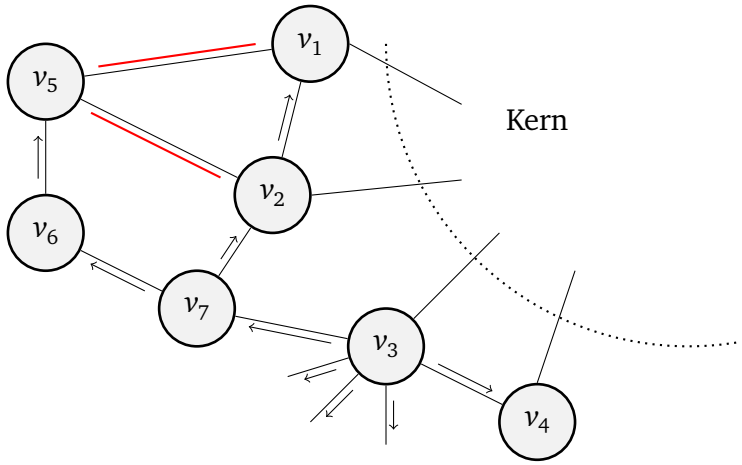


Abbildung 5.1: Auswahl von Wurzeln für Extrakanten-Spannbäume (Dense Mode)

Für die Konstruktion der Extrakanten-Spannbäume wird die eingangs erwähnte Unterscheidung in den BC-ähnlichen Dense Mode und den zur Reduktion der Adresslängen eingeführten Sparse Mode vorgenommen.

Dense Mode Im Dense Mode bestimmt jeder Knoten zunächst unter Ausschluss von T_h den Typ aller seiner Kanten innerhalb des jeweiligen Randbereichs. Kanten, die in den Kern führen, werden dabei nicht betrachtet.

Im Dense Mode bestehen für jede Kante die folgenden Möglichkeiten:

- Eine Kante ist durch den jeweiligen Randspannbaum aus T_F bereits abgedeckt und damit keine Extrakante.
- Für eine Kante (u, v) ist entweder u oder v Wurzel eines Extrakanten-Spannbaums, der diese Kante abdeckt; damit ist die Erzeugung eines zusätzlichen Extrakanten-Spannbaums wegen dieser Kante nicht notwendig.
- Sonst ist die Kante nicht abgedeckt und die Erzeugung eines weiteren Extrakanten-Spannbaums ist erforderlich.

Abbildung 5.1 zeigt dies an einem Beispiel. Knoten v_3 wurde im betrachteten Randbereich als Wurzel bestimmt und hat, wie durch die schwarzen Pfeile neben den Kanten angedeutet, den Rand-Spannbaum aufgebaut. Nach Ablauf von δ_F prüft jeder Knoten den Zustand seiner Kanten. In diesem Beispiel deckt der Rand-Spannbaum für alle Knoten bis auf v_1, v_2 und v_5 alle Kanten ab. Damit ist die Erzeugung von Extrakanten-Spannbäumen durch diese Knoten nicht erforderlich. Die Knoten v_1 und v_2 stellen hingegen fest, dass sie über eine Extrakante verfügen. Knoten v_5 besitzt 2 Extrakanten.

Für die Erzeugung von Extrakanten-Spannbäumen gibt es je nach zeitlichem Ablauf die folgenden Möglichkeiten:

1. Alle Knoten erkennen das Vorhandensein der Extrakanten gleichzeitig und erzeugen jeweils einen neuen Extrakanten-Spannbaum. Dies resultiert in drei Extrakanten-Spannbäumen.
2. Für die Knoten v_1 und v_2 läuft δ_F zuerst ab. Diese erzeugen jeweils einen Extrakanten-

Spannbaum. Nach dem Aufbau der beiden Spannbäume verfügt Knoten v_5 über keine Extrakanten mehr und muss folglich auch keinen zusätzlichen Extrakanten-Spannbaum erzeugen. Insgesamt werden hier also zwei Extrakanten-Spannbäume erzeugt.

3. δ_F läuft zuerst für Knoten v_5 ab und dieser erzeugt einen Extrakanten-Spannbaum. Der Extrakanten-Spannbaum deckt die Extrakanten der Knoten v_1 und v_2 ab und diese müssen daher keine weiteren Extrakanten-Spannbäume erzeugen. Hier wird nur ein Extrakanten-Spannbaum erzeugt, was auch das Minimum in diesem Beispiel darstellt.

Aus diesem Beispiel lassen sich zwei Folgerungen ableiten:

- Die gleichzeitige Bestimmung der Wurzeln führt zur maximalen Anzahl an Extrakanten-Bäumen. Während es genügt, wenn einer der an eine Extrakante angrenzenden Knoten zur Wurzel eines Extrakanten-Spannbaums wird, kann daraus bei gleichzeitiger Bestimmung kein Vorteil gezogen werden. Daher ist es wünschenswert, einen **Jitter in die Bestimmung der Wurzeln einzuführen**.
- Einen Knoten mit einer hohen Anzahl an Extrakanten, wie Knoten v_5 im vorangegangenen Beispiel, als Wurzel zu wählen, kann die Anzahl insgesamt erzeugter Extrakanten-Spannbäume reduzieren. Daher wird eine Verzögerungsfunktion benötigt, die eine **Verzögerung in Abhängigkeit von der Anzahl Extrakanten eines Knotens** bestimmt. Knoten mit vielen Extrakanten sollen vor Knoten mit wenigen Extrakanten zur Wurzel eines Extrakanten-Spannbaums werden.

Um lokal zu bestimmen, ob ein Knoten Wurzel eines Extrakanten-Spannbaums wird, wurde daher die folgende, in Abbildung 5.2 beispielhaft skizzierte Verzögerungsfunktion gewählt. Diese bestimmt nach Ablauf von δ_F in Abhängigkeit von der Anzahl der lokal vorhandenen Extrakanten, wie lange ein Knoten abwartet, bevor er die Wurzel eines Extrakanten-Spannbaums wird.

$$\max(0, \delta^{\max}_{\text{Dense}} - \alpha_{\text{Dense}} \cdot \text{Anzahl Extrakanten}) + \text{runif}(0, \epsilon_{\text{Dense}})$$

Dabei haben die Variablen die folgende Bedeutung:

- $\delta^{\max}_{\text{Dense}}$ ist die maximale zusätzliche Verzögerung und betrug in den später durchgeführten Simulationen 5,5s.
- α_{Dense} ist die Reduktion der maximalen Verzögerung pro vorhandener Extrakante eines Knotens und wurde in den Simulationen mit 0,5s gewählt.
- $\text{runif}(0, \epsilon_{\text{Dense}})$ bestimmt durch die Funktion *runif* den Jitter gleichverteilt zufällig zwischen 0 und dem maximalen Jitter ϵ_{Dense}. In der Evaluierung wurde ϵ_{Dense} auf 0,5s gesetzt.

Reduziert sich die Anzahl der Extrakanten eines wartenden Knotens auf 0, so bricht dieser den Wartevorgang ab und wird nicht zur Wurzel, da dies durch die inzwischen erzeugten Extrakanten-Spannbäume nicht mehr erforderlich ist.

Die für die spätere Evaluierung gewählte Parametrisierung des Dense Mode straft Knoten mit weniger als 11 Extrakanten mit einer zusätzlichen Verzögerung von bis zu 5s bei nur einer Extrakante.

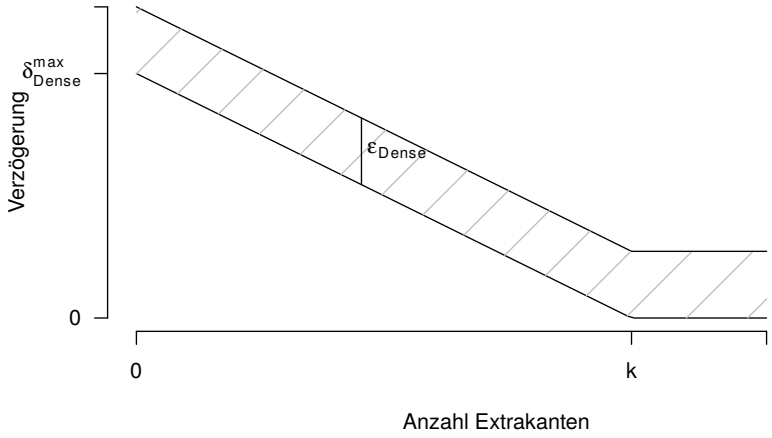


Abbildung 5.2: Skizze der Verzögerungsfunktion zur lokalen Bestimmung, ob ein Knoten Wurzel eines Extrakanten-Spannbaums wird (Dense Mode)

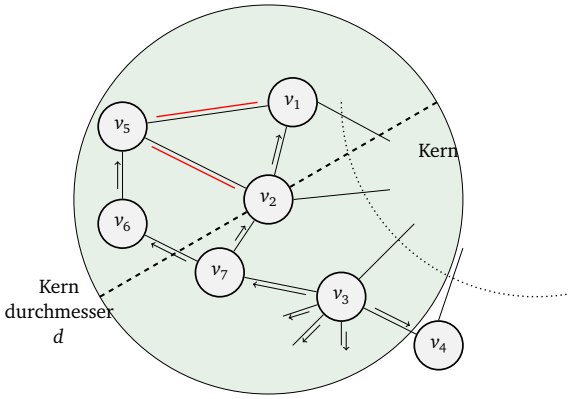


Abbildung 5.3: Auswahl von Wurzeln für Extrakanten-Spannbäume (Sparse Mode)

Sparse Mode Die Erzeugung der Extrakanten-Spannbäume bei Brady und Cowen und auch im Dense Mode von Sprinkles erfolgt so, dass jede Extrakante innerhalb des Randbereichs durch einen Extrakanten-Spannbaum mit einer direkt angrenzenden Wurzel abgedeckt wird. Dies ist für die Einhaltung der Stretch-Schranke nicht notwendig, sondern reduziert nur auf Kosten einer Erhöhung der Adresslänge den durchschnittlichen Stretch. Ähnlich wie beim Hauptspannbaum kann auch im Randbereich ein maximaler additiver Stretch bis zum Kerndurchmesser d zugelassen werden. Während es auch möglich wäre, diesen kleiner als den Kerndurchmesser anzusetzen, ist es das Ziel des Sparse Mode, nur die minimal für die Einhaltung der Stretch-Schranke notwendigen Spannbäume zu erzeugen und so die Adresslänge zu reduzieren. Verfügt ein Knoten über direkt angrenzende Extrakanten und befindet sich innerhalb einer Entfernung von $d/2$ von dem Knoten bereits die Wurzel eines Extrakanten-Spannbaums, so werden diese im Sparse Mode als abgedeckt betrachtet und der Knoten kann davon absehen, einen neuen Spannbaum zu erzeugen.

Abbildung 5.3 stellt die gleiche Situation dar, die zuvor als Beispiel für den Dense Mode verwendet wurde. Lässt man im Rand einen additiven Stretch von bis zu d zu, so genügt die Konstruktion eines einzelnen Spannbaums, um alle Extrakanten abzudecken. Die Wurzel

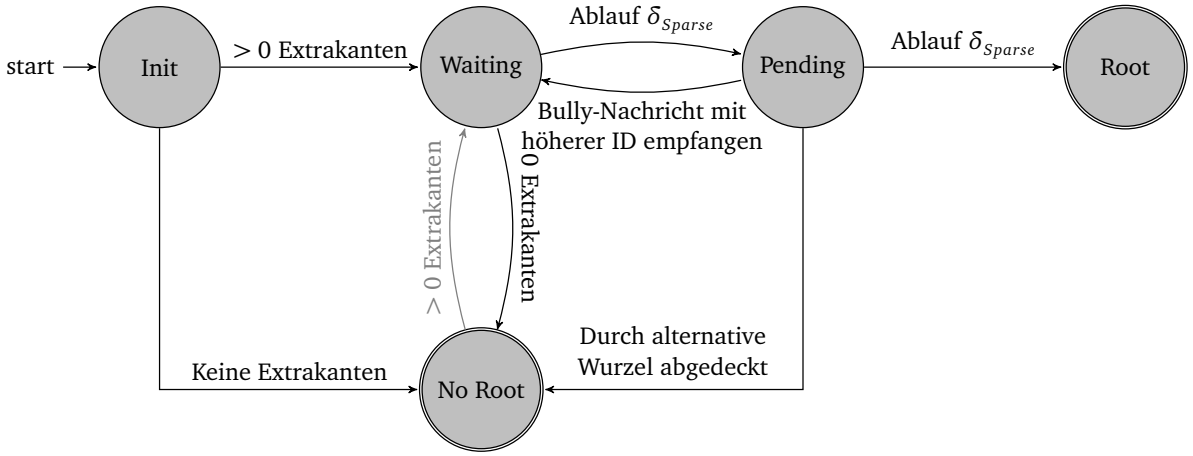


Abbildung 5.4: Zustandsautomat für Sprinkles im Sparse Mode

dieses Spannbaums kann dabei v_1, v_5 oder wie hier der Knoten v_2 sein. Von Knoten v_2 wird ein Extrakanten-Spannbaum aufgebaut (hier nicht eingezeichnet), der den ganzen Randbereich umfasst und damit in der Routing-Tabelle jedes Knotens im Randbereich enthalten ist. Anhand der dadurch bekannten Distanz zur Wurzel v_2 kann jeder der Knoten mit einer Distanz von kleiner gleich $d/2$ (grüner Bereich) zu Knoten v_2 seine Extrakanten als abgedeckt betrachten und muss keine weiteren Extrakanten-Spannbäume erzeugen.

Im Gegensatz zum Dense Mode, der ohne zusätzliche Kommunikation auskommt, wird für den Sparse Mode mit den sogenannten Bully-Nachrichten ein zusätzlicher Nachrichtentyp eingeführt. Bully-Nachrichten enthalten den Identifikator des Knotens, von dem sie erzeugt werden, und die während der Weiterleitung aktualisierte Distanz zu diesem Knoten. Die Weiterleitung der Bully-Nachrichten erfolgt durch einen eingeschränkten Broadcast innerhalb eines Radius von $d/2$ um den Ursprungsknoten und ist zusätzlich auf den jeweiligen Randbereich beschränkt. Knoten, die eine Extrakante abdecken müssen, geben durch den Versand einer Bully-Nachricht zunächst in ihrem Umkreis die Absicht bekannt, Wurzel zu werden. Konkurrieren mehrere Knoten darum, zur Wurzel eines Extrakanten-Spannbaums zu werden, so setzt sich zunächst der Knoten mit dem höheren Identifikator, also der Bully, durch.

Abbildung 5.4 stellt den Zustandsautomaten für Sprinkles im Sparse Mode für den Versand und die Behandlung von Bully-Nachrichten dar. Nach Ablauf des Schutzzeitraums δ_F startet die Ausführung des Zustandsautomaten im Startzustand **Init** und der Knoten bestimmt die Anzahl seiner nicht abgedeckten Extrakanten. Ein Kante ist im Sparse Mode abgedeckt, wenn eines der folgenden Kriterien erfüllt ist:

1. Eine Kante ist durch den jeweiligen Randspannbaum aus T_F bereits abgedeckt und damit keine Extrakante.
2. Für eine Kante (u, v) ist entweder u oder v Wurzel eines Extrakanten-Spannbaums, der diese Kante abdeckt; damit ist die Erzeugung eines zusätzlichen Extrakanten-Spannbaums wegen dieser Kante nicht notwendig.
3. Zusätzlich zu den vorhergehenden Bedingungen, die denen des Dense Mode entspre-

chen, wird eine Kante im Sparse Mode auch als abgedeckt betrachtet, wenn sich die Wurzel eines Extrakanten-Spannbaums in einer Entfernung kleiner gleich $d/2$ befindet.

4. Sonst ist die Kante ist nicht abgedeckt und die Erzeugung eines weiteren Extrakanten-Spannbaums ist erforderlich.

Falls nach diesen Kriterien keine abzudeckende Extrakante vorhanden ist, wechselt der Knoten in den Endzustand No Root. Ist eine Extrakante vorhanden, so setzt der Knoten einen Timer δ_{Sparse} und wechselt in den Zustand Waiting.

Ein Übergang vom Zustand **Waiting** wird unter zwei Bedingungen ausgelöst:

1. Es werden Baum-Nachrichten empfangen, die dazu führen, dass der Knoten über keine Extrakanten mehr verfügt. Damit muss durch den Knoten kein Extrakanten-Spannbaum mehr erzeugt werden. Der Zeitgeber δ_{Sparse} kann abgebrochen werden und der Knoten wechselt in den Zustand **No Root**.
2. Läuft schließlich der Zeitgeber δ_{Sparse} ab, bestehen immer noch unabgedeckte Extrakanten. Der lokale Knoten wechselt daher in den Zustand Pending und gibt durch den Versand einer Bully-Nachricht seine Bereitschaft zur Erzeugung eines Extrakanten-Spannbaumes bekannt. Der Zeitgeber δ_{Sparse} wird erneut gestartet.

Befindet sich ein Knoten im Zustand **Pending**, so findet unter den folgenden Bedingungen ein Zustandswechsel statt:

1. Es wird eine Bully-Nachricht empfangen.
   - Ist der Identifikator des annoncierenden Knotens größer als der des lokalen Knotens, so wird in den Zustand *Waiting* gewechselt und der Zeitgeber δ_{Sparse} neu gestartet.
   - Ist der Identifikator des annoncierenden Knotens hingegen kleiner als der des lokalen Knotens, wird keine Aktion unternommen und der Zeitgeber δ_{Sparse} läuft unberührt weiter.
2. Es werden Baum-Nachrichten empfangen und die bekanntgegebenen Bäume decken alle Kanten des lokalen Knotens ab. In diesem Fall ist es nicht mehr erforderlich, dass der lokale Knoten einen Extrakanten-Spannbaum erzeugt. Der Zeitgeber δ_{Sparse} wird abgebrochen und der Knoten wechselt in den Zustand *No Root*.
3. Läuft der Zeitgeber δ_{Sparse} ab, so existieren noch unabgedeckte Extrakanten und es gab in der Umgebung keinen Bully, der den lokalen Knoten verdrängt hätte. Der lokale Knoten wechselt daher in den Zustand **Root** und versendet für den neu erzeugten Baum die entsprechenden Baum-Nachrichten an seine Nachbarn.

Der grau dargestellte Übergang vom Zustand *No Root* in den Zustand Waiting ist in dem in diesem Kapitel betrachteten Fall statischer Netze nicht erforderlich. Kämen aber zu einer Topologie Kanten hinzu, so wäre dieser Zustandsübergang erforderlich, um die Einhaltung der Stretch-Schranke durch die Erzeugung gegebenenfalls erforderlicher weiterer Extrakanten-Spannbäume zu garantieren. In diesem Fall würde ein Knoten in den Zustand *Waiting* wechseln und dabei wieder den Zeitgeber δ_{Sparse} starten.

Dieses Protokoll ist, wie sich in der späteren Evaluierung herausstellen wird, sehr leichtgewichtig, da es nur geringen Nachrichtenoverhead erzeugt. Für realistische Topologien ist es zudem eine sehr effiziente Möglichkeit zur Reduktion der Anzahl der notwendigen

Tabelle 5.1: Zusammenfassung der Unterschiede bei der Konstruktion der Spannbäume zwischen dem kompakten Routing-Verfahren von Brady und Cowen sowie Sprinkles im Dense Mode und im Sparse Mode

Eigenschaft	BC	Dense Mode	Sparse Mode
Rand-Spannbäume	Erweiterung von T_h	dedizierter Spannbaum	dedizierter Spannbaum
Abdeckung Extrakanten-Spannbäume	gesamter Graph	Ursprungs-Randbereich	Ursprungs-Randbereich
Zulassen von add. Stretch d im Randbereich	nein	nein	ja
Entkopplung Einhaltung Stretch-Schranke / Reduktion Stretch	nein	nein	ja

Extrakanten-Spannbäume, da es diese um einen Faktor von mehr als 40 reduzieren kann. In den Evaluierung wurde der Initialwert von δ_{Sparse} zufällig gleichverteilt aus dem Intervall [0,5] gewählt und zusätzlich mit einem Jitter aus dem Intervall [0,1] belegt.

5.2.2.4 Zusätzliche Spannbäume

Wie bereits in Abschnitt 5.2.1 eingeführt, können in Sprinkles sogenannte zusätzliche Spannbäume zur Reduktion des durchschnittlichen Stretch eingesetzt werden. Dies entspricht in Kombination mit dem Sparse Mode einer klaren Trennung der Belange. Die durch den Sparse Mode erzeugten Spannbäume sind für die Einhaltung der Stretch-Schranke notwendig, während durch weitere zusätzliche Spannbäume unabhängig davon der durchschnittliche Stretch reduziert werden kann. Diese zusätzlichen Spannbäume werden in der späteren Evaluierung wie bei PIE (siehe Abschnitt 2.2.2.2) unter Verwendung von Baum-Ebenen erzeugt.

5.2.2.5 Zusammenfassung

Tabelle 5.1 fasst abschließend die zentralen Unterschiede bei der Konstruktion der Spannbaumstrukturen zwischen dem Routing-Verfahren von Brady und Cowen, Sprinkles im Dense Mode und Sprinkles im Sparse Mode zusammen. Für die verteilte Realisierung in Sprinkles werden immer dedizierte Randspannbäume erzeugt. Zur Reduktion der Adresslänge werden Extrakanten-Spannbäume auf ihren Ursprungsrandbereich beschränkt, statt, wie bei Brady und Cowen, den gesamten Graphen abzudecken. Der Sparse Mode lässt auch in Randbereichen einen additiven Stretch von d zu und erreicht dadurch eine Trennung der Belange zwischen der Einhaltung der Stretch-Schranke und optional zur Reduktion des durchschnittlichen Stretch erzeugbaren zusätzlichen Bäume.

Tabelle 5.2: Zusammenfassung der Unterschiede bzgl. Adressvergabe und Paketweiterleitung zwischen dem kompakten Routing-Verfahren von Brady und Cowen und Sprinkles

Eigenschaft	BC	Sprinkles
Adresslänge beschränkt	ja	nein
Wechseln des Baums möglich	nein	ja
Weiterleitung zu konstanten Kosten	ja	nein

5.2.3 Vergabe der Adressen

Die Vergabe der Adressen innerhalb der Spannbäume geschieht im Routing-Verfahren von Brady und Cowen über zwei zentralisierte Algorithmen, die den kompletten Graphen als Eingabe erwarten. Der eine vergibt ein exaktes Distanz-Label, wie in Abschnitt 2.1.1 vorgestellt. Der andere bestimmt ein Routing-Label, mit dem ein Paket bei konstanten Kosten für eine Weiterleitungsentscheidung entlang eines Spannbaums geroutet werden kann. Die optimale rekursive Unterteilung der Bäume, die für beide Algorithmen jeweils notwendig ist, ist im verteilten Fall nicht ohne massiven Nachrichtenaufwand realisierbar. Daher wird eine alternative Möglichkeit benötigt, diese Funktionalität verteilt umzusetzen. Mit dieser Alternative muss es möglich sein, einerseits die Distanz zwischen zwei Knoten in einem Spannbaum zu bestimmen und andererseits Pakete entlang eines Baumes weiterzuleiten. Wie bereits eingangs erwähnt, wurde dafür die im Rahmen von PIE entwickelte isometrische Einbettung von Spannbäumen (siehe Abschnitt 2.2.2.2) verwendet. Diese Entwurfsentscheidung bringt sowohl Vorteile als auch Nachteile mit sich, die an dieser Stelle diskutiert werden.

Die isometrische Einbettung ist nach erfolgter Spannbaumkonstruktion sehr leicht durchzuführen, da die virtuellen Koordinaten in einem einzelnen Durchlauf von der Wurzel durch den Spannbaum vergeben werden können. Damit entsteht für einen Spannbaum mit n Knoten ein Nachrichtenaufwand von $n-1$ Nachrichten. Der Preis für diese nachrichteneffiziente Vergabe virtueller Koordinaten besteht darin, dass die Adressen nicht ausbalanciert sind und damit keine obere Schranke für die Adresslänge existiert. In der Praxis verhält sich diese jedoch auf Power-Law Topologien, wie auch im Rahmen von PIE bereits gezeigt, ebenfalls polylogarithmisch [44] und liegt in der Größenordnung von $O(\log^3 n)$. Während das BC Routing-Verfahren nach der Entscheidung für einen Spannbaum beim Sender Pakete strikt entlang dieses Spannbaumes weiterleitet, wird die Wahl des optimalen Baumes bei der gierigen Weiterleitung auf jedem Knoten neu getroffen. Dies ermöglicht einerseits die Nutzung von Abkürzungen, die erst unterwegs durch einen besser geeigneten Spannbaum sichtbar werden. Andererseits erhöht dies die Kosten pro Weiterleitungsentscheidung gegenüber dem BC Routing-Verfahren. Während das zweite durch das BC Routing-Verfahren vergebene Label die Weiterleitung entlang des einmal gewählten Spannbaums in konstanter Zeit ermöglicht, trifft PIE auf jedem Weiterleitungsknoten erneut die Entscheidung, welcher Spannbaum genutzt wird. Dazu wird auf jedem Knoten, über den das Paket weitergeleitet wird, für alle Nachbarn des Knotens die Distanz zum Ziel erneut bestimmt und gierig die beste gewählt.

5.2.4 Einhaltung der Stretch-Schranke

Um die Einhaltung der Stretch-Schranke durch Sprinkles zu zeigen, wird hier der Sparse Mode ohne zusätzliche Spannbäume untersucht. Dieser Fall stellt durch die Reduktion der konstruierten Extrakanten-Spannbäume gegenüber dem BC Routing-Verfahren den kritischen Fall bezüglich der Einhaltung der Stretch-Schranke dar. Anhand der vorhergegangenen Tabelle 5.1 und Tabelle 5.2 wird dazu nun gezeigt, welche Teile des Beweises von Brady und Cowen unverändert übernommen werden können und wo Anpassungen notwendig sind.

1. $u, v \in$ Kern: Für den Fall, dass beide Knoten u und v im Kern liegen, ist nur der Hauptspannbaum T_h für die Einhaltung der Stretch-Schranke relevant, der in beiden Modi von Sprinkles exakt wie bei Brady und Cowen konstruiert wird. Im Gegensatz zur Weiterleitung von Paketen bei Brady und Cowen werden bei Sprinkles Pakete auch über Nichtspannbaumkanten, die Abkürzungen darstellen, weitergeleitet. Dies kann aber nur zu kürzeren Pfaden führen und ist damit ebenfalls keine Einschränkung. Ebenso kann das Vorhandensein von zusätzlichen Baum-Ebenen nur zu besseren Pfaden führen, da diese nur verwendet würden, wenn gegenüber dem sonst verwendeten T_h eine Verbesserung erreicht wird. Daher gilt dieser Teil des Beweises unverändert:
   $d_{T_h}(u,v) \leq d_{T_h}(u,h) + d_{T_h}(h,v) = d(u,h) + d(h,v) \leq d/2 + d/2 = d$. Da $d(u,v) \geq 1$, ist $d_{T_h}(u,v) \leq d(u,v) + (d-1) < d(u,v) + d$.
2. $u \in$ Kern, $v \in$ Randbereich: Befindet sich ein Knoten o. B. d. A. Knoten u im Kern und Knoten v in einem Randbereich, so gibt es im Sparse Mode ohne zusätzliche Bäume nur den Baum T_h, über den ein Paket weitergeleitet werden kann. Dabei ist es durch die Möglichkeit der Nutzung von Nichtspannbaumkanten oder eventuell vorhandene zusätzliche Spannbäume möglich, dass der gewählte Pfad kürzer wird als eine Weiterleitung entlang von T_h; schlechter werden kann er jedoch nicht. Damit gilt auch dieser Teil des Beweises unverändert:
   $d_T(u,v) \leq d_{T_h}(u,h) + d_{T_h}(h,v) = d(u,h) + d(h,v) \leq d/2 + d(h,v) \leq d/2 + d(u,h) + d(u,v) \leq d/2 + d/2 + d(u,v) = d(u,v) + d$
3. $u, v \in$ Randbereich: Ist ein kürzester Pfad P zwischen u und v Teil des Randspannbaums, so erfolgt die Weiterleitung entlang des kürzesten Pfades. Ist hingegen $d_{T_F}(u,v) > d(u,v) = |P|$, so muss es eine Kante (u', v') geben, die im kürzesten Pfad P enthalten, aber nicht Bestandteil von T_F ist.
   a) $u', v' \in$ Randbereich: Ist eine Kante (u', v') in dem Randbereich nicht Bestandteil von T_F, so muss nach Ausführung des Sparse Mode Protokolls für jeden der beiden Knoten ein (nicht notwendigerweise unterschiedlicher) Extrabaum mit einer Wurzel $r_{u'}$ bzw. $r_{v'}$ konstruiert worden sein, wobei $d(u', r_{u'}) \leq d/2$ und $d(v', r_{v'}) \leq d/2$. Betrachtet wird nun o. B. d. A. der Baum T mit der Wurzel $r_{u'}$ und dafür gilt:
      $d(u,v) \leq d_T(u, r_{u'}) + d_T(r_{u'}, v) = d(u, r_{u'}) + d(r_{u'}, v)$. Unter Verwendung der Dreiecksungleichung folgt, dass $d(u, r_{u'}) \leq d(u,u') + d(u', r_{u'})$ und $d(r_{u'}, v) \leq d(r_{u'}, u') + d(u', v)$.
      Damit folgt, dass $d(u, r_{u'}) + d(r_{u'}, v) \leq d(u,u') + d(u', r_{u'}) + d(r_{u'}, u') + d(u', v) \leq d(u,u') + 2 \cdot d/2 + d(u', v) = d(u,v) + d$
   b) u' oder $v' \in$ Kern: Ist der kürzeste Pfad nicht vollständig im Randbereich ent-

halten, so können Pakete über den Hauptspannbaum T_h weitergeleitet werden. Da mindestens ein Knoten u' des kürzesten Pfades im Kern liegt, gilt $d(u,u') + d(u',v) = d(u,v)$. Damit folgt $d(u,v) = d(u,u') + d(u',v) \leq d_{T_h}(u,h) + d_{T_h}(h,v) \leq d_{T_h}(u,x) + d/2 + d/2 + d_{T_h}(x,v) \leq d(u,v) + d$.

Damit gilt also selbst für Sprinkles im Sparse Mode, das nach dem Prinzip Trennung der Belange nur die minimal zur Einhaltung der Stretch-Schranke notwendigen Routing-Strukturen aufbaut, die additive Stretch-Schranke d.

5.3 Evaluierung

Um Sprinkles im Detail zu untersuchen, wurde dieses im zuvor vorgestellten Simulator RoutingSim (siehe Kapitel 4) implementiert. In den hier durchgeführten Experimenten werden die folgenden Arten von Topologien untersucht:

- Um das makroskopische Verhalten von Sprinkles zu untersuchen, wird dieses zunächst auf synthetischen Power-Law Topologien (siehe Abschnitt 2.1.4.1) evaluiert. Der Fokus dieser Untersuchung liegt dabei einerseits darauf, die Einhaltung der Stretch-Schranke auf dem sehr breit gewählten Parameter-Raum empirisch zu bestätigen. Zudem werden praktisch verwendbare Werte für den Kerndurchmesser identifiziert und die Effektivität des Sparse Mode zur Reduktion der Adresslänge untersucht.
- Anschließend wird Sprinkles auf der CAIDA Router-Topologie evaluiert, um dessen Verhalten auf einer realistischen Topologie zu untersuchen und eine optimale Parametrisierung für den Kerndurchmesser zu bestimmen.

Als praktikable *maximale* Adresslänge werden hier 100 Koordinaten definiert, wobei eine Koordinate bei PIE in etwa einem Byte entspricht. Mit einer im weiteren Verlauf der Arbeit vorgeschlagenen, effizienteren Kodierung (siehe Abschnitt 6.5.2.1) ist es möglich, eine Koordinate unter Verzicht auf Kantengewichte in nur 2 Bits darzustellen. Damit entspricht die maximale Adresslänge, die hier als noch praktikabel definiert wird, 100 Bytes (mit Kantengewichten) oder mit der hier entwickelten effizienteren Kodierung 100 Koordinaten x 2 Bits = 200 Bits = 25 Bytes (ohne Kantengewichte). Dabei ist zu beachten, dass diese Beschränkung sich nicht auf die durchschnittliche Adresslänge bezieht. Diese kann deutlich niedriger ausfallen.

5.3.1 Synthetische Power-Law Topologien

Als synthetische Power-Law Topologien wurden die durch Brady und Cowen teilweise bereitgestellten Topologien (`http://digg.cs.tufts.edu/samples/`) verwendet, die diese auch zur Evaluierung ihres kompakten Routing-Schemas verwendet haben [2]. Nicht zur Verfügung gestellte Topologien wurden für die im Rahmen dieser Arbeit durchgeführten Simulationen anhand des ebenfalls bereitgestellten Topologie-Generators[1] neu erzeugt. Diese Topologien werden mit einer festen Anzahl von N Knoten (Parameter der Topologiegenerierung) erzeugt und so verbunden, dass die dem Power-Law Exponenten β ent-

[1] `http://digg.cs.tufts.edu/source/generators.zip`

Tabelle 5.3: Untersuchter Parameterraum auf den Digg Topologien

Parameter	Werte
Topologie-Erzeugung N	$\{2500, 5000, 10000, 20000, 40000\}$
Power-Law Exponent β	$\{1.2, 1.3, ..., 2.9, 3.0\}$
Anzahl Topologien pro Konfiguration	5
Kerndurchmesser d	{4,6,8,10,12,14}
Zusätzliche Spannbäume	0
Aufbau der Extrakanten-Bäume	{Dense Mode, Sparse Mode}
Sampling Datenpakete pro Konfiguration	10000

sprechende Knotengradverteilung resultiert. Da mit wachsendem Power-Law Exponenten die größte Zusammenhangskomponente nicht notwendigerweise alle Knoten umfasst, wird jeweils die größte Zusammenhangskomponente der erzeugten Topologien untersucht. Um topologie-spezifische Effekte zu reduzieren, wurden für jede N-β-Konfiguration fünf Topologien erzeugt. Für Durchschnittswerte wurde der Mittelwert der jeweils erhaltenen Ergebnisse berechnet. Für Maximumswerte, wie beispielsweise die maximale Adresslänge oder den maximalen Stretch, wurde das Maximum über alle Topologien bestimmt.

Sprinkles wurde auf diesen Topologien, wie in Tabelle 5.3 dargestellt, mit Kerndurchmessern d im Bereich von 4 bis 14 untersucht. Zudem wurden die Simulationen sowohl im Dense Mode als auch im Sparse Mode durchgeführt. Während der durchschnittliche Stretch durch das Hinzufügen zusätzlicher Spannbäume reduziert werden kann, wurden an dieser Stelle bewusst nur Konfigurationen ohne zusätzliche Spannbäume untersucht. Diese Entscheidung wurde getroffen, da hier die Einhaltung der Stretch-Schranke empirisch überprüft werden soll und diese Überprüfung durch zusätzliche Spannbäume, die ihrerseits den Stretch reduzieren, verfälscht werden könnte.

Da die vollständige Evaluierung aller n^2 Knotenpaare durch die notwendigen Berechnungen kürzester Pfade zu aufwändig gewesen wäre, wurde ein Sampling verwendet und gleichverteilt zufällige Knotenpaare ausgewählt, zwischen denen Datenpakete versendet wurden.

Der Ablauf der Simulationen war dabei folgendermaßen: Zum Zeitpunkt $t = 0$ wurde der Hauptspannbaum vom Knoten h mit dem höchsten Grad aus erzeugt. Anschließend wurden zum Zeitpunkt $t = 5s$ zunächst die Rand-Spannbäume und zum Zeitpunkt $t = 15s$ die Extrakanten-Spannbäume erzeugt und danach in allen Bäumen virtuelle Koordinaten vergeben. Nach erfolgtem Aufbau der Routing-Tabellen wurden 10000 Nachrichten zwischen gleichverteilt zufällig ausgewählten Knotenpaaren versendet, um auch bei einer Topologiegröße von 40000 Knoten genaue Ergebnisse bestimmen zu können. Für kleinere Topologien wurde ebenfalls mit 10000 Nachrichten pro Konfiguration gearbeitet.

5.3.1.1 Einhaltung der additiven Stretch-Schranke

Um die durch Sprinkles gegebene Garantie für die Einhaltung der Stretch-Schranke empirisch zu überprüfen, wird zunächst der maximale Stretch der Simulationen untersucht. Abbildung 5.5 stellt dazu für jeden untersuchten Power-Law Exponenten β in Abhängigkeit vom gewählten Kerndurchmesser d den maximalen im Sparse Mode beobachteten additiven Stretch für alle N dar. Der Sparse Mode wurde hier gewählt, da dieser im Vergleich zum Dense Mode weniger Spannbäume für das Routing aufbaut und damit einen höheren Stretch aufweisen kann. In der linken oberen Ecke jeder Abbildung ist für die Topologien mit $N = 40000$, die später noch im Detail untersucht werden, jeweils deren Durchmesser (diam) und der durchschnittliche Knotengrad (deg) angegeben.

Zunächst lässt sich anhand der ebenfalls eingezeichneten Winkelhalbierenden leicht überprüfen, dass der maximale additive Stretch (y-Achse) für alle Konfigurationen kleiner oder gleich dem als Stretch-Schranke agierenden Kerndurchmesser (x-Achse) ist. Dies war auch im hier nicht dargestellten Dense Mode durchgängig der Fall. Die Stretch-Schranke wird also ergänzend zu dem zuvor geführten Beweis auch in den Simulationen stets eingehalten.

Anhand der in jeder Teilabbildung beispielhaft für $N = 40000$ vermerkten Graphendurchmesser ist ersichtlich, dass Sprinkles die verteilte Realisierung nicht-trivialer Stretch-Schranken ermöglicht. Während ein Routing-Protokoll wie beispielsweise PIE durch seinen Aufbau der Spannbäume eine triviale, vom Durchmesser der Topologie abhängige Stretch-Schranke besitzt, können mit Sprinkles deutlich niedrigere Stretch-Schranken realisiert werden. Für beispielsweise $\beta = 2.7$ hat die mit $N = 40000$ erzeugte Topologie einen Durchmesser von 33 Übertragungsabschnitten und Sprinkles ist in der Lage, den maximalen additiven Stretch bei einem Kerndurchmesser von 4 auf 4 Übertragungsabschnitte zu beschränken.

Allgemein lässt sich anhand der in Abbildung 5.5 eingetragenen Durchmesser und durchschnittlichen Knotengrade für $N = 40000$ die Eigenschaft von Power-Law Topologien beobachten, dass mit steigendem β der durchschnittliche Knotengrad sinkt (Spearman Rangkorrelationskoeffizient $\rho < -0.99$, $p < 0.00001$) und der Durchmesser der Topologien steigt (Spearman Rangkorrelationkoeffizient $\rho > 0.98$, $p < 0.00001$). Für kleine β sind die Topologien dicht vermascht und alle kürzesten Wegen sind kurz, während für große β das Gegenteil der Fall ist.

Für $\beta \leq 1.5$ ist der maximale beobachtete additive Stretch stets kleiner gleich 4 und für $\beta \leq 1.9$ stets kleiner 6 Übertragungsabschnitte. Dies ist auf die dichte Vermaschung und den verhältnismäßig kleinen Durchmesser der jeweiligen Topologien zurückzuführen. Für $2 \leq \beta \leq 2.3$ steigt der maximale Stretch mit steigendem Kerndurchmesser erkennbar an. Bis zu einem Kerndurchmesser von $d = 8$ ($\beta = 2.3$, $N = 40000$) wird der maximale Stretch auch tatsächlich beobachtet. Für $2.4 \leq \beta \leq 2.7$, also weniger dicht vermaschte Topologien mit größerem Durchmesser, wird der maximale Stretch deutlich häufiger angenommen als für kleinere β. Für Topologien mit $\beta \geq 2.8$ sinkt der maximal beobachtete Stretch dagegen wieder. Die Topologien haben im Falle dieser hohen β selbst eine baumartige Struktur (durchschnittlicher Knotengrad 2.0). Dadurch entsteht bei Routing entlang von Bäumen weniger Stretch als wenn es bei höherem Knotengrad mehr Abkürzungen zwischen den Ästen eines Baumes gibt.

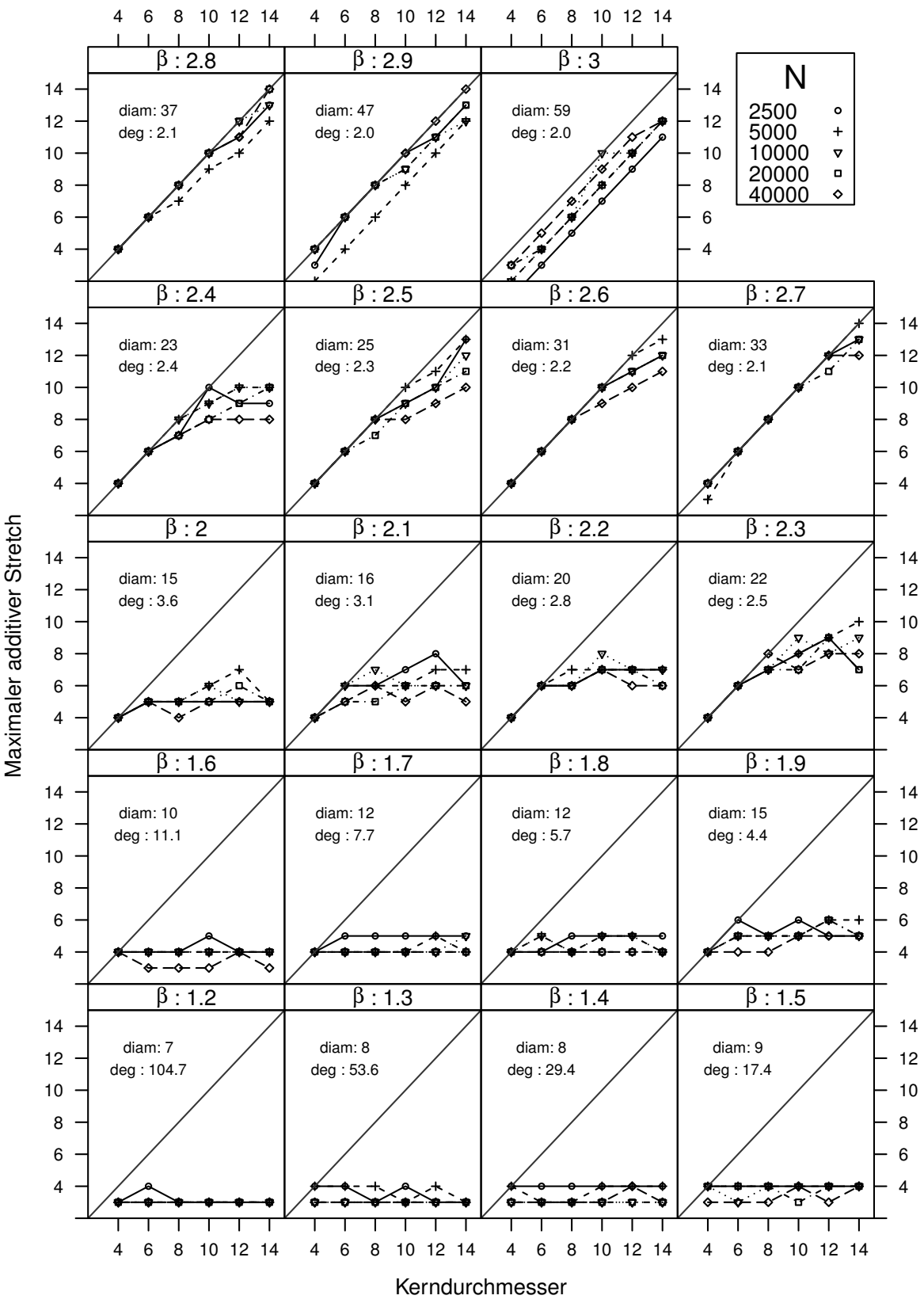


Abbildung 5.5: Maximaler additiver Stretch für Kerndurchmesser von 4 bis 14 für die untersuchten Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ und $N = 2500, 5000, 10000, 20000, 40000$ im Sparse Mode.

5.3.1.2 Kosten für die Einhaltung der Stretch-Schranke

Nachdem die Einhaltung der Stretch-Schranke bewiesen und auch empirisch gezeigt wurde, stellt sich an dieser Stelle die Frage nach den Kosten für die Einhaltung der Stretch-Schranke. Eine Metrik für diese Kosten ist die Größe der Routing-Tabellen, die sich hier durch die gierige Weiterleitung von Paketen aus der Anzahl der Nachbarn eines Knotens und deren Adresslänge bestimmt. Unter der Annahme, dass ein Knoten mit vielen Nachbarn auch über entsprechend mehr Ressourcen (beispielsweise Speicher und Rechenleistung) verfügt als ein Knoten mit wenigen Nachbarn, stellt die Adresslänge den dominierenden Faktor dar. Adressen müssen für die Speicherung in den Routing-Tabellen möglichst klein sein. Viel wichtiger ist aber, dass die Adresse eines Knotens auch in Kontroll- und Datenpaketen mitgeführt werden muss. Als maximale praktikable Adresslänge wurden eingangs 100 Koordinaten festgelegt, was im Falle von ungewichteten Kanten 25 Bytes und damit etwa dem 1.5-fachen einer IPv6-Adresse entspricht. Anhand dieser maximalen Adresslänge werden im folgenden praktikable Parametrisierungen des Kerndurchmessers für die untersuchten Topologien identifiziert.

Abbildung 5.6 gibt einen Überblick über die maximale beobachtete Adresslänge über den untersuchten Parameterraum. Jede Teilabbildung stellt für die mit dem entsprechenden β generierten Topologien die maximale beobachtete Adresslänge auf einer logarithmischen Skala in Abhängigkeit vom Kerndurchmesser d für den Sparse Mode dar. Zur besseren Orientierung ist ebenfalls eine horizontale Linie bei der zuvor definierten Schranke von 100 Koordinaten eingezeichnet. In der Abbildung ist zu erkennen, dass die maximale Adresslänge für Topologien mit $\beta \leq 1.6$ unabhängig vom Kerndurchmesser d durchgängig geringer als 100 Koordinaten ist. Für Topologien im Bereich $1.7 \leq \beta \leq 2.6$ lässt sich dagegen in Abhängigkeit vom Kerndurchmesser ein sehr scharfer Übergang von völlig inpraktikablen maximalen Adresslängen im Bereich mehrerer zehntausend Koordinaten in den praktikablen Bereich mit kleiner als 100 Koordinaten beobachten. Im Bereich von $\beta \geq 2.7$ sinkt die maximale Adresslänge mit wachsendem Kerndurchmesser weiter stark (logarithmische Skala). Allerdings ist die in diesen Topologien beobachtete maximale Adresslänge fast durchgehend größer als die maximale für praktikabel definierte Adresslänge von 100 Koordinaten. Da mit wachsendem β die Topologien stark ausgedünnt werden, kann es große Unterschiede zwischen den resultierenden Spannbaumkonstruktionen auf den verschiedenen untersuchten Topologien geben, die auch die maximale Adresslänge für große β entsprechend beeinflussen.

Die Kosten einer sehr niedrigen additiven Beschränkung des Stretch äußern sich also, wie auch zu erwarten war, direkt in der Adresslänge. Dies resultiert aus dem inhärenten Trade-Off zwischen einer Beschränkung des Stretch und der resultierenden Größe der Routing-Tabellen, die bei Sprinkles aus einer pro Nachbar gespeicherten Adresse bestehen. Für eine detaillierte Untersuchung dieses Effekts sowie den Vergleich zwischen dem Dense Mode und dem Sparse Mode werden im weiteren Verlauf die mit $N = 40000$ erzeugten Topologien herangezogen.

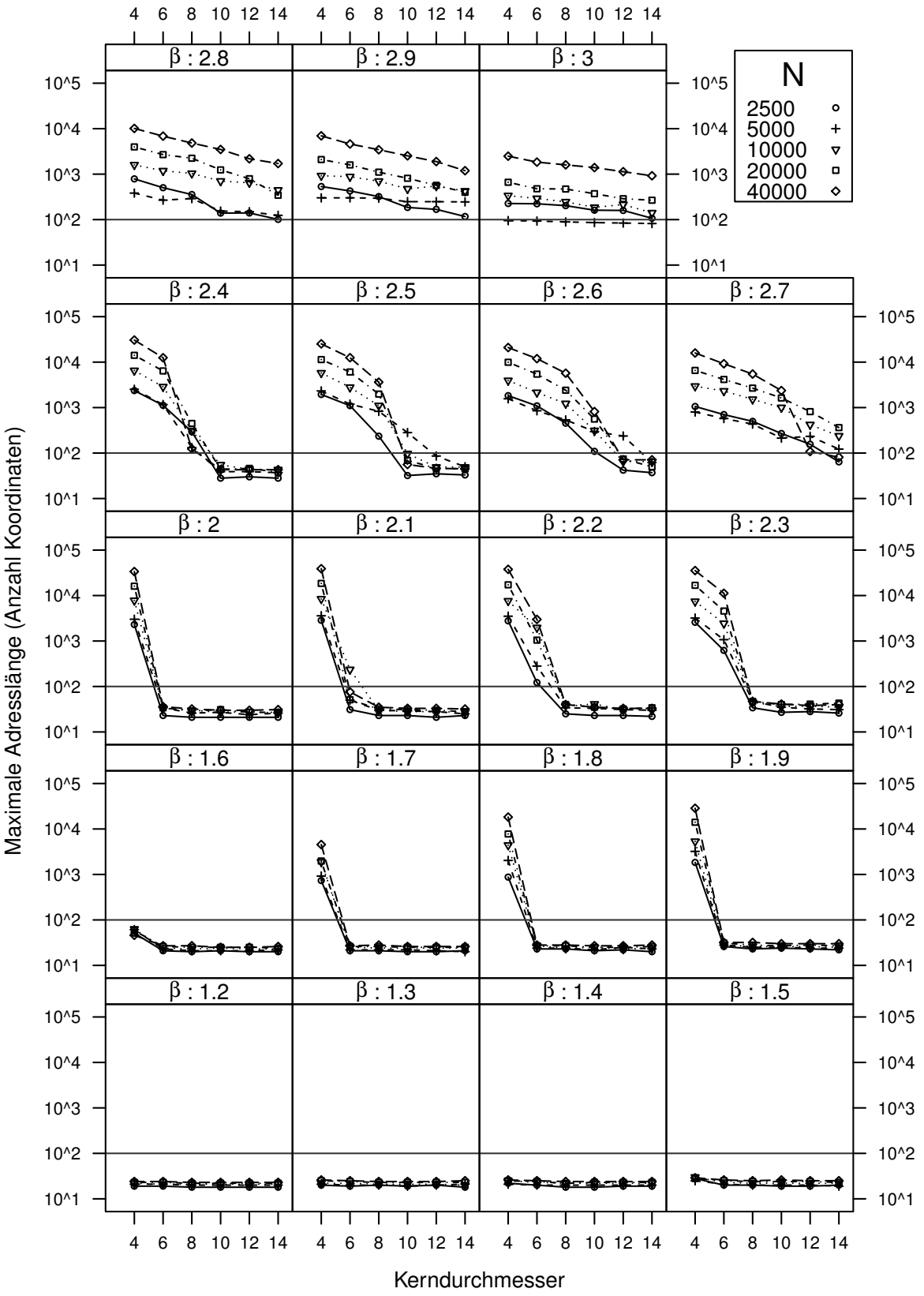


Abbildung 5.6: Maximale Adresslänge (Anzahl Koordinaten) für Kerndurchmesser von 4 bis 14 für die untersuchten Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ und $N = 2500, 5000, 10000, 20000, 40000$ im Sparse Mode.

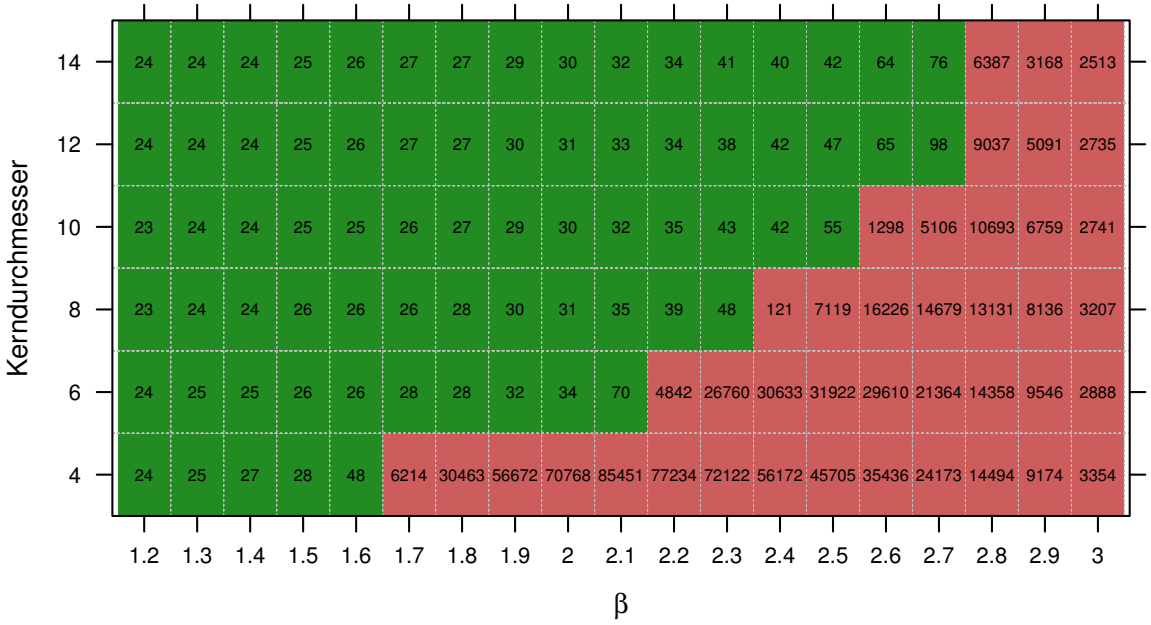


Abbildung 5.7: Einteilung der maximalen Adresslänge in praktikable (grün, ≤ 100) und inpraktikable (rot, > 100) Bereiche für Topologien mit $N = 40000$ und Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ für Kerndurchmesser von 4 bis 14 im Dense Mode.

5.3.1.3 Detailuntersuchungen für $N = 40000$

Für die Topologien mit $N = 40000$ wird zunächst, anschließend an den vorherigen Abschnitt, die Adresslänge betrachtet und dabei der Unterschied zwischen Dense Mode und Sparse Mode untersucht.

Abbildung 5.7 stellt die maximale beobachtete Adresslänge in Abhängigkeit von β und dem vorgegebenen Kerndurchmesser für den Dense Mode dar. Abbildung 5.8 zeigt die entsprechenden Ergebnisse für den Sparse Mode. Zunächst ist zu beobachten, dass sowohl der Dense Mode als auch der Sparse Mode in vielen Konfigurationen praktisch einsetzbar sind. Der Übergang zwischen dem praktikablen (grün) und nicht praktikablen (rot) Bereich ist dabei, wie auch zuvor in dem Gesamtüberblick zu beobachten war, mit zwei Ausnahmen ($\beta = 2.4$, $d = 8$ und $\beta = 2.7$, $d = 12$) sehr scharf ausgeprägt. Für die Konfiguration $\beta = 2.7$ mit $d = 12$ ist sogar der Dense Mode praktikabel, während der Sparse Mode die Grenze von 100 Koordinaten mit 108 Koordinaten knapp übersteigt. Für die inpraktikablen Bereiche reduziert der Sparse Mode die Adresslänge deutlich, oft um einen Faktor von mehr als 2; allerdings reicht dies nicht aus, um die dort vorherschenden extremen Adresslängen auf unter 100 Koordinaten zu reduzieren. Der Grund für das im Vergleich schlechte Abschneiden des Dense Mode lässt sich anhand von Abbildung 5.9 erkennen, die den Anteil von Knoten innerhalb des Kerns in Relation zur Topologiegröße darstellt. Die dicke schwarze Linie trennt die Abbildung dabei in den praktikablen Teil links der Linie und den inpraktikablen rechts der

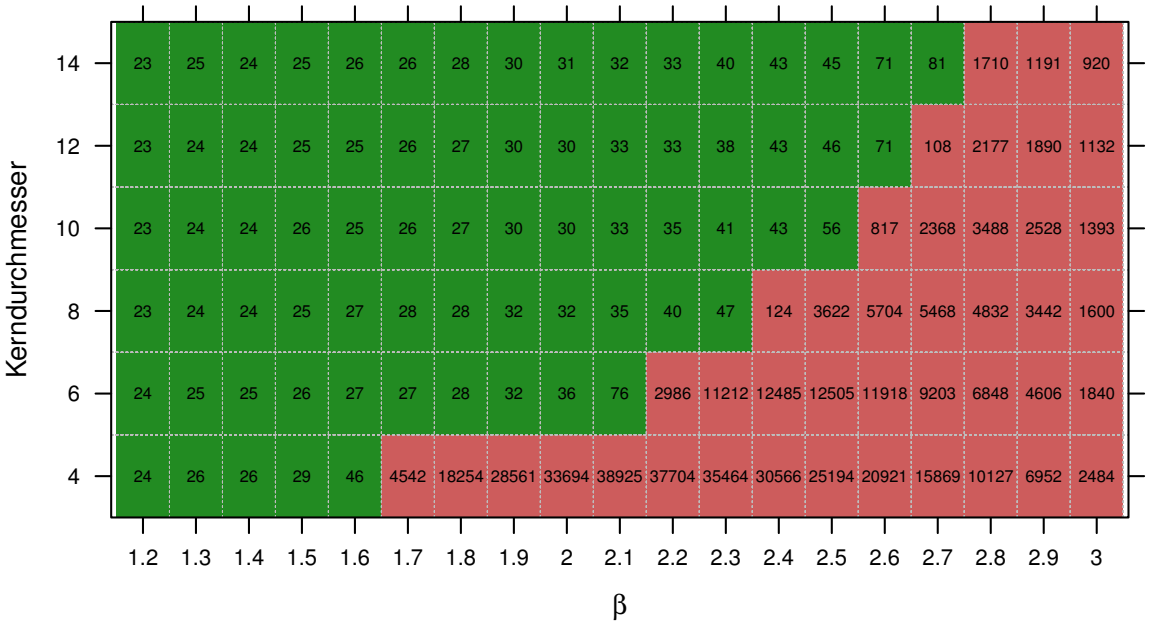


Abbildung 5.8: Einteilung der maximalen Adresslänge in praktikable (grün, ≤ 100) und inpraktikable (rot, > 100) Bereiche für Topologien mit $N = 40000$ und Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ für Kerndurchmesser von 4 bis 14 im Sparse Mode.

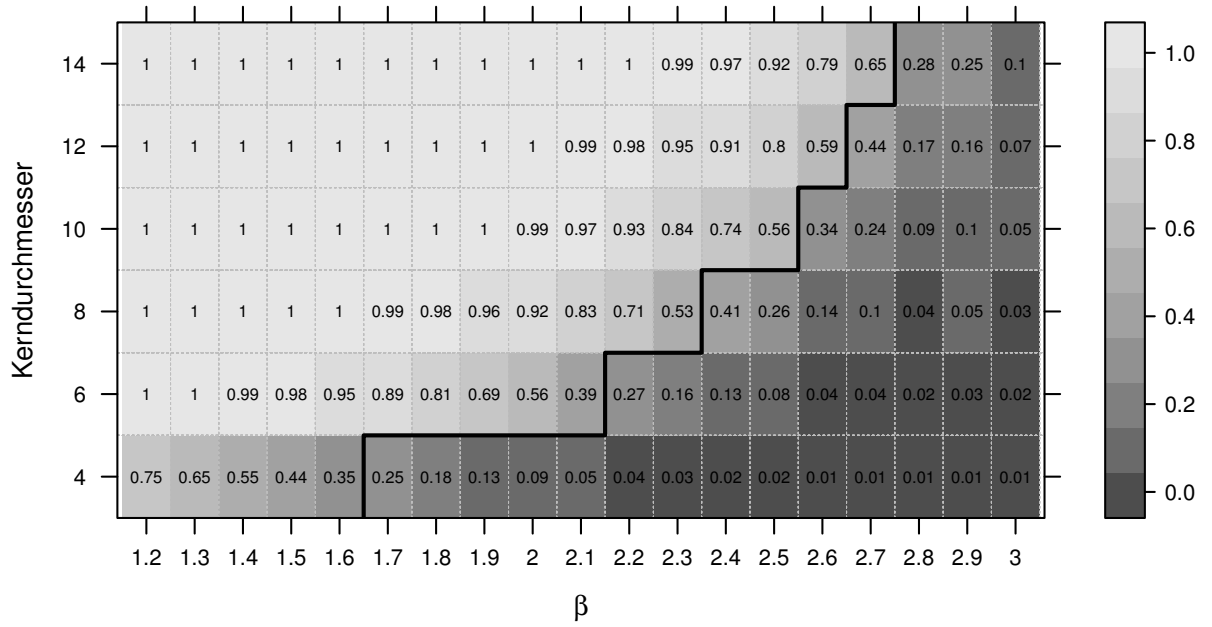


Abbildung 5.9: Anteil der im Kern befindlichen Knoten für Topologien mit $N = 40000$ und Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ für Kerndurchmesser von 4 bis 14.

Linie. Für die inpraktikablen Konfigurationen liegen dabei in fast allen Fällen 40% oder sogar deutlich weniger Knoten im Kern. Durch die große Anzahl von Knoten im Randbereich ist die Entstehung großer individueller Randbereiche zu erwarten, die ihrerseits in der Erzeugung von Extrakanten-Spannbäumen und damit zusätzlichen virtuellen Koordinaten resultieren. Ein Grund für das besonders gute Abschneiden bezüglich der Adresslänge von Topologien mit $\beta \leq 1.5$ wird an dieser Abbildung auch deutlich, da für Kerndurchmesser $d \geq 8$ alle Knoten im Kern liegen und damit nie Rand-Spannbäume oder Extrakante-Spannbäume erzeugt werden. Der interessante Bereich liegt also zwischen diesen beiden Extremen.

Abbildung 5.10 stellt die über alle Topologien beobachtete maximale Größe eines einzelnen Randbereichs dar. Mit Ausnahme der bereits vorher erwähnten Konfigurationen ($\beta = 2.4$, $d = 8$ und $\beta = 2.7$, $d = 12$), die sehr nahe an dem definierten praktikablen Bereich liegen, sind alle anderen Randbereiche deutlich größer als 3000 Knoten und derart strukturiert, dass es weder im Dense Mode noch im Sparse Mode möglich ist, in die Nähe der gewünschten Adresslänge zu kommen.

Abbildung 5.11 stellt die Anzahl an Randbereichen dar, in die ein gewisser Kerndurchmesser die Topologien unterteilt. Betrachtet man die Abbildung zeilenweise, so kann man feststellen, dass die Anzahl an Randbereichen innerhalb des praktikablen Bereichs links der Grenze anwächst, was zu eher kleineren Randbereichen führt. Jenseits dieser Grenze fällt die Anzahl an Randbereichen ab. Die Randbereiche werden also größer und gleichzeitig steigt deren Durchmesser bei sinkendem durchschnittlichen Grad. All diese Eigenschaften führen dazu, dass dieser Bereich mit Sprinkles nicht sinnvoll nutzbar ist.

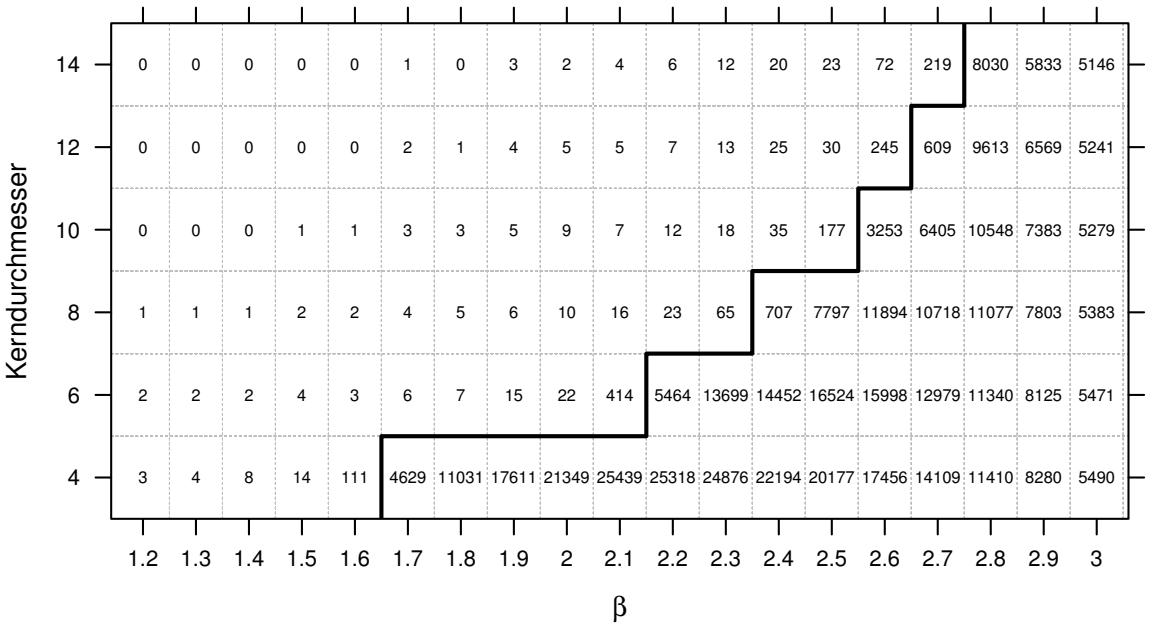


Abbildung 5.10: Anzahl der Knoten im größten Randbereich für Topologien mit $N = 40000$ und Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ für Kerndurchmesser von 4 bis 14.

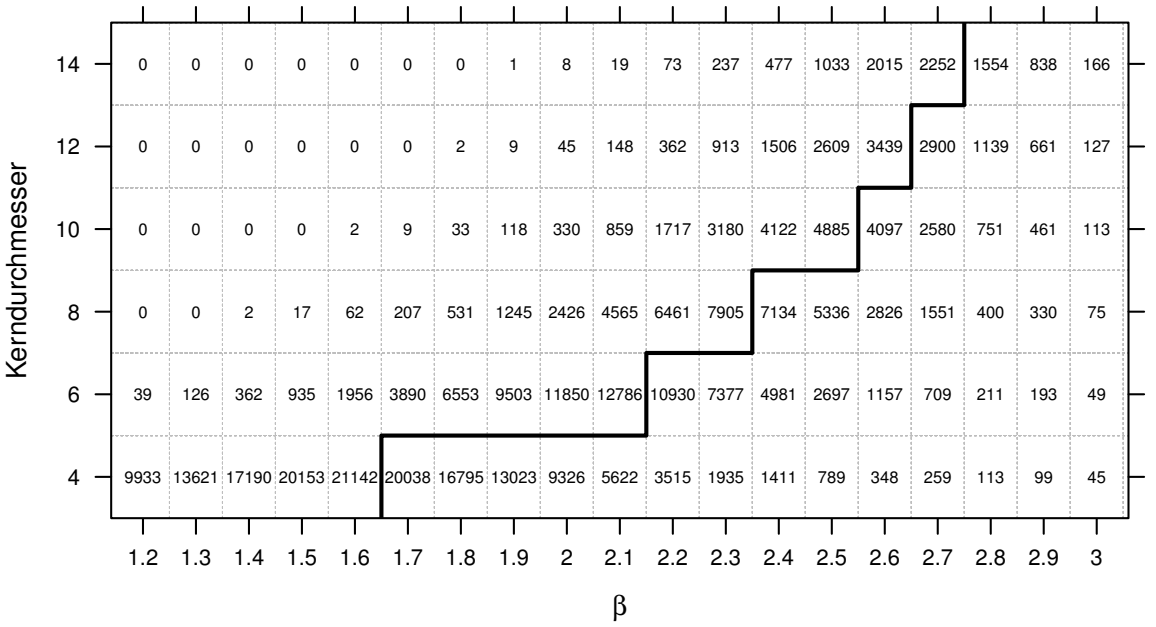


Abbildung 5.11: Anzahl der Randbereiche für Topologien mit $N = 40000$ und Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ für Kerndurchmesser von 4 bis 14.

5.3.1.4 Durchschnittlicher Stretch

Abbildung 5.12 stellt für jeden untersuchten Power-Law Exponenten β in Abhängigkeit vom gewählten Kerndurchmesser den durchschnittlichen im Sparse Mode beobachteten multiplikativen Stretch für alle N dar. Der multiplikative Stretch wurde hier zur besseren Vergleichbarkeit mit anderen Protokollen verwendet, da der durchschnittliche Stretch in den Evaluierungen dort üblicherweise multiplikativ angegeben wird. Der durchschnittliche multiplikative Stretch bewegt sich dabei für alle untersuchten Konfigurationen unterhalb von 1.3 und liegt stellenweise deutlich darunter. Betrachtet man den durchschnittlichen Stretch in Kombination mit den in Abbildung 5.6 dargestellten maximalen Adresslängen, so ist zu beobachten, dass der durchschnittliche Stretch dann niedrig ist, wenn die Adresslängen durch viele erzeugte Extrakanten-Spannbäume hoch sind. Dadurch entsteht in den Randbereichen zu Lasten einer unbrauchbaren Adresslänge weniger Stretch. Werden dagegen, wie beispielsweise im Bereich von $1.2 \leq \beta \leq 1.5$, durch die mit maximal 14 Knoten (siehe Abbildung 5.10) sehr kleinen Randbereiche nur sehr wenige Randspannbäume erzeugt, so bewegt sich der durchschnittliche multiplikative Stretch im Bereich von 1.2-1.3. Ein durchschnittlicher multiplikativer Stretch größer als 1.3 wurde in keiner der Untersuchungen beobachtet.

5.3.2 CAIDA Router-Level Topologie

In diesem Abschnitt wird mit der Caida Topologie, einer Topologie des Internets auf Router-Ebene, die größte hier betrachtete Topologie untersucht. Dabei zeigt sich, dass der Sparse Mode im Gegensatz zu den wenig vielversprechenden Untersuchungen auf synthetischen Topologien hier eine deutliche Verbesserung erreicht und der durchschnittliche Stretch durch das Hinzufügen zusätzlicher Spannbäume gut reduziert werden kann. Wie in Abschnitt 5.2.2.4 eingeführt, werden für die Konstruktion der zusätzlichen Spannbäume die PIE Baum-Ebenen verwendet und in der Evaluierung 0, 1 und 2 Baum-Ebenen untersucht.

Abbildung 5.13 stellt zunächst den maximalen additiven Stretch in Abhängigkeit vom gewählten Kerndurchmesser dar. Untersucht wurden jeweils der Dense und Sparse Mode bei 0, 1 und 2 Baum-Ebenen. Dabei ist zu beobachten, dass der maximale additive Stretch an keiner Stelle sein mögliches Maximum (den Kerndurchmesser) annimmt, was gegebenenfalls auf die nicht vollständige Untersuchung aller möglichen Knotenpaare zurückzuführen ist. Ebenfalls ist eine geringe Reduktion des maximalen additven Stretch durch das Hinzufügen von zusätzlichen Baum-Ebenen sichtbar. Dies schlägt sich noch viel deutlicher und konsistenter im durchschnittlichen Stretch nieder, der in Abbildung 5.14 wieder für 0, 1 und 2 Baum-Ebenen und den Dense Mode wie auch den Sparse Mode dargestellt ist.

Der durchschnittliche multiplikative Stretch sinkt hier durch das Hinzufügen zusätzlicher Baum-Ebenen deutlich ab. Zugleich weisen der Dense Mode und der Sparse Mode beim durchschnittlichen Stretch eine sehr ähnliche Leistungsfähigkeit auf, was bei der folgenden Betrachtung der Adresslängen verwunderlich ist, da der Dense Mode deutlich längere Adressen und damit größere Routing-Tabellen erzeugt.

Abbildung 5.15 stellt die maximale Adresslänge auf einer logarithmischen Skala in Abhängigkeit vom Kerndurchmesser dar. Es werden wie zuvor 0,1 und 2 Baum-Ebenen für den Dense

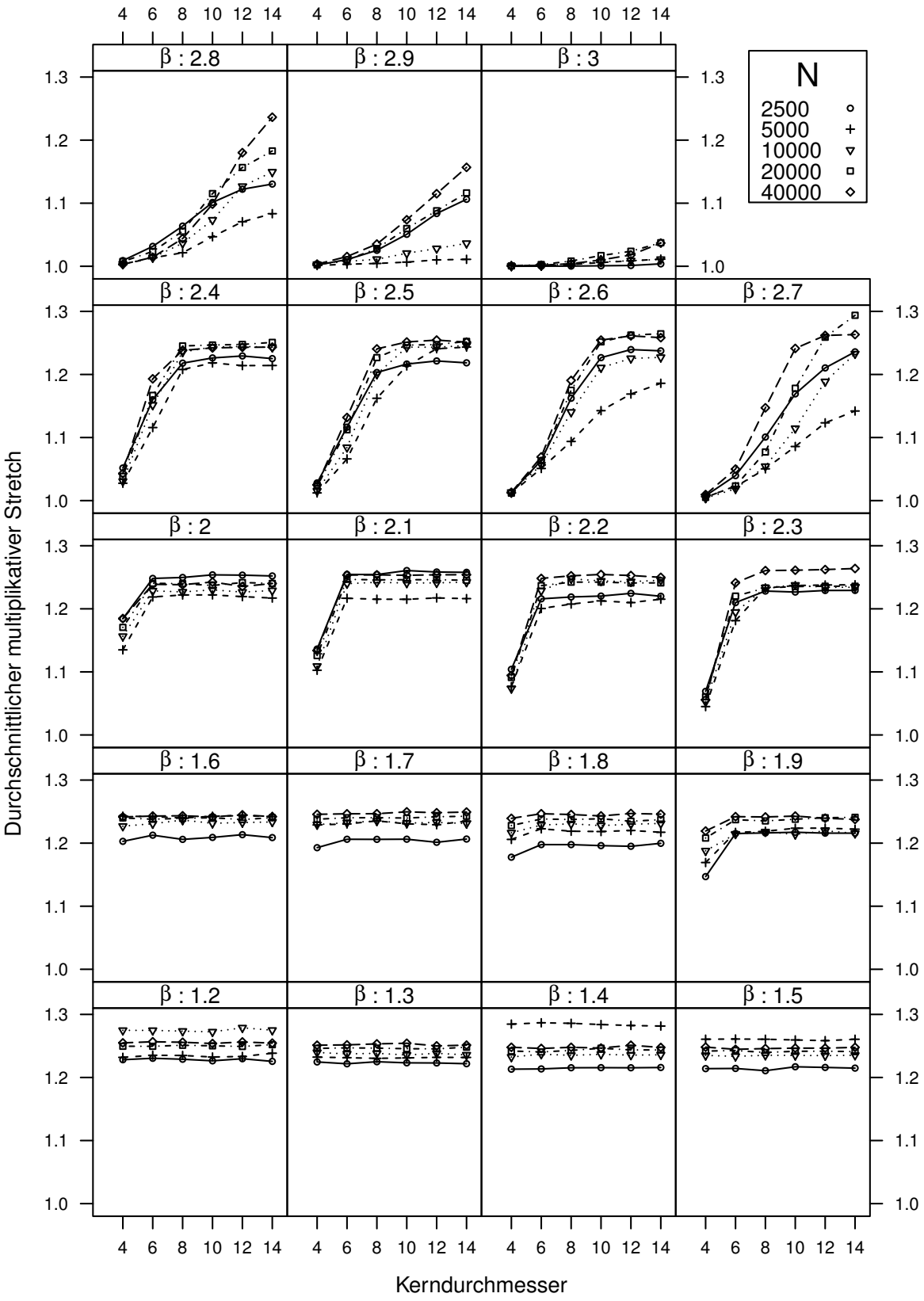


Abbildung 5.12: Durchschnittlicher multiplikativer Stretch für Kerndurchmesser von 4 bis 14 für die untersuchten Power-Law Exponenten $\beta = 1.2, 1.3, ..., 3.0$ und $N = 2500, 5000, 10000, 20000, 40000$ im Sparse Mode.

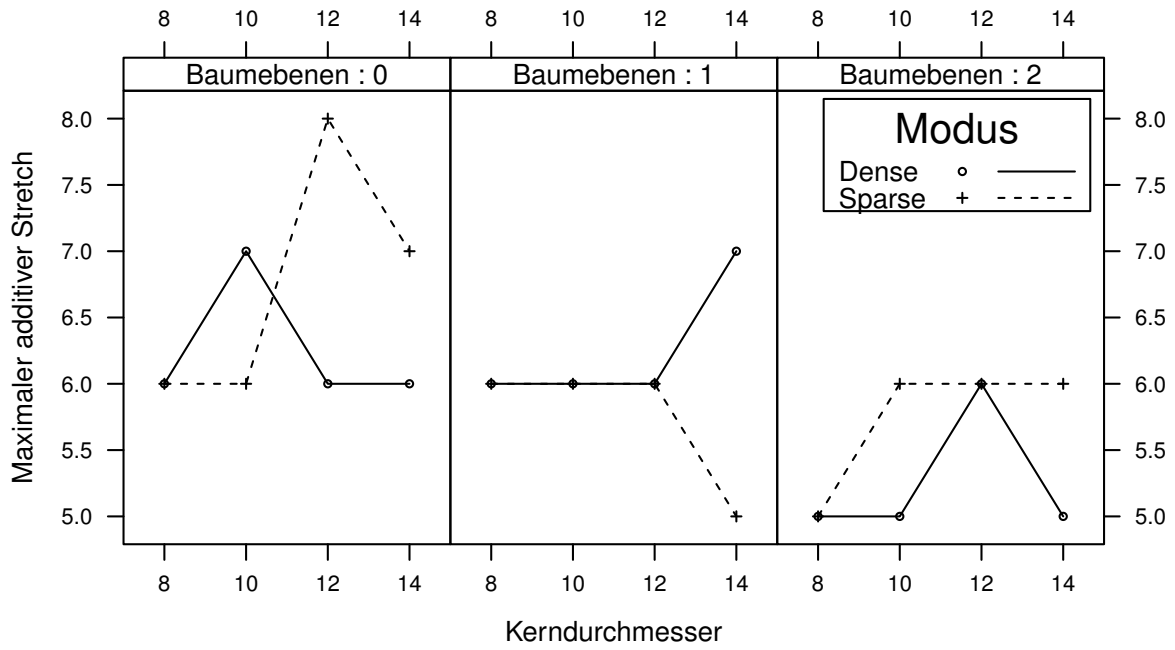


Abbildung 5.13: Maximaler additiver Stretch auf der CAIDA Topologie für Kerndurchmesser $d \in \{8, 10, 12, 14\}$ sowie 0,1 und 2 Baumebenen

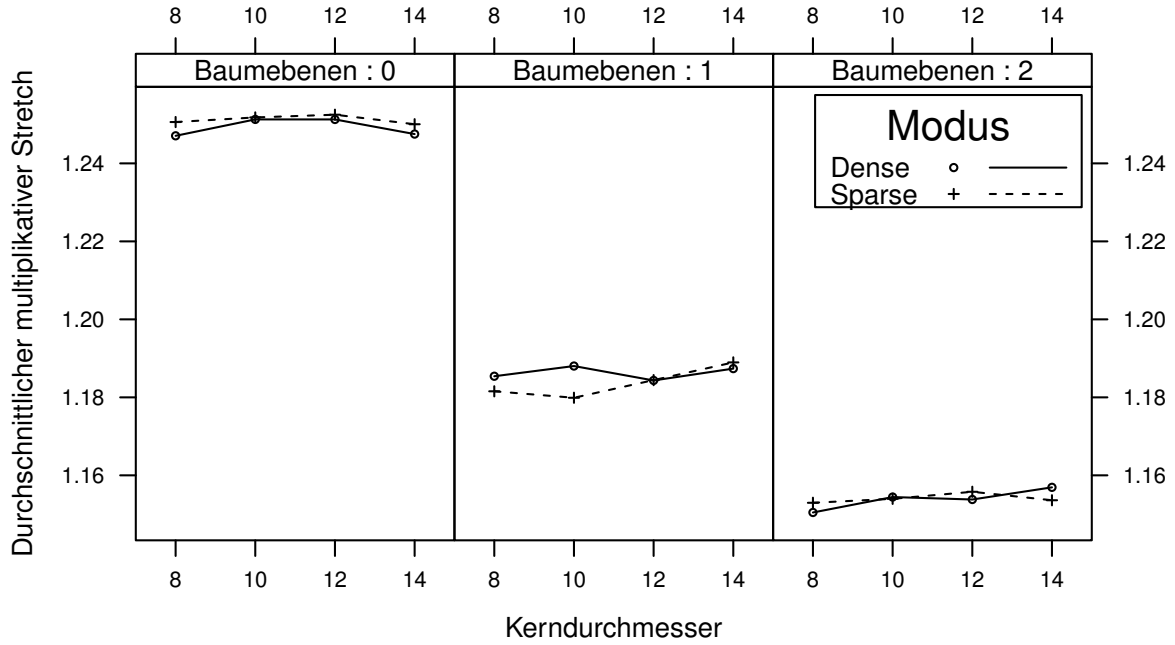


Abbildung 5.14: Durchschnittlicher multiplikativer Stretch auf der CAIDA Topologie für Kerndurchmesser $d \in \{8, 10, 12, 14\}$ sowie 0,1 und 2 Baumebenen

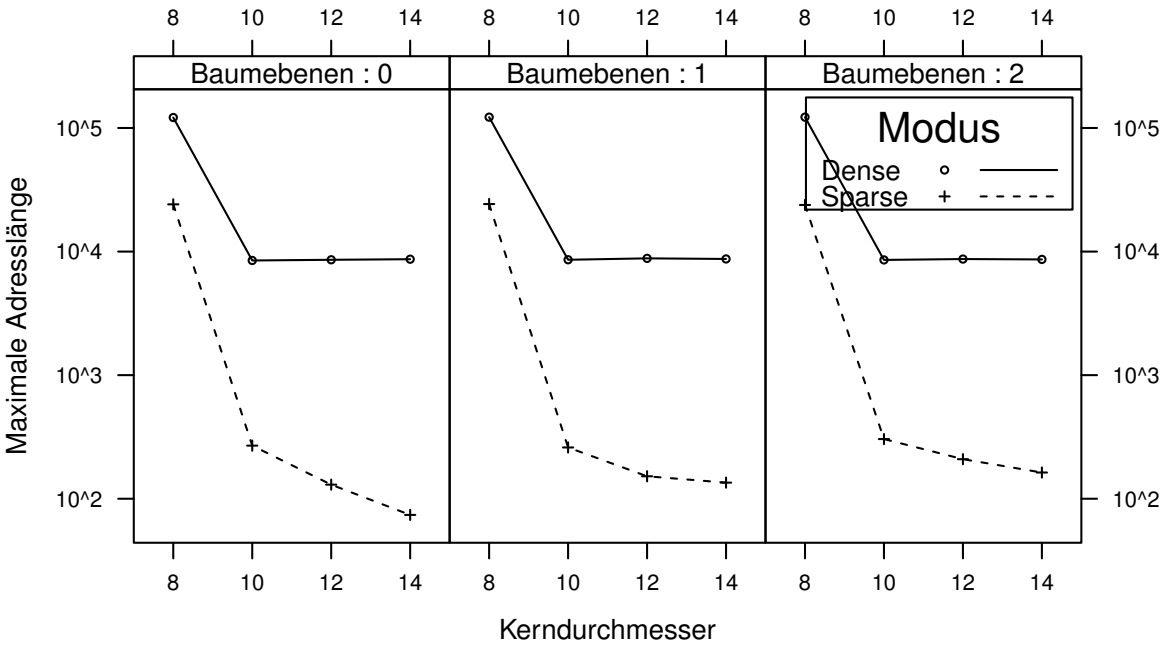


Abbildung 5.15: Maximale Adresslänge (logarithmisch) auf der CAIDA Topologie für Kerndurchmesser $d \in \{8, 10, 12, 14\}$ sowie 0,1 und 2 Baumebenen

Mode als auch den Sparse Mode untersucht. In der Betrachtung der maximalen Adresslänge wird die Stärke des Sparse Mode deutlich (siehe detaillierte Ergebnisse in Tabelle 5.4). Dieser reduziert die maximale Adresslänge gegenüber dem Dense Mode beispielsweise bei $d = 10$ deutlich um einen Faktor von bis zu 31. Während ein Kerndurchmesser von $d = 8$ in beiden Modi nicht praxistauglich ist, erlaubt der Sparse Mode für größere Kerndurchmesser eine deutliche Reduktion der Adresslänge. Die bei einem Durchmesser von $d = 10$ ohne zusätzliche Bäume maximale Adresslänge von 269 Koordinaten entspricht bei ungewichteten Kanten einer Adresslänge von 68 Bytes. Diese für nur wenige Knoten angenommene maximale Adresslänge ist zwar immer noch sehr hoch und liegt außerhalb der zuvor definierten 100 Koordinaten für eine praktikable Adresslänge, ist aber theoretisch immer noch in einem Paketkopf verwendbar. Zudem liegt die durchschnittliche Adresslänge in dieser Konfiguration bei nur 28 Koordinaten, was bei ungewichteten Kanten einer Adresslänge von 7 Byte entspricht und sogar kürzer als eine IPv6-Adresse ist. Die Konfigurationen mit $d \geq 10$ im Sparse Mode werden damit als praktikabel betrachtet. Im Gegensatz dazu wären maximale Adresslängen von mehreren tausend Koordinaten, wie sie im Dense Mode auftreten, nicht mehr im Paketkopf unterzubringen.

Durch die *Trennung der Belange* ist es auf dieser größten untersuchten Topologie unter Verwendung des Sparse Mode möglich, im Gegensatz zu dem BC-ähnlichen Dense Mode, praktikable Adresslängen zu erreichen. Der dabei beobachtete durchschnittliche Stretch ist mit dem im Dense Mode vergleichbar. Wie Abbildung 5.14 in Kombination mit Tabelle 5.4 zeigt, kann durch das Hinzufügen zusätzlicher Spannbäume (l) der durchschnittliche Stretch deutlich reduziert werden, während die Adresslängen nur geringfügig anwachsen.

Tabelle 5.4: Ergebnisse von Sprinkles auf der CAIDA Topologie

l	d	Modus	Max. Adress-länge	Adress-länge i. D.	Kerngröße	Anzahl Randbereiche	Größter Randbereich	Max. Extrakanten-Bäume	Gesendete Sparse-Modus Nachrichten
0	8	Dense	121489	3799	72994	33117	7708	1787	0
0	8	Sparse	24103	725	72994	33117	7708	353	64025
0	10	Dense	8469	70	136928	27954	1184	564	0
0	10	Sparse	269	28	136928	27954	1184	13	12820
0	12	Dense	8579	68	172639	10280	1181	576	0
0	12	Sparse	130	27	172639	10280	1181	5	3409
0	14	Dense	8671	66	184453	3185	1162	562	0
0	14	Sparse	74	27	184453	3185	1162	2	2258
1	8	Dense	122384	3846	72994	33117	7708	1803	0
1	8	Sparse	24233	748	72994	33117	7708	354	63754
1	10	Dense	8584	94	136928	27954	1184	571	0
1	10	Sparse	260	51	136928	27954	1184	10	12396
1	12	Dense	8810	92	172639	10280	1181	587	0
1	12	Sparse	152	51	172639	10280	1181	5	3211
1	14	Dense	8711	90	184453	3185	1162	559	0
1	14	Sparse	135	50	184453	3185	1162	2	2055
2	8	Dense	122331	3839	72994	33117	7708	1796	0
2	8	Sparse	23821	768	72994	33117	7708	346	63066
2	10	Dense	8559	115	136928	27954	1184	565	0
2	10	Sparse	304	73	136928	27954	1184	10	12755
2	12	Dense	8697	114	172639	10280	1181	579	0
2	12	Sparse	209	72	172639	10280	1181	4	3973
2	14	Dense	8638	112	184453	3185	1162	559	0
2	14	Sparse	163	73	184453	3185	1162	1	1818

Während der durchschnittliche Stretch durch zwei zusätzliche Spannbäume von etwa 1.25 auf 1.16 fällt, erhöht sich die durchschnittliche Adresslänge um nur etwa 20 Koordinaten pro zusätzlichem Spannbaum.

Anhand der detaillierten Ergebnisse in Tabelle 5.4 wird außerdem ersichtlich, dass durch die Verwendung des Sparse Mode nur vernachlässigbarer Kommunikationsaufwand erzeugt wird. Für die Konfiguration von $d = 10$ im Sparse Mode wird ein einmaliger Aufwand für die Initialisierung des Netzes von 10.8 Nachrichten pro Knoten verursacht, wobei die Nachrichtengröße proportional zur durchschnittlichen Adresslänge von 7 Bytes ist. Dieser Aufwand ist in der Praxis vertretbar.

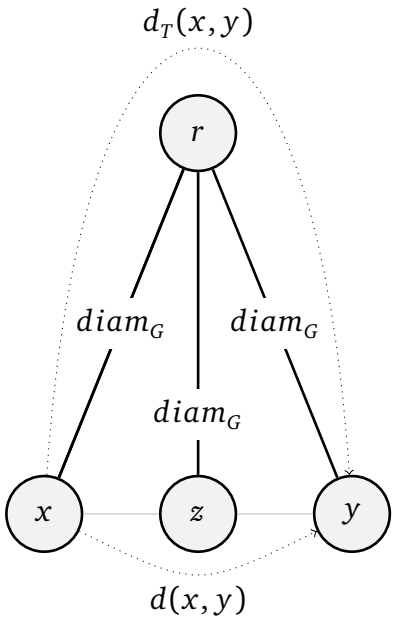


Abbildung 5.16: Maximaler Stretch von PIE

5.3.3 Bewertung der Stretch-Schranke

Beispielhaft wird die auf der CAIDA Topologie durch Sprinkles im Sparse Mode erreichbare Stretch-Schranke von $d = 10$ nun mit der unter Verwendung von PIE erreichbaren Beschränkung des Stretch verglichen.

Für zwei Knoten x, y tritt der maximale Stretch bei *Verwendung von PIE* genau dann auf, wenn diese zwei nicht direkt benachbarte Blattknoten eines Baumes sind und die Wurzel r dieses Baumes sich in der Entfernung $diam_G$ von beiden Knoten befindet, wie auch in Abbildung 5.16 dargestellt. Der Zwischenknoten z ist für die Konstruktion des schlechtesten Falls erforderlich, da Pakete unter direkt benachbarten Knoten sonst direkt und damit ohne Stretch zugestellt würden. Es ergibt sich damit ein additiver Stretch von

$$s_a = d_T(x,y) - d(x,y) = 2\text{diam}_G - 2 = 2(\text{diam}_G - 1)$$

und ein multiplikativer Stretch von

$$s_m = d_T(x,y)/d(x,y) = 2\text{diam}_G/2 = \text{diam}_G.$$

Abhängig von der realen Distanz zwischen zwei beliebigen Knoten, die hier mit dist bezeichnet wird, ergeben sich für $d \geq 2$ damit die folgenden Funktionen für den distanz-abhängigen maximalen Stretch von PIE:

Der maximale additive Stretch bei einer realen Distanz von *dist* zwischen zwei Knoten beträgt

$$s_a(\text{dist}) = 2\text{diam}_G - \text{dist}$$

und der maximale multiplikative Stretch beträgt

$$s_m(\text{dist}) = 2\text{diam}_G/\text{dist}.$$

Bei *Verwendung von Sprinkles* ist der additive Stretch beschränkt durch den Kerndurchmesser d. Daraus lässt sich der jeweils maximale multiplikative Stretch abhängig von der tatsächlichen Distanz dist bestimmen als

$$s_m(\text{dist}) = (\text{dist} + d)/\text{dist}.$$

Abbildung 5.17 stellt den distanzabhängigen maximalen multiplikativen Stretch von PIE und Sprinkles am Beispiel der bereits in der Evaluierung verwendeten CAIDA-Topologie gegenüber. Auf der x-Achse sind die möglichen realen Distanzen bis zum Durchmesser der Topologie von 26 Übertragungsabschnitten aufgetragen. Darüber ist der maximal mögliche multiplikative Stretch für PIE und Sprinkles dargestellt. Zudem ist der durchschnittliche gemessene Stretch als schwarzer Balken und der maximale gemessene Stretch als grauer Balken in Abhängigkeit von der jeweiligen Distanz eingezeichnet. Beim Sampling wurde dabei kein Knotenpaar mit einer Distanz von mehr als 19 Übertragungsabschnitten ausgewählt. Im Vergleich zu PIE wird durch die gezielte Konstruktion von Spannbäumen in Sprinkles der maximale multiplikative Stretch insbesondere für verhältnismäßig nahe Knotenpaare, beispielsweise mit einer Distanz von 2 bis 5 Übertragungsabschnitten, deutlich reduziert. Sprinkles ermöglicht also auf realistischen Topologien eine Reduktion des maximalen Stretch.

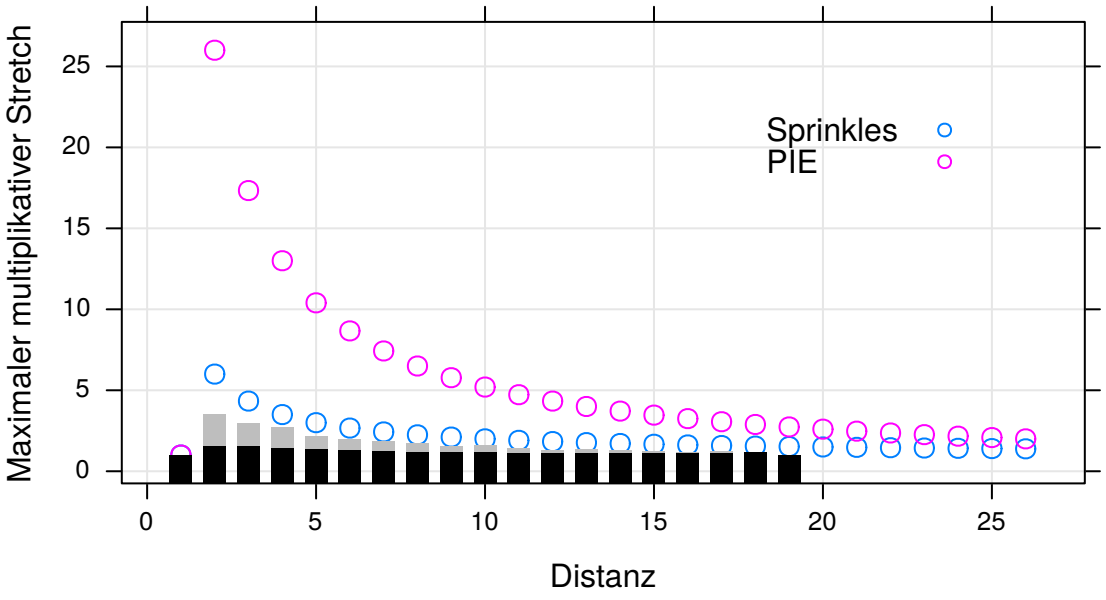


Abbildung 5.17: Vergleich des maximalen multiplikativen Stretch in Abhängigkeit von der Distanz für die CAIDA-Topologie mit Kern-Durchmesser $d = 10$

5.4 Zusammenfassung

Sprinkles realisiert in einem verteilten Routing-Protokoll die additive Stretch-Schranke des kompakten Routing-Verfahrens von Brady und Cowen. Dies wird durch die Aufgabe der oberen Schranke der Adresslänge erreicht. Statt der zentralisierten Algorithmen zur Adressvergabe beim BC Routing-Verfahren setzt Sprinkles eine verteilte isometrische Einbettung ein. Diese erzeugt in der Praxis nicht beschränkte, aber sehr gut verwendbare Adressen zur Weiterleitung entlang von Spannbäumen.

Als ein wichtigeres Problem beim Einsatz von Sprinkles wurde die Anzahl der für die Einhaltung der Stretch-Schranke notwendigen Spannbäume identifiziert und ein Verfahren zur Reduktion der erforderlichen Spannbäume entwickelt. Der im Rahmen dieser Arbeit entwickelte Sparse Mode reduziert die Anzahl der zu konstruierenden Spannbäume deutlich. Es wurde theoretisch und empirisch gezeigt, dass dabei die Beschränkung des Stretch intakt bleibt.

In einer ausführlichen Untersuchung synthetischer Topologien wurden praktikable Parameterbereiche identifiziert und die Ursachen analysiert, wieso gewisse Parameterkombinationen zu nicht praktikablen Adresslängen führen. Der Sparse Mode von Sprinkles konnte auf den synthetischen Topologien die problematischen Adresslängen zwar deutlich reduzieren aber diese nicht in einen praktikablen Bereich von maximal 100 Koordinaten bringen.

Auf der CAIDA Router-Level Topologie hingegen konnte der Sparse Mode deutliche Verbesserungen erreichen. Während der Dense Mode auch bei einem Kerndurchmesser von $d = 14$ bei weitem keine praktikablen Adresslängen erreicht, kann unter Verwendung des Sparse Mode der additive Stretch, wie in Abschnitt 5.3.1.4 erläutert, bei nahezu praktikablen Adresslängen (durchschnittliche Adresslänge kleiner als eine IPv6-Adresse) auf $d = 10$ beschränkt werden. Dies ist, wie in Abschnitt 5.3.3 gezeigt, eine deutliche Verbesserung gegenüber der Verwendung von beispielsweise PIE. Für die CAIDA Topologie wurde zudem gezeigt, dass die initialen Kommunikationskosten zur Bestimmung der erforderlichen Spannbäume im Sparse Mode sehr gering sind.

Damit lassen sich die eingangs formulierten Fragestellungen wie folgt beantworten:

V	Kann das zentralisierte Routing-Verfahren von Brady und Cowen als verteiltes Routing-Protokoll umgesetzt werden?

Mit Sprinkles wurde durch die verteilte Konstruktion der notwendigen Spannbäume die Stretch-Schranke des BC Routing-Verfahrens in einem verteilten Protokoll umgesetzt. Bei der Bestimmung der Adressen wurde als Kompromiss die beim BC Routing-Verfahren durch zentralisierte Algorithmen mögliche Längenbeschränkung der pro Baum erforderlichen Labels von $O(\log^2 n)$ aufgegeben. Stattdessen verwendet Sprinkles eine isometrische Einbettung zur Bestimmung der Adressen pro Baum, die in der Praxis gute Adresslängen in $O(\log^3 n)$ erreicht [44], die aber nicht beschränkt sind. Mit dem Sparse Mode wurde gemäß dem Prinzip „Trennung der Belange“ eine klare Unterscheidung zwischen für die Einhaltung der Stretch-Schranke notwendigen Spannbäumen und optionalen, für die Reduktion des durchschnittlichen Stretchs einsetzbaren Spannbäumen vorgenommen. Durch den Einsatz des

Sparse Mode kann die Anzahl der erforderlichen Spannbäume und damit die Adresslänge und Routing-Tabellengröße deutlich reduziert werden.

V-1 | Ist damit auf Internet-ähnlichen Topologien eine Verbesserung des maximalen Stretch gegenüber alternativen aktuellen Ansätzen erreichbar?

Als umfangreichste Topologie wurde hier die CAIDA Topologie, die das Internet auf Ebene von Routern beschreibt, untersucht. Für diese Topologie mit fast zweihunderttausend Knoten wurde in Abschnitt 5.3.3 gezeigt, dass der maximale Stretch gegenüber Ansätzen wie beispielsweise PIE deutlich reduziert werden kann. Der Sparse Mode trägt dazu entscheidend bei, da erst mit diesem eine Beschränkung des additiven Stretch von $d = 10$ bei praktikablen Adresslängen möglich ist.

V-2 | Was sind die Einschränkungen eines solchen Routing-Protokolls?

Die Anwendbarkeit von Sprinkles auf einer gegebenen Topologie ist abhängig von einer geeigneten Parametrisierung des Kerndurchmessers d. Wird dieser zu klein gewählt, resultieren daraus inpraktikable Adresslängen. Bei zu hoher Wahl von d wird nicht das gesamte Potential von Sprinkles ausgeschöpft. Hier stellt sich auch für weitere Arbeiten die Frage, welche Informationen über eine Topologie für eine gute Bestimmung des idealen Kerndurchmessers notwendig sind und wie diese damit erfolgen kann.

Ein Problem, das in diesem Kapitel nicht behandelt wurde, ist das Verhalten solcher spannbaum-basierter gieriger Routing-Verfahren wie Sprinkles unter Einbezug von Dynamik der Topologie in Form von Knoten- und Kantenausfällen. Bereits für einen einzelnen Spannbaum führt eine Knotenausfall immer zu einer Partitionierung der Topologie, die aufwändig zu reparieren sein kann. Diese Problemstellung wird im folgenden Kapitel analysiert und eine entsprechende Lösung entwickelt.

KAPITEL 6

Greedy Failure-Carrying Packets

Viele Verfahren für skalierbares Routing basieren, ähnlich zu dem zuvor in Kapitel 5 vorgestellten Sprinkles, auf der Konstruktion von gierig eingebetteten Spannbäumen [40, 41, 42, 14, 43, 44]. Auf diesem Ansatz basierende Routing-Protokolle erreichen in der Praxis auf Internet-ähnlichen Topologien meist einen sehr niedrigen durchschnittlichen Stretch bei gleichzeitig kleinen Routing-Tabellen.

Ein zentrales, oft unberücksichtigtes Problem für den praktischen Einsatz dieser Verfahren ist jedoch der hohe Aufwand, den diese Routing-Protokolle unter Dynamik verursachen. Dynamik bedeutet dabei an dieser Stelle den Ausfall oder das Hinzukommen von Knoten oder Kanten zu einer existierenden Topologie. Eine detaillierte Diskussion aller Fälle bietet Anhang A.1, während im folgenden beispielhaft der Ausfall eines Knotens betrachtet wird. Abbildung 6.1 stellt einen Spannbaum T_1 mit einer Wurzel r_1 und einem ausgefallenen Knoten u dar. Kanten des Spannbaums sind dabei mit dicken schwarzen Linien dargestellt, während Nicht-Spannbaumkanten mit dünnen grauen Linien dargestellt sind. Der Ausfall

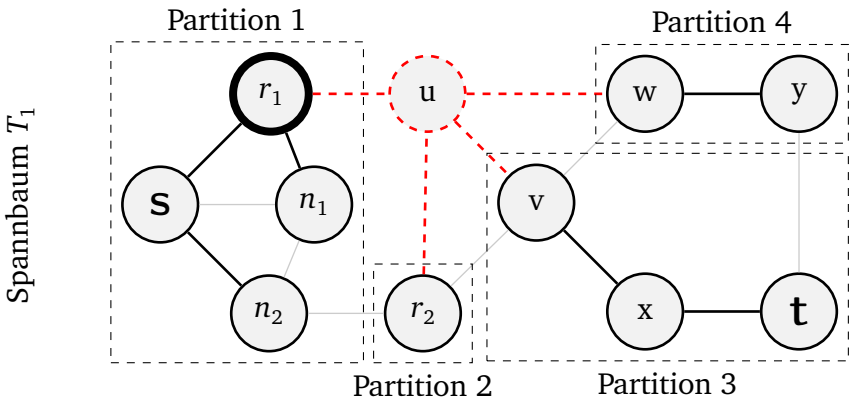


Abbildung 6.1: Auswirkungen eines Knotenausfalls

des Knotens u ist hier gleichbedeutend mit dem Ausfall aller seiner Kanten, die als rot gestrichelte Linien dargestellt sind. Der Ausfall dieser vormaligen Spannbaumkanten zerteilt den Spannbaum in vier Partitionen. Da Pakete üblicherweise entlang des Spannbaums weitergeleitet werden, ist die Erreichbarkeit zwischen den Partitionen nun nicht mehr gewährleistet und Pakete müssen bei der verwendeten gierigen Weiterleitung von Paketen an durch den Ausfall entstandenen lokalen Minima verworfen werden.

Der Aufwand für eine Reparatur dieser Situation, um wieder konsistente Routing-Tabellen zu erhalten, setzt sich zusammen aus

1. der Reparatur des Spannbaums, so dass alle Knoten wieder über einen kürzesten Pfad an die Wurzel r_1 angebunden sind.
2. einer erneuten Durchführung der gierigen Einbettung und der daraus resultierenden Neu-Vergabe von virtuellen Koordinaten in Teilen des Spannbaums.
3. der Aktualisierung der Adresse im Namensdienst, wenn sich diese durch eine Änderung einer virtuellen Koordinate geändert hat.

Erst nachdem diese Reparatur durchgeführt wurde, können sich wieder alle Knoten gegenseitig erreichen. Treten während oder nach der Reparatur weitere Fehler auf, muss die Prozedur wiederholt werden.

Der durch Ausfälle resultierende, hohe Reparaturaufwand stellt für große Topologien ein Problem dar. Insbesondere mit der im Netz eines amerikanischen Internetanbieters gemachten Beobachtung, dass viele praktisch auftretende Ausfälle Kanten betreffen und von diesen Kantenausfällen etwa 75% weniger als zwei Minuten andauern [3], stellt sich die Frage nach einer alternativen Behandlung solcher Ausfälle, die den enormen Reparaturaufwand vermeidet. Eine Möglichkeit zur Reduktion des Reparaturaufwandes besteht darin, die Reparatur nur periodisch durchzuführen. Zwischenzeitlich auftretende Fehler werden dabei nicht behandelt und die Auswirkungen der resultierenden inkonsistenten Routing-Tabellen beispielsweise in Form nicht zugestellter Pakete bis zum nächsten Reparaturzeitpunkt werden akzeptiert. Während die periodische Reparatur der Routing-Tabellen zur Reduktion des Reparaturaufwandes sinnvoll ist, stellt sich nun die Frage, ob die Auswirkungen durch die zwischen Reparaturen inkonsistenten Routing-Tabellen, die sich in Form von verworfenen Paketen äußern, weiter eingeschränkt werden können.

In diesem Kapitel werden daher die folgenden Fragestellungen im Hinblick auf die **Robustheit spannbaum-basierter gieriger Routing-Protokolle** betrachtet:

R	Wie können spannbaum-basierte gierige Routing-Protokolle zur Vermeidung einer aufwändigen Reparatur robuster gestaltet werden, so dass die Auswirkungen von kurzlebigen Ausfällen auf die Qualität des Routings, gemessen durch die Anzahl zustellbarer Pakete, möglichst gering sind?
R-1	Was sind die Kosten eines solchen Verfahrens?
R-2	Inwieweit ist ein solches Verfahren praktisch einsetzbar?

Als Lösung für diese Fragestellungen wurde im Rahmen dieser Arbeit Greedy Failure-Carrying Packets (GFCP) [9] entwickelt. Eine erste Implementierung und Evaluierung dieser Idee

wurde im Rahmen einer Diplomarbeit durchgeführt [94]. Während Pakete bei gieriger Weiterleitung an lokalen Minima, die beispielsweise durch den Ausfall eines Knotens entstanden sind, verworfen werden, ermöglicht GFCP, dass diese über eventuell vorhandene längere Wege mit niedrigem Stretch zugestellt werden können. Dazu kodiert GFCP beim Antreffen eines lokalen Minimums eine Beschreibung des ursächlichen Fehlers in das Paket. In nachfolgenden Weiterleitungsentscheidungen werden die in einem Paket enthaltenen Fehlerbeschreibungen dann dazu genutzt, um Pfade mit bereits bekannten Fehlern auszuschließen und dadurch nur Pfade auszuwählen, die bisher als fehlerfrei bekannt sind. GFCP erreicht dadurch auch in Gegenwart von Routing-Tabellen, die nicht an eine durch Ausfälle veränderte Topologie angepasst wurden, eine signifikante Erhöhung der zustellbaren Pakete.

Nach einer detaillierten Beschreibung der Zielsetzungen von GFCP wird zunächst das bisherige Netzwerkmodell um eine Formulierung für Dynamikaspekte erweitert und ein Überblick über existierende Ansätze gegeben. Anschließend wird GFCP zunächst in einer allgmeinen Form vorgestellt und anschließend anhand einer konkreten gierigen Einbettung für den praktischen Einsatz optimiert und evaluiert. Dabei werden die praktische Anwendbarkeit von GFCP sowie dessen Stärken und Schwächen diskutiert und bewertet. Das Kapitel schließt mit einer Zusammenfassung.

6.1 Zielsetzung

Greedy Failure-Carrying Packets ist eine Rerouting-Strategie für Routingprotkolle, die auf spannbaum-basierten gierigen Einbettungen aufbaut. Beim Entwurf von GFCP spielten die folgenden Ziele eine entscheidende Rolle:

Erweiterung GFCP soll als Erweiterung zur Fehlerbehandlung für die zuvor erwähnte Klasse von Routingprotokollen dienen, die auf der gierigen Einbettung von Spannbäumen basieren.

Hohe Zustellrate Während es durch den Verzicht auf eine Aktualisierung der Routing-Tabellen zum Verwerfen von Paketen an lokalen Minima kommen kann, ist es das Ziel von GFCP eine möglichst hohe Zustellrate zu erreichen.

Geringer Mehraufwand GFCP passt die Weiterleitungsentscheidung abhängig von den Fehlern, die ein Paket bisher angetroffen hat, an und verursacht dadurch an mehreren Stellen Mehraufwand, der jedoch möglichst gering ausfallen sollte:

1. Da GFCP Fehlerinformation in den Paketköpfen ablegt, muss diese Information möglichst kompakt kodiert werden.
2. Durch die Umleitung von Paketen um Fehlerstellen herum verlängert sich der Pfad, den ein Paket bis zu dessen Zustellung oder Verwerfen zurücklegt, und damit die Belastung des Netzes:
   - Für Umleitungen sollen nur Pfade verwendet werden, die nach aktuellem Kenntnisstand tatsächlich zum Ziel führen können und die Umleitung von Paketen soll zielgerichtet erfolgen.
   - Bezogen auf die durch die Veränderungen neu entstandene Topologie sollte möglichst geringer Stretch verursacht werden; dies gilt sowohl für den

durchschnittlichen als auch den maximalen Stretch. Während der Weiterleitung von Paketen sollte die Nutzung von Abkürzungen in Form von Nicht-Spannbaumkanten möglich sein, um zusätzlichen Stretch zu vermeiden.

3. Die Bestimmung eines alternativen Pfades unter Berücksichtigung der GFCP Fehlerinformation soll den Aufwand der Weiterleitungsentscheidung auf den Routern möglichst geringhalten.
4. In Abwesenheit von Fehlern soll kein Mehraufwand verursacht werden.

Unveränderte Routingtabellen GFCP nimmt an, dass die Routing-Tabellen für ein Netz bereits konstruiert sind. Zusätzlich gilt:

1. Es werden, neben dem üblichen Aufbau der Routing-Tabellen, keine zusätzlichen Informationen, wie beispielsweise Alternativpfade, vorausberechnet und in den Routingtabellen gespeichert.
2. Die zuvor konstruierten Routing-Tabellen werden beim Ausfall von Knoten oder Kanten nicht an diese Änderungen angepasst. Dies bedeutet einerseits, dass keinerlei aufwändige Aktualisierungen in der Kontrollebene stattfinden müssen. Andererseits kann dadurch die gierige Einbettung zerstört werden und die Weiterleitung von Paketen nach dem Greedy-Prinzip ist nicht mehr in allen Fällen möglich. Dies kann zu Paketverlusten führen.

Änderungen der Topologie Wie bereits eingangs erwähnt, geht GFCP davon aus, dass permanente Änderungen der Topologie durch einen periodischen Neuaufbau der Routing-Tabellen behandelt werden. GFCP hat an dieser Stelle zum Ziel, kurzfristige Topologie-Änderungen, wie sie in der Praxis häufig auftreten, zu behandeln und die Zustellrate in Gegenwart inkonsistenter Routing-Tabellen zu erhöhen. In einer Studie auf dem Backbone-Netzwerk der Sprint Corporation [3] wurde gezeigt, dass etwa 55% der Ausfälle von fehleranfälligen Kanten weniger als 30s lang dauerten. Ziel von GFCP ist es, durch die Duldung inkonsistenter Routing-Tabellen die Zeitintervalle zwischen zwei Reparaturen zu verlängern und so den Reparaturaufwand zu senken oder im Falle eines nur kurzfristigen Ausfalls sogar überflüssig zu machen. Aufgrund ihrer Häufigkeit ist insbesondere die effiziente Behandlung von Kantenausfällen ein wichtiges Ziel.

Keine zusätzliche Signalisierung GFCP soll mit den lokal auf einem Knoten vorhandenen Informationen arbeiten und keine zusätzlichen Kontrollnachrichten beispielsweise zur Suche alternativer Pfade für die Umleitung von Paketen benötigen.

6.2 Netzwerkmodell

Da bisher nur statische Netze betrachtet wurden, wird für GFCP hier ein erweitertes Netzwerkmodell zugrundegelegt, das auch Dynamik-Aspekte umfasst.

Ausgehend von einem Graphen $G = (V, E)$ mit $|V| = n$, $|E| = m$, wird an dieser Stelle angenommen, dass l gierig eingebettete Spannbäume auf dem Graphen aufgebaut wurden. Über die konkrete Art der gierigen Einbettung werden hier bewusst noch keine Annahmen

getroffen, da GFCP für alle einsetzbar ist. Über die l Spannbäume können Pakete in Abwesenheit von Ausfällen unter Verwendung von gieriger Paketweiterleitung zwischen beliebigen Knotenpaaren erfolgreich zugestellt werden.

Für die Beschreibung von Dynamik-Aspekten werden an dieser Stelle nur Ausfälle von Knoten oder Kanten betrachtet, da Ausfälle im Gegensatz zu einer Erweiterung der Topologie durch die Entstehung lokaler Minima die gegenseitige Erreichbarkeit der Knoten einschränken (siehe Anhang A). Weiterhin werden Knotenausfälle gleichgesetzt mit dem Ausfall all ihrer Kanten. Dies ist sinnvoll, da auch in der Praxis in der Regel nicht festgestellt werden kann, ob nur die Kante zu einem Nachbarn ausgefallen ist oder sogar der Nachbar selbst. Damit müssen zur Beschreibung von Topologieänderungen ausschließlich Kanten berücksichtigt werden. Zu einem Beobachtungszeitpunkt ist dabei eine Menge $F = \{e_1, e_2, ..., e_k\}, |F| = k$ von Kanten ausgefallen und kann nicht zur Weiterleitung von Paketen genutzt werden. Die Menge F ausgefallener Kanten ist nicht a-priori bekannt. Dass eine Kante $e = (u, v)$ ausgefallen ist, ist zunächst nur für direkt an die Kante angrenzenden Knoten u, v sichtbar.

6.3 Existierende Rerouting-Verfahren

Im Folgenden werden einige Rerouting-Verfahren vorgestellt und dazu zunächst Protokolle aus dem Bereich des Internet-Routings betrachtet. Die meisten dieser Ansätze setzen voraus, dass durch ein Link-State Routing-Protokoll eine vollständige Sicht auf die Topologie zur Verfügung steht, wodurch entweder die Vorausberechnung von Alternativpfaden oder die bedarfsgesteuerte Berechnung eines alternativen Pfades unter Einbezug von bekannten Fehlern ermöglicht wird. Während diese Routing-Protokolle damit für das hier betrachtete spannbaum-basierte gierige Routing nicht anwendbar sind, sind die Art und Weise der Fehlerbehandlung dennoch von Interesse. In einem zweiten Abschnitt werden Rerouting-Ansätze für gierige Routing-Verfahren skizziert, die allerdings teilweise spezielle Topologien voraussetzen.

6.3.1 Internet-Routing

Aufgrund der besonderen Bedeutung der im Internet eingesetzten Routing-Protokolle existieren für diese viele Ansätze zur Behandlung von Fehlern, die wie beispielsweise das im Folgenden vorgestellte IP Fast Reroute Framework standardisiert wurden. Daneben stellen neue Ansätze wie das im weiteren Verlauf vorgestellte Failure-Carrying Packets alternative Ansätze für Rerouting-Verfahren dar.

6.3.1.1 IP Fast Reroute Framework

Das von der IETF erarbeitete IP Fast Reroute Rahmenwerk [95] hat die Entwicklung von Rerouting-Verfahren zum Ziel, die nach dem Auftreten eines einzelnen Fehlers schnellstmöglich auf einen funktionierenden, vorberechneten Alternativpfad umschalten können. So werden nur minimale Paketverluste während des Umschaltvorgangs verursacht statt auf die längerwährende Konvergenz des Routing-Protokolls warten zu müssen. Der Fokus bei

der Entwicklung konkreter Verfahren innerhalb dieses Rahmenwerks liegt dabei auf Link-State Routing-Protokollen. Diese sind dadurch, dass sie jedem Router eine vollständige Sicht auf das Netz zur Verfügung stellen besonders für diesen Ansatz geeignet, da so eine Vorberechnung von Alternativpfaden möglich wird. Mögliche Verfahren dafür sind beispielsweise Equal Cost Multipath, Loop-Free Alternates [96, 97] oder Not-Via Addresses [98]. Während dabei zwar kein zusätzlicher Nachrichtenaufwand anfällt, ist das Fast Reroute Framework durch die angenommene globale Sicht jedes Knotens sowie die Vorberechnung von Alternativpfaden nicht für das hier betrachtete Szenario geeignet.

6.3.1.2 Resilient Routing Tables

Resilient Routing Tables, also widerstandsfähige Routing-Tabellen, speichern wie auch im IP Fast Reroute Framework, zusätzliche Einträge, die im Fehlerfall ein sofortiges Umschalten auf einen alternativen Pfad erlauben. Eine theoretische Analyse solcher widerstandsfähiger Routing-Tabellen [99] liefert als positives Ergebnis, dass dieser Ansatz für einzelne Fehler in polynomieller Zeit berechenbar ist. Ein negatives Ergebnis der Untersuchung bezieht sich auf sogenannte perfekt widerstandfähige Routing-Tabellen. Diese stellen für alle möglichen Fehlerkombinationen, die auftreten könnten, die gegenseitige Erreichbarkeit von Knoten sicher, solange diese noch untereinander verbunden sind. Dabei wurde gezeigt, dass es Netze gibt, für die perfekte widerstandsfähige Routing-Tabellen nicht möglich sind. Sowohl Resilient Routing Tables als auch das verwandte Failure-Insensitive Routing [100] setzen eine vollständige Sicht der Knoten auf das Netz zur Bestimmung der Alternativpfade voraus.

6.3.1.3 Crankback Routing

Crankback Routing [101, 102] ist ein auf Kontroll-Ebene arbeitendes Rerouting-Verfahren für leitungsvermittelte oder auf virtuellen Verbindungen arbeitende Netze. Dabei verfügen Knoten durch ein Link-State Routing-Protokoll über eine vollständige Sicht auf die Topologie und signalisieren aktiv den Aufbau von Pfaden ausgehend von sogenannten Initiator.

Beim Aufbau eines Pfades ist es möglich, dass ein Zwischensystem ausgefallen ist oder nicht die erwünschte Dienstgüte zusichern kann. In diesem Fall erfolgt eine Signalisierung vom verursachenden System zurück in Richtung des Initiators, die eine Beschreibung der Fehlerstelle sowie die Ursache dafür enthält. Auf dem Rückweg zum Initiator befindliche Knoten können nun anhand der enthaltenen Fehlerinformation und ihrer lokalen Routing-Tabelle einen neuen Weg unter Ausschluss der Fehlerstelle bestimmen. Die Pfadaufbau-Nachricht wird dann über diesen neuen Weg in Richtung des Ziels weitergeleitet oder, falls kein solcher existiert, weiter zurück in Richtung des Initiators gesendet. Crankback Routing arbeitet im Gegensatz zu dem im Rahmen dieser Arbeit entwickelten Rerouting-Verfahren in der Kontroll-Ebene. Wie auch die vorhergehenden Verfahren setzt Crankback Routing ein Link-State Routing-Protokoll und damit eine vollständige Sicht auf die Topologie voraus.

Algorithmus 2 Paketweiterleitung Failure-Carrying Packets

```
 1: G                                                    ▷ Langfristige Topologie
 2: pkt.failedLinks ← NULL          ▷ Ein Paket trägt zunächst keine Fehlerinformation
 3:
 4: procedure FORWARDFCP(pkt)                            ▷ Weiterleitung des Pakets pkt
 5:     while True do
 6:         path ← ComputePath(G\pkt.failedLinks)        ▷ Berechne (nächst-)besten Pfad
 7:         if path == NULL then
 8:             drop(pkt)
 9:             return
10:         else if hasFailed(path.nexthop) then
11:             pkt.failedLinks ← pkt.failedLinks ∪ failedLink(path.nexthop)
12:         else
13:             forward(pkt, path.nexthop)
14:             return
15:         end if
16:     end while
17: end procedure
```

6.3.1.4 Failure-Carrying Packets

Failure-Carrying Packets [103] ist eine Rerouting-Strategie für Link-State Routing-Protokolle. Failure-Carrying Packets (FCP) nutzt dabei aus, dass jeder Knoten nach bei Link-State Routing die gleiche, vollständige Sicht auf die Topologie hat. Wenn bei Link-State Routing eine Änderung der Topologie auftritt, so muss das Routing-Protokoll zunächst wieder konvergieren bis alle Knoten die gleiche Sicht auf das Netz haben und damit alle Pakete zugestellt werden können. FCP will diese Konvergenzzeit, die beispielsweise durch Neuberechnung von kürzesten Pfaden auch aufwändig ist, eliminieren und dabei trotzdem alle Pakete zustellen, solange die zugrundeliegende Topologie noch verbunden ist.

FCP unterteilt dazu auftretende Topologieänderungen in zwei Klassen: *Langfristige* Änderungen der Topologie, wie beispielsweise das geplante Einfügen eines neuen Links, der danach dauerhafter Bestandteil der Topologie ist und *kurzfristige* Änderungen der Topologie, wie den temporären Ausfall eines an sich dauerhaften Links. Langfristige Topologieänderungen werden dabei von einem zentralen Koordinator im Netz verteilt und dadurch die gemeinsame Sicht auf die langfristige Topologie konsistent gehalten. Die Behandlung von kurzfristigen Topologieänderungen erfolgt durch gezielte Umleitung von Paketen, die in der resultierenden Topologie nicht mehr gemäß dem vorgesehenen optimalen Pfad weitergeleitet werden können und sonst verworfen werden müssten. Dazu werden in Datenpaketen Beschreibungen der bisher angetroffenen ausgefallenen Kanten kodiert. Nachfolgende Knoten nehmen diese ausgefallenen Kanten aus ihrer globalen Sicht auf die Topologie aus und berechnen damit den optimalen Weiterleitungsknoten für das Paket.

Die Paketweiterleitung von Failure-Carrying Packets beschreibt Algorithmus 2, der beispielhaft anhand von Abbildung 6.2 veranschaulicht wird. Knoten *s* sendet in diesem Beispiel

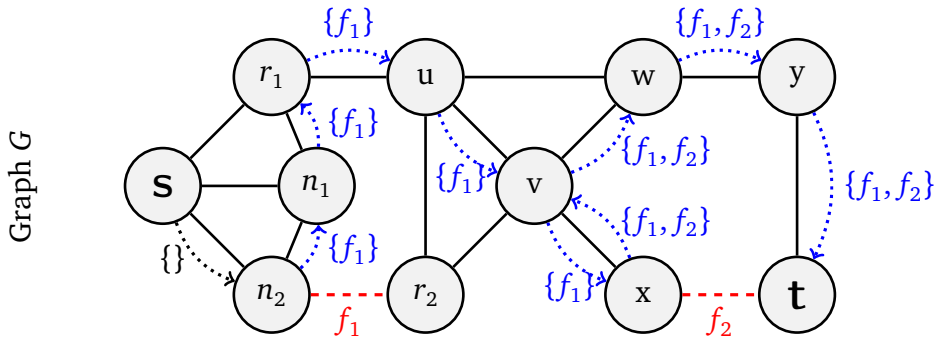


Abbildung 6.2: Beispiel für Paketweiterleitung mit Failure-Carrying Packets

ein Paket an Knoten t, bestimmt anhand seiner Routing-Tabellen Knoten n_2 als optimalen Weiterleitungsknoten (Zeile 6) und leitet das Paket an diesen weiter (Zeile 13). Dieses Paket enthält noch keine Fehlerbeschreibung. Fehlerbeschreibungen sind in Abbildung 2 jeweils in geschweiften Klammern neben dem durch gepunktete Pfeile dargestellten Weg des Pakets dargestellt. Nach dem Empfang des Pakets bestimmt Knoten n_2 den Knoten r_2 als optimalen Weiterleitungsknoten (Zeile 6). Da die Kante zu diesem Knoten jedoch ausgefallen (rot gestrichelt) ist und der Weiterleitungsknoten damit nicht erreichbar ist (Zeile 10), wird die ausgefallene Kante f_1 dem Paketkopf hinzugefügt (Zeile 11). Unter deren Ausschluss wird Knoten n_1 als nächstbeste Weiterleitungsmöglichkeit berechnet (Zeile 6) und das Paket an diesen weitergeleitet (Zeile 13). Die nachfolgenden Knoten r_1, u und v verfahren ebenso bis das Paket zu Knoten x gelangt. Dort wird als optimaler Weiterleitungsknoten zunächst der Zielknoten t selbst bestimmt. Da die zu diesem führende Kante f_2 jedoch ausgefallen ist (Zeile 10), wird diese nun zusätzlich in den Paketkopf aufgenommen (Zeile 11) und unter Ausschluss der beiden ausgefallenen Kanten $\{f_1, f_2\}$ Knoten v als der nächstbeste Weiterleitungsknoten bestimmt. Unter Ausschluss der nun im Paketkopf eingetragenen Kanten wird das Paket schließlich über die Knoten w und y erfolgreich an Knoten t zugestellt.

FCP erfüllt viele der zuvor gesetzten Anforderungen. Die Routingtabellen bei FCP entsprechen der langfristigen Topologie und dadurch kann der ansonsten notwendige Aufwand in der Kontrollebene für kurzfristige Topologieänderungen eliminiert werden. Im Gegenzug dafür werden Informationen über ausgefallene Kanten in Datenpakete kodiert und nachfolgend zur Berechnung neuer kürzester Pfade genutzt. Während Kanten dank der vollständigen Sicht auf die Topologie durch jeden Knoten beispielsweise durch einen eindeutigen Identifikator kodiert werden können, erfordert die oft notwendige Neuberechnung kürzester Pfade zusätzliche Mechanismen für eine effiziente Umsetzung. Dafür werden Source-Routing, die Vorausberechnung von alternativen Pfaden, Caching und die Verwendung effizienter Algorithmen vorgeschlagen. Wie auch in der Zielsetzung gefordert, verursacht FCP aber nur zusätzlichen Aufwand, falls tatsächlich Fehler aufgetreten sind. FCP kann, sofern die zugrundeliegende Topologie verbunden ist, alle Pakete zustellen. Da das für Link-State Routingprotokolle entworfene FCP allerdings voraussetzt, dass alle Knoten eine vollständige Sicht auf die Topologie haben, ist es für den hier im Fokus stehenden Einsatz auf spannbaumbasierten gierigen Einbettungen nicht geeignet.

6.3.2 Gieriges Routing

Rerouting-Verfahren für gierige Routing-Protokolle unterscheiden sich teilweise sehr in ihren Annahmen über die zugrundeliegende Topologie. Im Folgenden wird zunächst ein Rerouting-Verfahren für planare Graphen vorgestellt und anschließend mit Gravity-Pressure Routing ein allgemein auf gieriges Routing anwendbares Rerouting-Verfahren vorgestellt.

6.3.2.1 Face Routing

Für geometrisches Routing, also beispielsweise Routing auf GPS-Koordinaten, wird oft das sogenannte Face-Routing [104] oder davon abgeleitete Verfahren zur Behandlung lokaler Minima verwendet. Bevor Face Routing eingesetzt werden kann, muss jedoch der zugrundeliegende Graph planarisiert werden; es darf also keine Kanten geben, die sich kreuzen. Damit wird es möglich, Pakete entlang sogenannter Facetten in Richtung des Ziels weiterzuleiten. Face Routing ist für das im Rahmen dieser Arbeit behandelte Szenario nicht interessant, da es in sehr hohem Stretch resultieren kann und sein Einsatz zudem auf planare Graphen beschränkt ist.

6.3.2.2 Gravity Pressure Routing

Gravity Pressure Routing [40] (GP) ist eine Rerouting-Strategie für gierige Routing-Verfahren und besteht aus zwei Weiterleitungs-Modi:

- dem *Gravity-Modus*, der einer gierigen Weiterleitung entspricht
- und dem *Pressure-Modus*, der zum Einsatz kommt, wenn ein Paket auf ein lokales Minimum trifft. Dabei wird erfasst, welche Knoten bereits wie oft besucht wurden und diese Information in nachfolgenden Weiterleitungsentscheidungen dazu genutzt, das Paket aus dem lokalen Minimum heraus in Richtung des Ziels weiterzuleiten.

Algorithmus 3 beschreibt die Weiterleitung eines Pakets *pkt* durch Knoten x an Zielknoten t mit Gravity-Pressure Rerouting. Pakete werden dabei zunächst im Gravity-Modus weitergeleitet. Empfängt Knoten x ein Paket im Gravity-Modus (Zeile 6), so wird zunächst aus allen Nachbarn von Knoten x der Knoten als Weiterleitungsknoten bestimmt, der die Distanz zum Zielknoten t minimiert (Zeile 7). Stellt diese Distanz eine Verbesserung gegenüber der Distanz von Knoten x zu Knoten t dar (Zeile 8) und ist Knoten x damit kein lokales Minimum, so erfolgt die Weiterleitung des Pakets über den gierig bestimmten Weiterleitungsknoten (Zeile 9).

Erfolgte die Weiterleitung bisher gierig und stellt Knoten x ein lokales Minimum dar, so wird das Paket in den Pressure-Modus versetzt (Zeile 11). Dabei wird die aktuelle Distanz zum Ziel im Paket vermerkt (Zeile 12). Diese Distanz kann während der Weiterleitung im Pressure-Modus dazu genutzt werden, um zu prüfen, wann eine Umschaltung in den Gravity-Modus wieder möglich ist. Erreicht das Paket nämlich später einen Knoten, dessen Distanz zum Ziel kleiner ist als die im Paket vermerkte Distanz, so kann das Paket wieder gierig weitergeleitet werden.

Ein Paket, dessen Modus zuvor auf Pressure gesetzt wurde, oder das bereits im Pressure-Modus bei Knoten x ankam, wird folgendermaßen behandelt (Zeile 16). Es wird überprüft,

Algorithmus 3 Paketweiterleitung Gravity-Pressure Rerouting

```
1: Weiterleitung des Pakets pkt durch Knoten x zum Zielknoten t
2: dist(x,y) bestimmt die gierige Distanz
3:
4:
5: procedure FORWARDGP(pkt)
6:     if pkt.mode == Gravity then
7:         nexthop ← arg min_{n∈Γ(x)} dist(n, t)
8:         if dist(nexthop,t) < dist(x,t) then
9:             forward(pkt, nexthop)
10:        else
11:            pkt.mode ← Pressure                    ▷ Umschalten in Pressure-Modus
12:            pkt.pressureEntryDist ← dist(x,t)      ▷ Distanz bei Wechsel
13:        end if
14:    end if
15:
16:    if pkt.mode == Pressure then
17:        if dist(x, t) ≥ pkt.pressureEntryDist then
18:            pkt.visited[x]+ = 1        ▷ Markiere x als besucht inkl. Zählererhöhung
19:            minVisited ← min_{n∈Γ(x)} pkt.visited[n]
20:            Candidates ← {n ∈ Γ(x)|pkt.visited[n] == minVisited}
21:            nexthop ← arg min_{n∈Candidates} dist(n, t)
22:            forward(pkt,nexthop)
23:        else
24:            pkt.mode ← Gravity
25:            Weiterleitung im Gravity-Modus
26:        end if
27:    end if
28: end procedure
```

ob Knoten x näher am Ziel liegt als die im Paket enthaltene *pressureEntryDist*. Ist dies der Fall (Zeile 23), so kann das Paket wieder in den Gravity-Modus versetzt werden und die Weiterleitung entsprechend gierig erfolgen (Zeilen 6–14). Eine frühstmögliche Rückkehr in den Gravity-Modus ist wünschenswert, da dieser durch eine zielgerichtete Weiterleitung deutlich weniger Stretch verursacht als der ineffiziente Pressure-Modus.

Ist eine Rückkehr in den Gravity-Modus nicht möglich und muss das Paket daher im Pressure-Modus weitergeleitet werden (Zeilen 18–22), trägt sich Knoten x mit seinem Identifikator und einem Zähler in eine Liste besuchter Knoten ein. Dieser sogenannte Besuchszähler (pkt.visited in Algorithmus 3) gibt dabei an, wie oft ein Knoten von einem Paket bereits besucht wurde. Ist Knoten x noch nicht im Paket pkt enthalten, so wird der Besuchszähler auf 1 gesetzt oder wenn Knoten x bereits besucht wurde, um 1 erhöht (Zeile 18). Anschließend werden diejenigen Nachbarn von x bestimmt, die durch das Paket bisher am wenigsten oft besucht wurden (Zeilen 19–20). Aus der resultierenden Knotenmenge *Candidates* wird derjenige Knoten als Weiterleitungsknoten ausgewählt, der die geringste Distanz zum Ziel aufweist und das Paket schließlich über diesen weitergeleitet (Zeile 22).

Solange der zugrundeliegende Graph verbunden ist, ist Gravity-Pressure Routing theoretisch in der Lage alle Pakete zuzustellen. Dies wird dadurch erreicht, dass anhand von Algorithmus 3 im schlechtesten Fall alle Knoten des Netzes gegegebenfalls mehrfach besucht werden können. In der Praxis müssen Pakete über eine Time-To-Live verfügen, um für Gravity-Pressure Routing eine Abbruchbedingung zu bilden und ein Paket, das nach einer gewissen Anzahl von Übertragungsabschnitten nicht zugestellt werden konnte, zu verwerfen. Dadurch kann es auch unter Gravity-Pressure Routing zu Paketverlusten kommen.

Tabelle 6.1 erläutert die Weiterleitung eines Pakets von Knoten s nach Knoten y auf der Topologie aus Abbildung 6.3 auf der ein gierig eingebetteter Spannbaum T_1 aufgebaut wurde. Einträge in der Spalte Γ(Knoten) bestehen aus Nachbarknoten und deren Distanz zum Zielknoten; beispielsweise bedeutet $r_1/3$, dass der Nachbar r_1 3 Hops vom Zielknoten y entfernt ist. Grau gefärbte Einträge sind Nachbarn, die bei einer Weiterleitung im Pressure-Modus ausgeschlossen werden, da sie nicht zu den bisher am wenigsten oft besuchten Knoten gehören. Der unterstrichene Eintrag ist dabei der schließlich als Weiterleitungsknoten gewählte Nachbar.

Knoten s bestimmt als Sender gierig den Weiterleitungsknoten r_1 und sendet das Paket im Gravity-Modus an diesen. Durch den Ausfall der Kante zwischen den Knoten r_1 und u stellt Knoten r_1 ein lokales Minimum dar und versetzt daher das Paket in den Pressure-Modus. Er trägt seine Knoten-ID mit Besuchszähler 1 sowie die aktuelle Distanz 3 zum Ziel in das Paket ein. Die verbleibenden Nachbarn s und n_1 haben im Spannbaum T_1 beide die Distanz 4 zum Ziel und sind bisher noch nicht im Pressure-Modus besucht worden. Im Rahmen dieses Beispiels wählt Knoten r_1 nun zufällig Knoten n_1 als Weiterleitungsknoten aus. Knoten n_1 betrachtet seine bisher im Pressure-Modus unbesuchten Nachbarknoten s und n_2 mit Distanz 4 und 5 zum Ziel und wählt daher Knoten s als Weiterleitungsknoten aus. Diesem verbleibt als einziger bisher im Pressure-Modus unbesuchter Nachbar Knoten n_2, an den er das Paket entsprechend weiterleitet. Über Knoten r_2 wird das Paket schließlich zu Knoten u weitergeleitet, der bisher unbesucht ist und die Distanz zum Ziel minimiert. Knoten u stellt im Gegensatz zu allen vorher im Pressure-Modus besuchten Knoten fest, dass er näher am Ziel liegt als die im Paket bei Aktivierung des Pressure-Modus aufgezeichnete Distanz von

Tabelle 6.1: Beispiel für Paketweiterleitung mit GP Routing

Schritt	Knoten	Distanz zu y	$\Gamma(Knoten)$	Pressure-Modus	Besuchte Knoten mit Besuchszähler
0	s	4	$\underline{r_1/3}, n_1/4, n_2/5$	Nein	–
1	r_1	**3**	$\underline{n_1/4}, s/4$	Ja	$<r_1, 1>$
2	n_1	4	$\underline{s/4}, n_2/5, r_1/4$	Ja	$<r_1, 1>, <n_1, 1>$
3	s	4	$\underline{n_2/5}, r_1/3, n_1/4$	Ja	$<r_1, 1>, <n_1, 1>, <s, 1>$
4	n_2	5	$\underline{r_2/3}, s/4, n_2/4$	Ja	$<r_1, 1>, <n_1, 1>, <s, 1>, <n_2, 1>$
5	r_2	3	$\underline{u/2}, v/3, n_2/5$	Ja	$<r_1, 1>, <n_1, 1>, <s, 1>, <n_2, 1>, <r_2, 1>$
6	u	**2**	$\underline{w/1}, v/3, r_2/3$	Nein	$<r_1, 1>, <n_1, 1>, <s, 1>, <n_2, 1>, <r_2, 1>$
7	w	1	$\underline{y/0}, u/2, v/3$	Nein	$<r_1, 1>, <n_1, 1>, <s, 1>, <n_2, 1>, <r_2, 1>$
8	y	0	–	Nein	$<r_1, 1>, <n_1, 1>, <s, 1>, <n_2, 1>, <r_2, 1>$

3 Hops. Damit kann für das Paket nun wieder die Weiterleitung nach dem Greedy-Prinzip aktiviert werden, wodurch dieses schließlich entlang des Spannbaums T_1 über Knoten w zum Zielknoten y zugestellt wird.

GP ist für alle Routingverfahren, die Pakete nach dem Greedy-Prinzip weiterleiten, geeignet, da keine zusätzlichen Anforderungen erfüllt werden müssen. Dieser Vorteil wird dadurch erkauft, dass in den Routingtabellen eigentlich vorhandenes Wissen über die Topologie nicht für Weiterleitungsentscheidungen genutzt wird. GP beschreibt durch die Kombination von Knoten-ID und zugehörigem Besuchszähler in den Paketköpfen lediglich, wie oft bereits erfolglos versucht wurde, das Paket über einen Knoten weiterzuleiten und bevorzugt nach dieser Metrik bessere Knoten. Dabei spielt es keine Rolle, wieso ein Knoten das Paket nicht weiterleiten konnte. GP verfügt zudem über keinen eigenen Mechanismus zur Feststellung, ob ein Paket verworfen werden kann, sondern ist auf eine korrekte Parametrisierung der Time-To-Live für das zugrundeliegende Netz angewiesen. Ist in dem vorangegangenen Beispiel zusätzlich die Kante zwischen Knoten n_2 und r_2 ausgefallen, würde das Paket bis zum Ablauf der TTL in der dadurch entstehenden linken Partition zwischen den dortigen Knoten (bzw. ohne eine TTL auch endlos) hin- und hergesendet. Dieses aggressive Weiterleitungsverhalten des Gravity-Pressure Routings bietet auf den hier im Fokus stehenden spannbaumbasierten gierigen Einbettungen die Möglichkeit, ein Paket so lange im Pressure-Modus weiterzuleiten bis es wieder auf einen gültigen gierigen Pfad in Richtung des Zielknotens stößt. Trotz einiger hier angeführter Schwächen erfüllt Gravity-Pressure Routing unter den hier vorgestellten Routing-Verfahren am ehesten die eingangs formulierten Anforderungen und wird daher auch später als Vergleichsbasis für das im Rahmen dieser Arbeit entwickelte Greedy Failure-Carrying Packets verwendet.

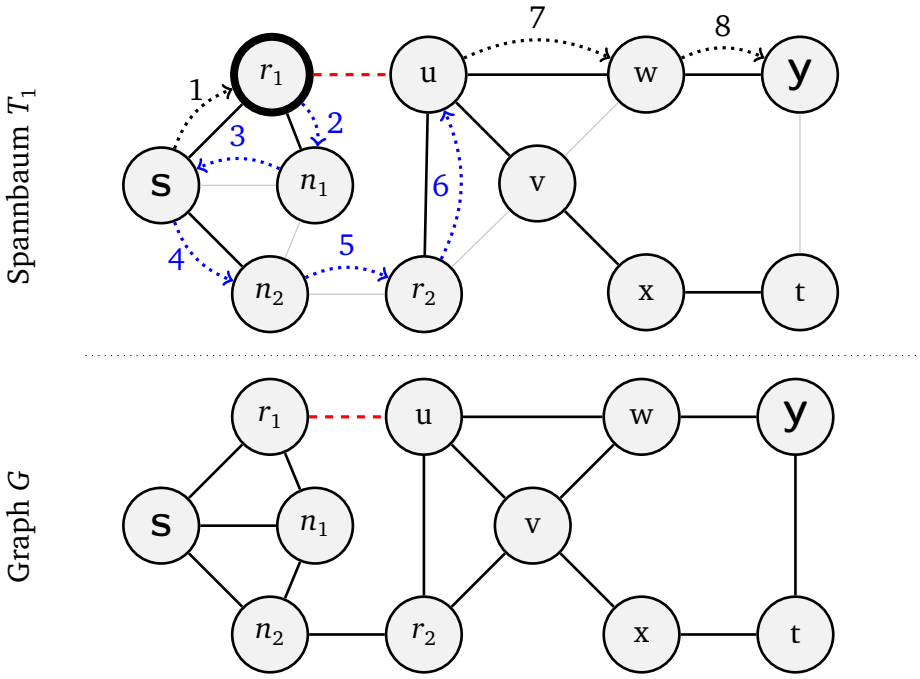


Abbildung 6.3: Beispiel für Gravity-Pressure Routing

6.3.2.3 Fehlerbehandlung für Spannbaum-basiertes gieriges Routing

Die in diesem Abschnitt vorgestellten Rerouting-Verfahren [105] dienen der Behandlung von Linkausfällen für Routing auf gierig eingebetteten Spannbäumen und haben zum Ziel, die Neubestimmung von Adressen nach Kantenausfällen zu vermeiden.

Ein erstes Verfahren behandelt dabei exklusiv einzelne ausgefallene Kanten. Nachdem ein Knoten erkannt hat, dass er ein lokales Minimum für Pakete mit einem Zielknoten t darstellt, sucht dieser aktiv per Breitensuche einen Zwischenknoten, der über einen gültigen Pfad in einen Teilbaum verfügt und über den das lokale Minimum umgangen werden kann. Pakete werden anschließend zum Zwischenknoten getunnelt und von diesem aus unter Deaktivierung von Abkürzungen über Nicht-Spannbaumkanten zum Zielknoten t weitergeleitet. Das Auffinden eines solchen Zwischenknotens verursacht dabei bereits auf Power-Law Topologien mit nur 1000 Knoten und einem Power-Law Parameter um $\beta = 3$ einen durchschnittlichen Nachrichtenaufwand von 130 Nachrichten für eine einzelne ausgefallene Spannbaumkante. Durch die teilweise Deaktivierung der Nutzung von Abkürzungskanten ist es zudem möglich, dass nicht immer der optimale gierige Pfad genutzt wird. Dieser Ansatz entspricht den gestellten Anforderungen einerseits durch die hohe Anzahl erforderlicher Kontrollnachrichten nicht und zudem wird durch die Nicht-Nutzung von Abkürzungskanten, die hier zur Vermeidung von Schleifen erforderlich ist, der Stretch erhöht.

Das obige Verfahren ist nicht für mehrere gleichzeitig auftretende Linkausfälle geeignet ist, weil dann Schleifen entstehen könnten. Daher wurde in der gleichen Arbeit ein zweites Verfahren zur Behandlung mehrerer Linkausfälle vorgeschlagen. Dabei werden disjunkte Backup-Pfade für Nachbarn im Spannbaum vorberechnet, so dass bei einem Kantenausfall ein alternativer Pfad zur Verfügung steht. Für eine Kante (u, v) des Spannbaums verfügen

Tabelle 6.2: Übersicht über Rerouting-Ansätze

Verfahren	Unveränderte Routing-Tabellen		Geringer Mehraufwand		
	Keine Voraus-berechnungen	Ohne Kon-trollebene	Geringer Overhead	Geringe Netz-belastung	Eingeschränkte Sicht
Fast Reroute	x	✓	x	✓	x
Resilient Routing Tables	x	✓	x	✓	x
Crankback Routing	✓	x	✓	✓	x
Failure-Carrying Packets	✓	✓	✓	✓	x
Face Routing	✓	✓	x	x	✓
Gravity-Pressure	✓	✓	✓	x	✓
Spannbaum-basiert	(✓)	x	x	x	(✓)

die Knoten u, v jeweils über einen alternativen Pfad zum gegenüberliegenden Knoten, der beispielsweise als Source-Route gespeichert sein kann. Wird auf diesem alternativen Pfad ein weiterer Fehler angetroffen, wird dafür ebenfalls der Backup-Pfad benutzt, um den Nachbarn zu erreichen. Das Paket wird damit rekursiv zum Ziel getunnelt. Während dieser Ansatz zur Behandlung mehrerer gleichzeitiger Linkausfälle auf dicht vermaschten Power-Law Topologien mit 1000 Knoten und einem Mindestknotengrad von 10 bei bis zu 30% Linkausfällen eine perfekte Zustellrate erreicht, setzt er voraus, dass die vollständige Topologie im Voraus bekannt ist und aufwändige Vorberechnungen darauf durchgeführt werden. Diese aufwändigen Vorberechnungen und die dafür notwendige globale Sicht auf die Topologie widersprechen der eingangs formulierten Zielsetzung.

6.3.3 Zusammenfassung

Tabelle 6.2 gibt zusammenfassend einen Überblick über die vorgestellten Rerouting-Verfahren. Die vier zuerst aufgeführten Rerouting-Verfahren setzen voraus, dass die gesamte Topologie bekannt ist und können daher nicht mit einer eingeschränkten Sicht auf die Topologie umgehen, wie sie bei spannbaum-basierten gierigen Einbettungen vorliegt. Von diesen Ansätzen ist bis auf die Voraussetzung der globalen Sicht Failure-Carrying Packets derjenige, der am ehesten den gestellten Anforderungen entspricht. Die zweite Gruppe von Rerouting-Ansätzen für gierige Routing-Verfahren unterscheiden sich in ihren Voraussetzungen an die zugrundeliegende Topologie. Face-Routing und davon abgeleitete Verfahren benötigen einerseits eine planarisierte Topologie und resultieren andererseits in einer massiven Erhöhung des Stretch. Daher wurde dieser Ansatz nicht weiter berücksichtigt. Gravity-Pressure Routing ist für alle gierigen Routing-Protokolle einsetzbar und erfüllt unter allen vorgestellten Verfahren am ehesten die Zielsetzung dieser Arbeit. Die hier bereits angemerkte hohe Netzbelastung von Gravity-Pressure Routing wird erst im weiteren Verlauf dieser Arbeit im Vergleich zu dem hier vorgestellten Greedy Failure-Carrying Packets offensichtlich, wenn die beiden Verfahren in der Evaluierung gegenübergestellt werden. Die beiden spannbaum-basierten Rerouting-Verfahren aus Abschnitt 6.3.2.3 wurden ebenfalls nicht weiter verfolgt. Das Verfahren zur Behandlung einzelner Kantenausfälle verursacht einen deutlichen Signalisierungsaufwand

für die Bestimmung der Zwischenknoten während das Verfahren zur Behandlung mehrerer gleichzeitiger Fehler einen deutlichen Vorberechnungsaufwand erfordert. Außerdem ist die Nutzung von Abkürzungen über Nicht-Spannbaumkanten bei diesen beiden Verfahren nicht möglich, was in höherem Stretch resultiert.

6.4 Funktionsweise von GFCP

Greedy Failure-Carrying Packets (GFCP) ist eine Rerouting-Strategie für spannbaumbasierte gierige Routing-Verfahren. Trifft ein Paket bei der Weiterleitung auf ein durch einen Knoten- oder Kantenausfall entstandenes lokales Minimum, so kodiert GFCP eine Beschreibung dieses Fehlers im Paket. In nachfolgenden Weiterleitungsentscheidungen wird das Paket entgegen dem Greedy-Prinzip unter Ausschluss der im Paket enthaltenen Kanten über einen eventuell längeren Pfad um Fehlerstellen herum in Richtung des Ziels weitergeleitet.

Greedy Failure-Carrying Packets benötigt dazu die folgenden Bausteine, die in den folgenden Abschnitten vorgestellt werden:

1. Zur Kodierung von Fehlern, die ein Paket auf seinem Weg antrifft, ist eine **Beschreibung ausgefallener Spannbaumkanten** notwendig. Wie bereits im Netzwerkmodell argumentiert, können während der Weiterleitung eines Pakets Knotenausfälle nicht von Kantenausfällen unterschieden werden, und daher genügt eine Beschreibung ausgefallener Kanten. Da die gierige Einbettung eines Spannbaums durch den Ausfall von Nicht-Spannbaumkanten nicht beeinträchtigt wird und dadurch keine lokalen Minima entstehen, sind für die Weiterleitung von Paketen nur Spannbaumkanten relevant und müssen beschrieben werden (siehe Abschnitt 6.4.1).
2. Anhand der lokalen Routing-Tabellen eines Knotens muss es in Kombination mit den in einem Paket enthaltenen Fehlerbeschreibungen möglich sein, eine **Gültigkeitsprüfung von Pfaden**, über die das Paket in Richtung des Ziels weitergeleitet werden soll, vorzunehmen. Ein Pfad ist dabei gültig, wenn er von keinem der bisher in einem Paket enthaltenen Fehler betroffen ist (siehe Abschnitt 6.4.2).
3. Die **GFCP Paketweiterleitung** muss entsprechend um diese Prüfung auf Gültigkeit erweitert werden und muss ungültige Pfade ausschließen (siehe Abschnitt 6.4.3).
4. Während es durch die Umleitung von Paketen an lokalen Minima möglich ist, dass Knoten mehrfach besucht werden, dürfen durch die GFCP Paketweiterleitung keine dauerhaften Schleifen entstehen. Dazu ist ein Beweis für die **Schleifenfreiheit** notwendig (siehe Abschnitt 6.4.4).

6.4.1 Beschreibung ausgefallener Spannbaumkanten

Eine Spannbaumkante stellt die kleinste Einheit eines Ausfalls dar. Sollte ein Knoten u ein Paket über eine ausgefallene Spannbaumkante weiterleiten wollen und stellt ein lokales Minimum fest, muss er eine Beschreibung für diese Kante im Paket kodieren und kann dabei auf die folgenden, lokal vorliegenden Informationen zurückgreifen. Diese Informationen

stammen aus drei Quellen: der gierigen Einbettung (GE), den lokal bekannten Fehlern (F) und der Spannbaumstruktur (B).

- Die eigene Adresse, die aus l virtuellen Koordinaten besteht (GE-1)
- Die im Paket enthaltene Zieladresse, die sich ebenfalls aus l virtuellen Koordinaten zusammensetzt (GE-2)
- Die Adressen der Nachbarknoten $\Gamma_G(u)$ und die Distanz zu diesen. (GE-3)
- Daraus ableitbar die eigene Distanz sowie die Distanz der Nachbarn zur Wurzel entlang des Spannbaums. (GE-4)
- Den Zustand der Kanten zu Nachbarknoten: Ein Knoten kennt den Zustand der direkt verbundenen Kanten und weiß damit, ob diese nutzbar sind oder nicht. (F-1)
- Für Nachbarknoten ist zudem bekannt, in welcher Beziehung diese zum betrachteten Knoten u stehen, ob sie also in einem Spannbaum Kind, Eltern, oder sogar Wurzelknoten sind. Damit ist für Knoten u klar, welche Spannbaumkanten auf welche seiner Kanten fallen beziehungsweise welche seiner Kanten nicht durch einen Spannbaum abgedeckt sind. (B-1)
- Entlang des Spannbaums existiert jeweils nur ein Pfad zwischen zwei Knoten. (B-2)

Zur **Beschreibung einer ausgefallenen Spannbaumkante** muss zunächst der betroffene Spannbaum T referenziert werden, was beispielsweise über einen eindeutigen Identifikator wie die Wurzel dieses Spannbaums möglich ist. Um die Lage einer Kante (u, v) innerhalb dieses Spannbaumes T zu bestimmen, stehen lokal die virtuellen Koordinaten der angrenzenden Knoten, also u^T und v^T, zur Verfügung. Das **Tripel** (T, u^T, v^T) genügt zur Beschreibung einer Spannbaumkante (u,v) und ermöglicht, wie in Abschnitt 6.4.2 beschrieben, die Gültigkeitsprüfung von Pfaden. Die für die Beschreibung notwendigen Informationen stehen, wie gefordert, lokal zur Verfügung (GE-1, GE-3, F-1).

Neben der Fähigkeit, eine Spannbaumkante zu beschreiben, ist hier noch wichtig, welche Spannbaumkanten in die Fehlermenge $\mathscr{F}(p)$ eines Pakets p aufgenommen werden. Das **Hinzufügen einer ausgefallenen Spannbaumkante zu** $\mathscr{F}(p)$ erfolgt genau dann, wenn diese nach dem Greedy-Prinzip zur Weiterleitung des Pakets genutzt worden wäre. Bei den hier angenommenen l Spannbaum-Ebenen werden der Fehlermenge $\mathscr{F}(p)$ zusätzlich Fehlerbeschreibungen für alle ebenfalls auf der ausgefallenen Kante liegenden Spannbäume hinzugefügt, in denen auch das Ziel des Pakets enthalten ist. Dies ist über die Adressen der an die Kante angrenzenden Knoten sowie der Adresse des Ziels anhand von lokal vorhandener Information festzustellen (GE-1, GE-2, GE-3). Die letzte Bedingung, dass auch das Ziel des Pakets enthalten sein muss, ist für Spannbäume, die den ganzen Graphen abdecken trivialerweise erfüllt. Ist dies jedoch, wie beispielsweise bei Practical Isometric Embedding (siehe Abschnitt 2.2.2.2), das später auch als Anwendungsbeispiel für GFCP verwendet wird, nicht der Fall, macht diese Einschränkung Sinn. Pakete können dort nur entlang von Spannbäumen weitergeleitet werden, in denen das Ziel auch tatsächlich enthalten ist und können konsequenterweise auch nur in diesen Spannbäumen auf lokale Minima treffen, die die Kodierung eines Fehlers erforderlich machen würden.

Die Fehlermenge $\mathscr{F}(p)$ eines Pakets p enthält also nach den obigen Auswahlkriterien eine Menge von Fehlerbeschreibungen in Form von Tripeln (T, u^T, v^T). Anhand der in der Fehlermenge eines Pakets enthaltenen Fehlerbeschreibungen muss im Folgenden überprüft werden,

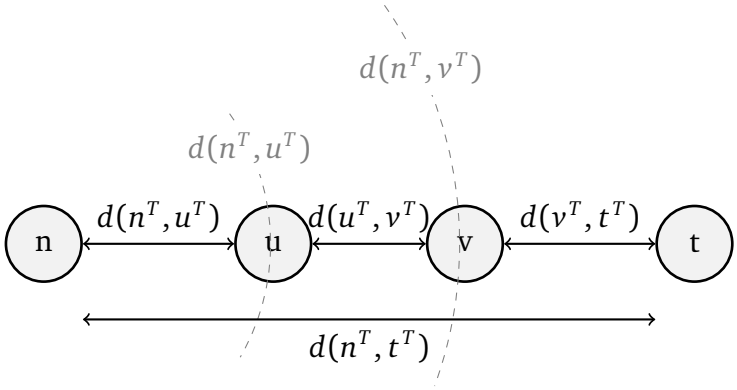


Abbildung 6.4: Veranschaulichung der Überprüfung, ob eine Kante entlang eines Spannbaums zwischen zwei Knoten liegt (Formulierung 1).

ob ein Pfad in Richtung des Ziels noch gültig ist. Diese Gültigkeitsprüfung beschreibt der folgende Abschnitt.

6.4.2 Gültigkeitsprüfung von Pfaden

Anhand der im letzten Abschnitt vorgestellten Fehlerbeschreibungen durch Tripel (T, u^T, v^T) soll nun überprüft werden, ob ein bestimmter Pfad gültig ist und also für die Paketweiterleitung genutzt werden kann. Dazu muss sichergestellt werden, dass keine der im Paket kodierten ausgefallenen Kanten auf diesem Pfad zum Ziel liegt. Da eine Fehlerbeschreibung immer nur genau einen Spannbaum betrifft, wird für eine einfachere Beschreibung der Fokus im Folgenden auf einen einzelnen Spannbaum T gelegt, der zunächst von genau einem Ausfall in Form der Kante (u, v) betroffen ist.

Es muss also überprüft werden, ob die Weiterleitung eines Pakets p mit enthaltener Fehlerbeschreibung (T, u^T, v^T) über einen Knoten n noch erfolgreich zum Zielknoten t des Pakets führen kann. Dazu gibt es zwei Formulierungen, die beide darauf beruhen, dass in einem Baum genau ein Pfad zwischen zwei Knoten existiert.

Die erste Formulierung nutzt die Tatsache aus, dass der einzige Pfad zwischen zwei Knoten n und t entlang des Baumes gleichzeitig auch der kürzeste darin ist. Wie in Abbildung 6.4 durch die grau gestrichelten Kreise angedeutet, kann es von Knoten n aus mehrere Kanten geben, deren angrenzende Knoten von n die Abstände $d(n^T, u^T)$ bzw. $d(n^T, v^T)$ haben. Ebenso können von Knoten t aus betrachtet mehrere Kanten durch die Abstände $d(t^T, u^T)$ bzw. $d(t^T, v^T)$ beschrieben werden. Da in einem Baum aber nur ein Pfad zwischen zwei Knoten existiert und dieser damit auch der kürzeste ist, muss die Länge des Weges von Knoten n nach t über die Kante $e = (u, v)$ der Länge dieses kürzesten Pfades $d(n^T, t^T)$ entsprechen, damit die Kante $e = (u, v)$ auf dem Pfad liegt.

Damit eine Kante $e = (u^T, v^T)$ in einem Spannbaum T also zwischen einem Knoten n^T und

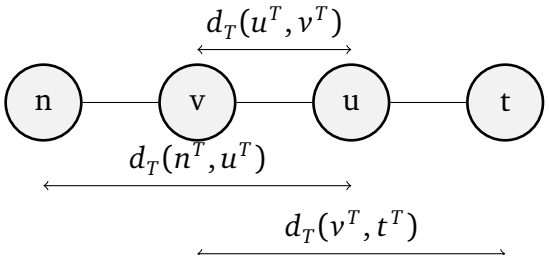


Abbildung 6.5: Reduktion des Berechnungsaufwandes durch alternativen Ausdruck für gedrehte Kantenorientierung.

einem Knoten t^T liegt, muss Gleichung 6.1 erfüllt sein.

$$\begin{aligned} & \text{onpath}(T, u^T, v^T, n^T, t^T) := \\ & \left[d_T(n^T, u^T) + d_T(u^T, v^T) + d_T(v^T, t^T) == d_T(n^T, t^T)\right] \\ \vee \quad & \left[d_T(n^T, v^T) + d_T(v^T, u^T) + d_T(u^T, t^T) == d_T(n^T, t^T)\right] \end{aligned} \tag{6.1}$$

Die erste Zeile von Gleichung 6.1 beschreibt dabei die in Abbildung 6.4 skizzierte Orientierung der Kante $n-(u-v)-t$ und die zweite Zeile den Fall, dass die Kante umgekehrt orientiert ist $n-(v-u)-t$.

Um die Anzahl der notwendigen Distanzberechnungen pro Überprüfung zu reduzieren, kann die zweite Zeile unter Wiederverwendung von Werten aus der ersten Zeile berechnet werden. Die zweite Zeile kann, wie in Gleichung 6.2 dargestellt, umformuliert werden, so dass keine erneuten Distanzberechnungen notwendig sind.

$$\begin{aligned} & \text{onpath}(T, u^T, v^T, n^T, t^T) := \\ & \left[d_T(n^T, u^T) + d_T(u^T, v^T) + d_T(v^T, t^T) == d_T(n^T, t^T)\right] \\ \vee \quad & \left[d_T(n^T, u^T) - d_T(u^T, v^T) + d_T(v^T, t^T) == d_T(n^T, t^T)\right] \end{aligned} \tag{6.2}$$

Dabei wird, wie Abbildung 6.5 verdeutlicht, die Kante (u, v) zunächst doppelt erfasst und muss daher einmal subtrahiert werden. Damit reduziert sich die Anzahl der ursprünglich notwendigen sechs Distanzberechnungen auf vier. Liegt die Kante (u, v) nicht auf dem Pfad, so sind in Gleichung 6.1 als auch in Gleichung 6.2 die linken Teile der beiden Vergleiche jeweils echt größer als $d_T(n, t)$, da dann eine Abweichung vom kürzesten Pfad vorliegt. Damit werden beide Vergleiche und folglich auch der Gesamtausdruck zu *false* ausgewertet.

Die zweite Formulierung zur Überprüfung, ob die Kante (u, v) auf einem Pfad im Spannbaum T liegt, nutzt ebenfalls die Eigenschaft aus, dass es nur einen Pfad entlang des Baumes zwischen zwei Knoten gibt. Betrachtet werden nun aber die zwei Teilbäume T_1 und T_2, in die ein Ausfall der Kante (u,v) den Spannbaum T teilen würde. Damit ein Knoten n, wie in Abbildung 6.6 skizziert, in Teilbaum T_1 liegt, muss gelten $d_T(n^T, u^T) < d_T(n^T, v^T)$. Dass ein Knoten y in Teilbaum T_2 liegt, muss entsprechend gelten $d_T(t^T, v^T) < d_T(t^T, u^T)$.

Damit eine Kante (u, v) also auf dem Pfad zwischen zwei Knoten n und t liegt, müssen diese

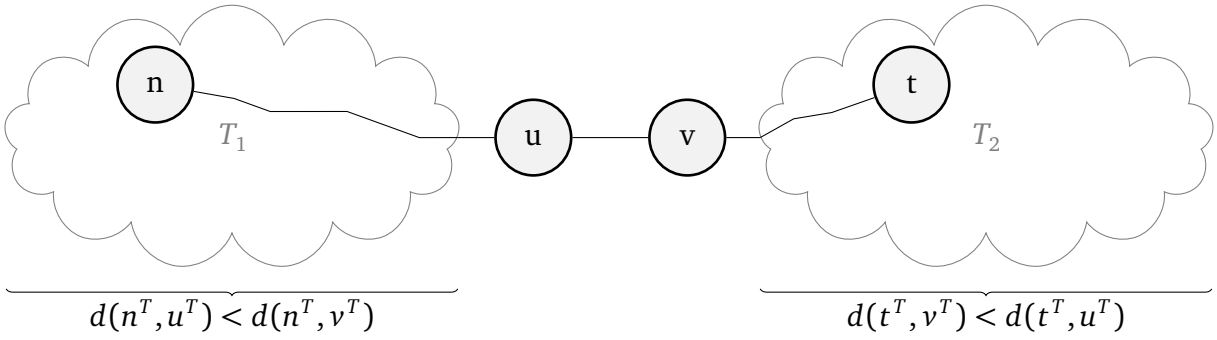


Abbildung 6.6: Veranschaulichung der Überprüfung, ob eine Kante entlang eines Spannbaums zwischen zwei Knoten liegt (Formulierung 2).

bezüglich der Kante (u, v) in verschiedenen Baumpartitionen liegen:

$$\begin{aligned} &\text{onpath}(T, u^T, v^T, n^T, t^T) := \\ &\left[d_T(n^T, u^T) < d_T(n^T, v^T) \wedge d_T(t^T, v^T) < d_T(t^T, u^T)\right] \\ \vee\ &\left[d_T(n^T, u^T) > d_T(n^T, v^T) \wedge d_T(t^T, v^T) > d_T(t^T, u^T)\right] \end{aligned} \tag{6.3}$$

Ob der Pfad entlang eines Spannbaums T über einen Nachbarn n in Richtung des Zielknotens t in Gegenwart einer ausgefallenen Kante $e = (u, v)$ gültig ist, kann dann anhand von Gleichung 6.4 überprüft werden:

$$\text{valid}(T, u^T, v^T, n^T, t^T) := \neg\text{onpath}(T, u^T, v^T, n^T, t^T) \tag{6.4}$$

Dabei kann die sowohl die Definition von *onpath* von Gleichung 6.2 als auch von Gleichung 6.3 verwendet werden.

Diese Überprüfung kann, wie eingangs auch gefordert, anhand der lokal vorhandenen Informationen vorgenommen werden. Der Spannbaum T, die Beschreibung der ausgefallenen Kante (u^T, v^T) und das Ziel t^T sind aus dem Paketkopf bekannt. Die Adresse des Nachbarknotens n und damit dessen virtuelle Koordinate in T ist Teil der Routingtabellen des sendenden Knotens s.

Damit erhält man die folgende **Gleichung zur Gültigkeitsprüfung von Pfaden** in Gegenwart einer Fehlermenge $\mathscr{F}(p)$. Ein Packet p kann dann über einen Knoten n entlang eines Baumes T weitergeleitet werden, wenn keine der enthaltenen Fehlerbeschreibungen den Pfad zwischen den Knoten n und t ungültig macht:

$$\text{valid}^T(n, t, \mathscr{F}(p)) := \bigwedge_{\substack{(T, u^T, v^T) \\ \in \mathscr{F}(p)}} valid(T, u^T, v^T, n^T, t^T)$$

Diese Gültigkeitsprüfung wird nun im folgenden Abschnitt in die Paketweiterleitung von GFCP integriert.

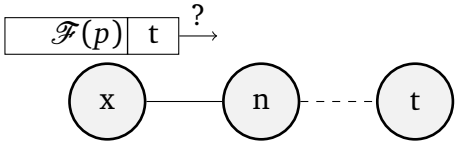


Abbildung 6.7: Knoten x muss bei der GFCP Paketweiterleitung überprüfen, ob ein Knoten n zur Weiterleitung verwendet werden kann.

6.4.3 GFCP Paketweiterleitung

Nachdem nun ausgefallene Spannbaumkanten beschrieben werden können und für eine ausgefallene Kante anhand Gleichung 6.4 überprüft werden kann, ob diese auf dem Pfad zwischen zwei Knoten liegt, verbleibt die Formulierung der Weiterleitungsfunktion für GFCP. Dazu betrachten wir nun ein Szenario mit l Baumebenen, die nicht notwendigerweise alle den kompletten Graphen umfassen.

Zur Weiterleitung eines Pakets p über einen Knoten n stehen damit zunächst all die Spannbäume $T_{n,t}$ zur Verfügung, in denen sowohl Knoten n als auch Knoten t enthalten sind. Durch bereits im Paket enthaltene Fehlerbeschreibungen $\mathscr{F}(p)$ (siehe Abbildung 6.7) kann die Menge τ der für die Weiterleitung verwendbaren Spannbäume durch den Ausschluss von Bäumen ohne einen gültigen Pfad zum Ziel auf die folgenden Spannbäume τ^* eingeschränkt werden:

$$\begin{aligned}&\tau^*(n,t,\mathscr{F}(p)) = \\ &\{T \in \tau \mid T \text{ ohne Fehler in } \mathscr{F}(p) \vee \text{valid}^T(n,t,\mathscr{F}(p))\}\end{aligned} \tag{6.5}$$

Damit umfasst $\tau^*(n,t,\mathscr{F}(p))$ genau die Spannbäume, für die bisher keine Fehlerbeschreibung vorliegt oder die über einen gültigen Pfad in Richtung des Ziels verfügen.

Gleichzeitig wird die Distanzberechnung für einen Weiterleitungsknoten n in Richtung von Knoten t in Gegenwart der Fehlermenge $\mathscr{F}(p)$ auf diese tatsächlich verwendbaren Spannbäume τ^* eingeschränkt:

$$d(n,t,\mathscr{F}(p)) = \min_{T \in \tau^*(n,t,\mathscr{F}(p))} \left\{ d_T(n^T, t^T) \right\} \tag{6.6}$$

Die Distanzfunktion aus Gleichung 6.6 liefert damit aus allen verbleibenden Pfaden den kürzesten als Ergebnis. Ist τ^* die leere Menge, so ist für eine einfachere Notation $d(n,t,\mathscr{F}(p)) = \infty$. Setzt man nun Gleichung 6.6 in die Bestimmung des Weiterleitungsknotens nach dem Greedy-Prinzip ein, so erhält man zur Bestimmung des Weiterleitungsknotens n für *Greedy Failure Carrying Packets* die folgende Gleichung:

$$n := \arg\min_{v \in \Gamma(G,u)} \{ d(u,v) + d(v,t,\mathscr{F}(p)) \} \tag{6.7}$$

Ist dabei $\mathscr{F}(p)$ leer, und hat das Paket damit bisher keine ausgefallene Kante angetroffen, erfolgt die Paketweiterleitung mit GFCP wie bisher nach dem Greedy-Prinzip. Nur für den Fall, dass im Paketkopf ausgefallene Kanten enthalten sind, weicht GFCP von der Bestimmung des Weiterleitungsknotens nach dem Greedy-Prinzip ab, um bereits als ausgefallen bekannte Wege zu vermeiden. Führt dies dazu, dass keiner der verfügbaren Spannbäume einen gültigen Weg in Richtung Ziel bietet, verwirft GFCP das entsprechende Paket.

Algorithmus 4 beschreibt die Weiterleitung von Paketen durch GFCP anhand der zuvor eingeführten Funktionen. Die Paketweiterleitung teilt sich damit grob in drei Teile, die im Folgenden beschrieben werden:

1. Die **Distanzbestimmung über alle Nachbarn**, wie sie bei Weiterleitung nach dem Greedy-Prinzip regulär erfolgt.
2. Den **Ausschluss ungültiger Pfade** durch Greedy Failure-Carrying Packets
3. Das **Verwerfen oder Weiterleiten des Pakets** abhängig davon, ob noch ein Weiterleitungsknoten zur Verfügung steht.

6.4.3.1 Distanzbestimmung über alle Nachbarn

Zur Bestimmung der bestmöglichen Distanz zum Ziel wird zunächst wie bei der Paketweiterleitung nach dem Greedy-Prinzip die Distanz zu diesem über alle Nachbarn und zur Verfügung stehenden Spannbäume vorgenommen. Statt jedoch nur das Optimum zu speichern, werden die Ergebnisse der Distanzbestimmung in dem MinHeap **distanceMap** mit der Distanz als Schlüssel gespeichert (Zeile 8). Als Wert wird für jeden Schlüssel zusätzlich ein Paar bestehend aus dem zugehörigen Weiterleitungsknoten n und dem Spannbaum T gespeichert.

6.4.3.2 Ausschluss ungültiger Pfade

Anschließend werden aufsteigend von der kürzesten Distanz Einträge aus distanceMap entfernt und für diese überprüft, ob die Kante zu dem darin gespeicherten Nachbarknoten (*entry.n*) ausgefallen ist. Ist dies der Fall, so werden der Fehlermenge *p.failedLinks* des Pakets p die notwendigen Fehlerbeschreibungen für die Kante zu Knoten *entry.n* hinzugefügt (Zeile 15) und der nächste Eintrag von distanceMap zur Bearbeitung entfernt.

Ist die Kante zum bisher besten Weiterleitungsknoten *entry.n* nicht ausgefallen, so wird für jede im Paket enthaltene Fehlerbeschreibung, die den Baum T des aktuell untersuchten Weiterleitungsknotens betrifft (Zeile 21), geprüft, ob damit immer noch eine erfolgreiche Weiterleitung zum Zielknoten t möglich ist. Verhindert dabei ein Fehler die erfolgreiche Weiterleitung, kann die Überprüfung mit dem nächsten Eintrag aus *distanceMap* fortgesetzt werden. Verläuft die Prüfung für alle im Paket enthaltenen Fehlerbeschreibungen jedoch erfolgreich, so wurde der optimale, nicht durch einen Fehler betroffene Weiterleitungsknoten gefunden und der bisher undefinierte *nexthop* wird auf den optimalen Weiterleitungsknoten gesetzt (Zeile 28). Steht kein gültiger Pfad zur Verfügung, so bleibt *nexthop* undefiniert.

6.4.3.3 Verwerfen oder Weiterleiten des Pakets

Wurde kein Nachbar gefunden, über den das Paket erfolgreich zum Ziel zugestellt werden kann, wird das Paket verworfen (Zeile 34). Ansonsten wird es über den bestimmten Nachbarn weitergeleitet (Zeile 36).

Algorithmus 4 Paketweiterleitung Greedy Failure-Carrying Packets

```
1:  procedure FORWARDGFCP(p)                          ▷ Weiterleitung des Pakets p
2:      distanceMap = ∅                               ▷ Min-Heap mit Distanz als Schlüssel
3:                                                    ▷ Distanz → (nexthop n, Baum-Identifikator T)
4:      nexthop = undefinedNode
5:
6:      for n ∈ Γ_G(x) do                             } Distanzbestimmung
7:          for T ∈ T_{n,t} do                        } über alle Nachbarn.
8:              distanceMap.insert( d_T(n^T, t^T), n, T)
9:          end for
10:     end for
11:
12:     while distanceMap != ∅ do                     } Ausschluss
13:         entry = distanceMap.getMin()              } ungültiger Pfade.
14:         if localLinkFailed(entry.n) then
15:             insert(p.failedLinks, entry.n)
16:             continue
17:         end if
18:
19:         foundValidNexthop = true
20:
21:         for f = (u^T, v^T) ∈ {p.failedLinks ∩ entry.T} do
22:             if ¬valid(entry.T, u^T, v^T, entry.n^T, t^T ) then
23:                 foundValidNexthop = false
24:                 break
25:             end if
26:         end for
27:         if foundValidNexthop then
28:             nexthop=entry.n
29:             break
30:         end if
31:     end while
32:
33:     if nexthop == undefinedNode then              } Verwerfen oder
34:         discard(p)                                } Weiterleiten
35:     else                                          } des Pakets.
36:         forward(p, nexthop)
37:     end if
38: end procedure
```

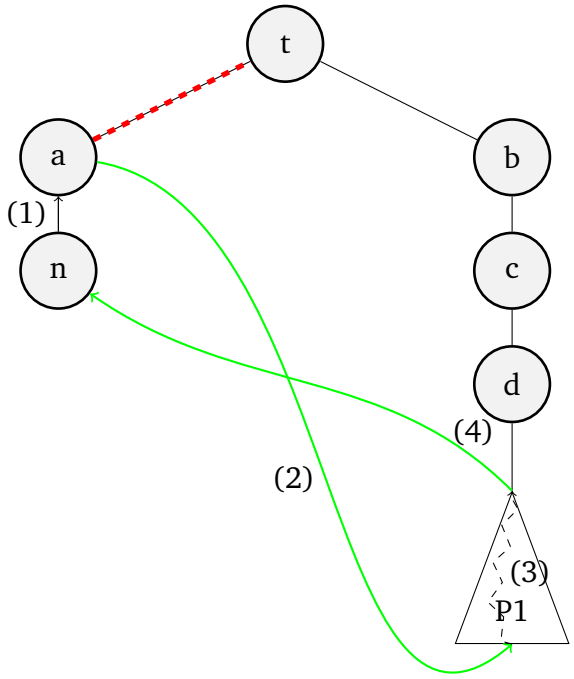


Abbildung 6.8: Beispiel das Auftreten einer Schleife, wenn die Paketweiterleitung nicht strikt gierig erfolgt.

6.4.4 Schleifenfreiheit

Die verwendeten gierigen Einbettungen von Spannbäumen sind in Abwesenheit von Kanten- oder Knotenausfällen schleifenfrei und die Weiterleitung nach dem Greedy-Prinzip erlaubt die Zustellung aller Pakete.

Wird die gierige Einbettung durch Ausfälle zerstört, so ist in der Regel die Erreichbarkeit der Knoten untereinander eingeschränkt und Pakete, die in lokale Minima geraten, werden bei gieriger Weiterleitung verworfen. Für Rerouting-Verfahren wie GFCP oder GP ist es an dieser Stelle erforderlich, Pakete zunächst entgegen der rein gierigen Weiterleitungsstrategie wieder weiter vom Ziel wegzuführen, um die Umgehung der Fehlerstelle zu ermöglichen.

Abbildung 6.8 stellt beispielhaft die Entstehung von Schleifen dar, wenn ohne weitere Maßnahmen an lokalen Minima auch eine nicht-gierige Paketweiterleitung möglich ist. In diesem Szenario sendet Knoten n ein Paket an den Zielknoten t und verwendet dazu nach dem Greedy-Prinzip Knoten a (1) zur Weiterleitung. Die Kante zwischen den Knoten a und t, über die das Paket eigentlich gierig weitergeleitet werden müsste, ist ausgefallen. Daher wird das Paket bei Weiterleitung nach dem Greedy-Prinzip verworfen werden, da kein Fortschritt in Richtung des Ziels mehr möglich ist.

Lässt man aber zu, dass das Paket auch wieder schlechtere Pfade nutzen kann, könnte dieses wie in der Abbildung skizziert beispielsweise den Teilbaum $P1$ betreten (2), über den

der Zielknoten t zu höheren Kosten erreichbar wäre. Das Paket wird nun in Richtung von Knoten t weitergeleitet (3). Da aber eine vermeintliche Abkürzung über Knoten n existiert (4), über die das Paket scheinbar zwei Übertragungsabschnitte (n-a-t) benötigt statt entlang des Baumes drei Übertragungsabschnitte (d-c-b-t), gelangt das Paket wieder zu Knoten n. Dadurch entsteht die Schleife (1)-(2)-(3)-(4).

Im den folgenden Abschnitten wird gezeigt, wie GFCP diesem Problem der Schleifenentstehung begegnet und dass dort keine Schleifen auftreten können. Dabei wird für eine einfachere Argumentation zunächst ein einzelner Spannbaum ohne Nutzung von Abkürzungskanten betrachtet. Anschließend wird die Nutzung von Abkürzungskanten zugelassen und schließlich die Erweiterung auf mehrere Spannbäume vorgenommen.

6.4.4.1 Einzelner Spannbaum ohne Nutzung von Abkürzungen

Steht nur ein einzelner Spannbaum T zur Weiterleitung von Paketen zur Verfügung und ermöglicht die zugrundeliegende gierige Einbettung keine Weiterleitung von Paketen entlang von Nicht-Spannbaumkanten, so ist bei Verwendung von GFCP keine Entstehung von Schleifen möglich.

Der Ausfall einer zum Erreichen des Zielknotens t notwendigerweise zu überquerenden Kante (u, v) partitioniert den Spannbaum T in zwei Teilbäume P_1 (auf der Seite von Knoten u) und P_2 (auf der Seite von Knoten v). Diese Situation stellt Abbildung 6.9 beispielhaft dar.

Sei ein Paket p nun o. B. d. A. auf Knoten u im Teilbaum P_1 und der Zielknoten t im Teilbaum P_2. Dann wird anhand der GFCP Weiterleitungsentscheidung (siehe Algorithmus 4) zunächst Knoten v als Weiterleitungsknoten bestimmt und, da die entsprechende Kante ausgefallen ist, dem Paket p eine Fehlerbeschreibung der Kante (u, v) hinzugefügt. Danach wird für alle weiteren Nachbarn von Knoten u überprüft, ob eine Weiterleitung über diese in Gegenwart der Fehlerbeschreibung der Kante (u, v) zum Ziel führen kann. Da nach Voraussetzung (keine Nutzung von Abkürzungen) nur die eine Verbindung (u,v) zwischen den Partitionen P_1 und P_2 existiert, ergibt sich die in Abbildung 6.9 dargestellte Situation. Für jeden der in Partition P_1 liegenden Nachbarn n_i von Knoten u ist damit die Überprüfung, ob die Kante (u, v) auf dem Weg zu Knoten t liegt (siehe Gleichung 6.3) wahr, da in diesem Fall die erste Bedingung

$$d_T(n_i^T, u^T) < d_T(n_i^T, v^T) \wedge d_T(t^T, u^T) < d_T(t^T, v^T)$$

von onpath (siehe Gleichung 6.3) immer wahr ist. Dadurch werden alle potentiellen Weiterleitungsknoten n_i durch die GFCP Weiterleitungsfunktion ignoriert und das Paket schließlich durch Knoten u verworfen.

6.4.4.2 Einzelner Spannbaum unter Nutzung von Abkürzungen

Besteht nun zusätzlich zu dem vorhergehenden Szenario die Möglichkeit auch Nicht-Spannbaumkanten für die Paketweiterleitung zu nutzen, ergibt sich die in Abbildung 6.10 dargestellte Situation. Dabei soll Knoten u wie zuvor ein Paket an Knoten t weiterleiten,

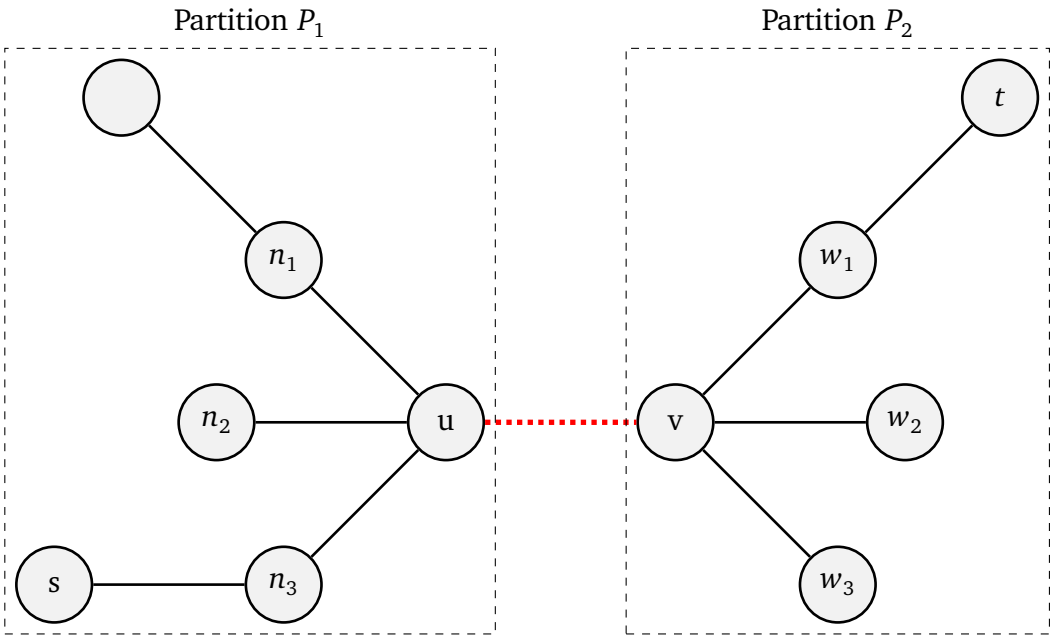


Abbildung 6.9: Einzelner Spannbaum ohne Nutzung von Abkürzungen

das bereits die Fehlerbeschreibung der Kante (u, v) enthält. Mit derselben Begründung wie zuvor scheiden daher alle in P_1 liegenden Nachbarn n_i von Knoten u aus.

Damit verbleibt für eine mögliche Weiterleitung nur die grün dargestellte Nicht-Spannbaumkante. Die Kosten einer Weiterleitung über diese Nicht-Spannbaumkante sind dabei höher als der bisher gierig verfolgte Pfad mit Weiterleitung über v. Anhand des GFCP Weiterleitungsalgorithmus wird das Paket aber über die Nicht-Spannbaumkante in die Partition P_2 weitergeleitet.

Nun, da sich das Paket in Partition P_2 befindet, trägt das Vorhandensein der Fehlerbeschreibung (u, v) dafür Sorge, dass dieses die Partition P_1 nie wieder betreten kann. Befindet sich das Paket in der gleichen Partition P_2 wie der Zielknoten, wird für alle Knoten n_i aus Partition P_1 die Bedingung

$$d_T(n_i^T, u^T) > d_T(n_i^T, v^T) \wedge d_T(t^T, u^T) > d_T(t^T, v^T)$$

von onpath (siehe Gleichung 6.3) immer wahr und damit in der GFCP Weiterleitungsfunktion eine Weiterleitung zu diesen Knoten ausgeschlossen.

Unabhängig vom Auftreten weiterer Fehler wird also ein einmal ausgeschlossener Teilbaum weder über eine Spannbaumkante noch über Abkürzungen wieder betreten. Durch das Hinzufügen weiterer Fehlerbeschreibungen werden die Weiterleitungsmöglichkeiten für ein Paket entweder immer weiter eingeschränkt, bis kein gültiger Weiterleitungsknoten mehr existiert und das Paket verworfen wird oder dieses schließlich seinen Zielknoten erreicht.

6.4.4.3 Mehrere Spannbäume zur Paketweiterleitung

Stehen für die Weiterleitung von Paketen mehrere Spannbäume zur Verfügung, so ist es möglich, dass ein Paket über einen Baum in einen durch Fehlerbeschreibungen ausgeschlossenen

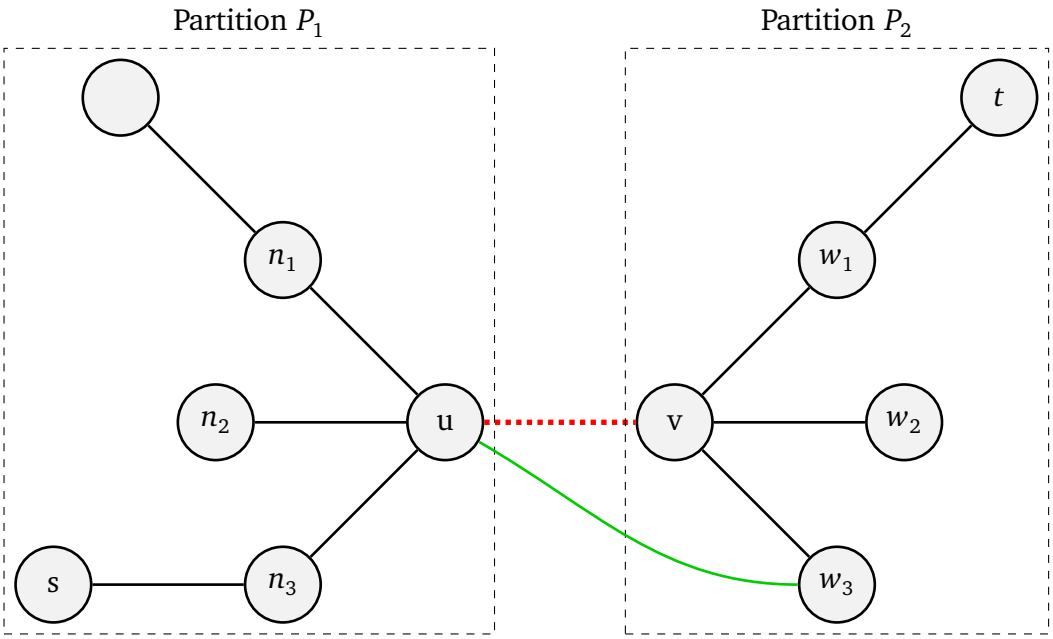


Abbildung 6.10: Einzelner Spannbaum unter Nutzung von Abkürzungen

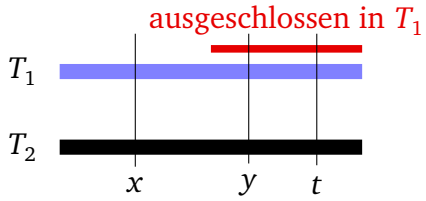


Abbildung 6.11: Mehrere Spannbäume zur Paketweiterleitung

Teilbaum eines anderen Baumes weitergeleitet wird, wie schematisch in Abbildung 6.11 dargestellt.

Dies ist jedoch kein Problem, da jeder Baum für sich mit der GFCP Paketweiterleitung schleifenfrei bleibt. Erreicht ein Paket durch die Weiterleitung von Knoten x zu Knoten y einen in Spannbaum T_1 ausgeschlossenen Teilbaum (rot markiert), so wird das Paket bei nachfolgenden Weiterleitungen im Spannbaum T_1 nur dann weitergeleitet, wenn dieses dadurch wieder in einen gültigen Bereich des Spannbaums kommt. Dies ist beispielsweise durch die Nutzung von Abkürzungskanten, wie im letzten Abschnitt erläutert, möglich.

Solange alle möglichen Weiterleitungsknoten innerhalb des ausgeschlossenen Teilbaums liegen, erfolgt keine Weiterleitung entlang von T_1. Ist T_2 wie im skizzierten Beispiel nicht von Fehlern betroffen, so kann das Paket schließlich über diesen Spannbaum zugestellt werden.

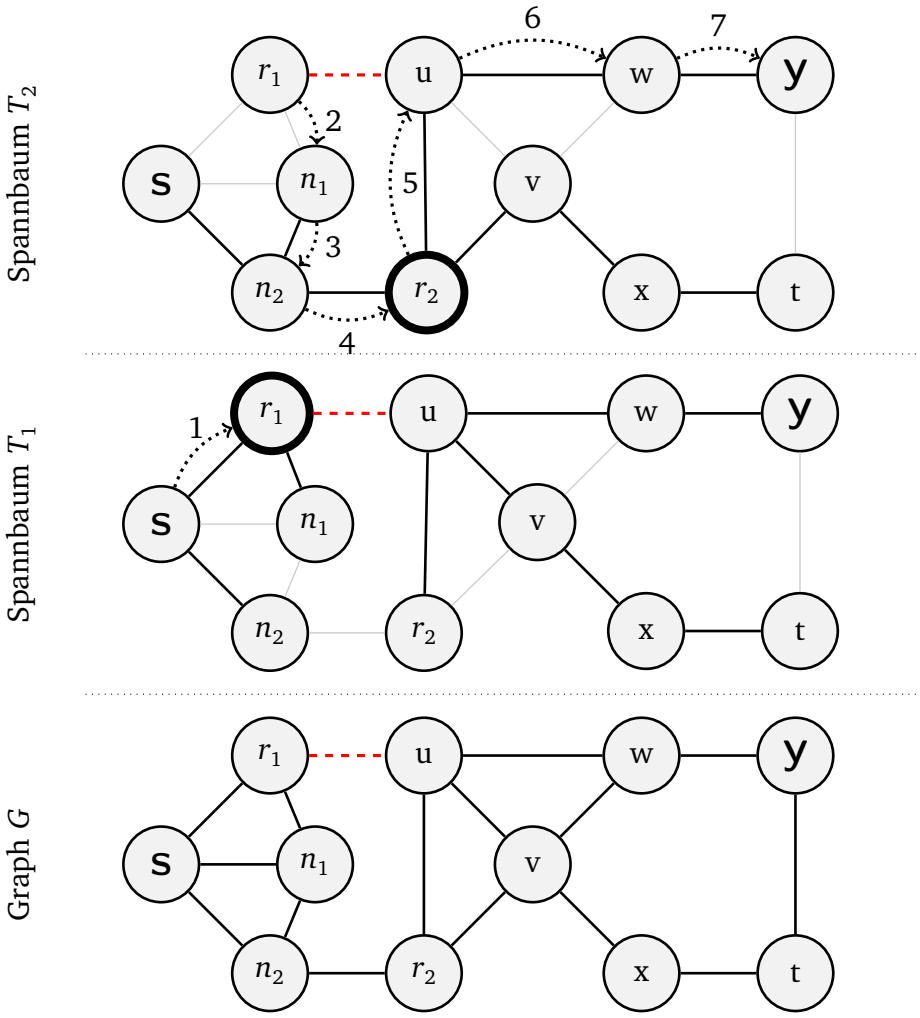


Abbildung 6.12: GFCP ermöglicht im Gegensatz zu Gravity-Pressure Forwarding frühes Verwerfen von Paketen.

6.4.5 Diskussion

Die Paketweiterleitung durch GFCP kann dazu führen, dass ein Paket verworfen wird, obwohl zwei Knoten in der zugrundeliegenden Topologie immer noch verbunden sind. In der Evaluierung wird gezeigt, inwiefern sich dies auf die Zustellrate auswirkt.

Im folgenden Beispiel wird die Paketweiterleitung von GFCP bezüglich des Verwerfens von Paketen der von Gravity-Pressure Routing gegenübergestellt. Dazu wird das zuvor in Abschnitt 6.3.2.2 für Gravity-Pressure Routing verwendete Beispiel erneut aufgegriffen (siehe Abbildung 6.12). Es wird ebenfalls die Weiterleitung eines Pakets von Knoten s an Knoten y, diesmal aber mit GFCP, betrachtet.

Zunächst wird der Fall untersucht, dass nur der Spannbaum T_1 zur Verfügung steht. Das Paket wird von Knoten s zunächst gierig an Knoten r_1 weitergeleitet. Dort wird festgestellt, dass die Kante zu Knoten u, die die beste Distanz zum Ziel böte, ausgefallen ist. Daher wird diese Kante nun dem Paketkopf hinzugefügt und bei der folgenden Bestimmung eines alternativen

Weiterleitungsknotens nach GFCP ausgeschlossen. Damit verbleiben die Nachbarknoten s und n_1, wobei eine Weiterleitung über beide jeweils in Kosten von 5 Hops resultiert. Werde nun o. B. d. A. zuerst Knoten n_1 betrachtet. Gemäß Algorithmus 4 wird nun geprüft, ob eine Weiterleitung über Knoten n_1 einen gültigen Pfad zu Knoten y bei ausgefallener Kante $r_1 - u$ bietet:

$$\begin{aligned} & \text{valid}(T_1, r_1^{T_1}, u^{T_1}, n_1^{T_1}, y^{T_1}) = \\ & [d_T(n_1^{T_1}, r_1^{T_1}) < d_T(n_1^{T_1}, u^{T_1}) \wedge d_T(r_1^{T_1}, y^{T_1}) < d_T(u^{T_1}, y^{T_1})] \\ \vee \quad & [d_T(n_1^{T_1}, r_1^{T_1}) > d_T(n_1^{T_1}, u^{T_1}) \wedge d_T(r_1^{T_1}, y^{T_1}) > d_T(u^{T_1}, y^{T_1})] \\ = \quad & (1 < 2 \wedge 3 < 2) \vee (1 > 2 \wedge 3 > 2) \\ = \quad & \textit{false} \vee \textit{false} \\ = \quad & \textit{false} \end{aligned} \tag{6.8}$$

Damit führt eine Weiterleitung über Knoten n_1 entlang des Spannbaums T_1 über die Kante $r_1 - u$ und der Pfad ist damit ungültig. Knoten n_1 wird daher bei nachfolgenden Weiterleitungsentscheidungen nicht mehr berücksichtigt. Ebenso führt eine Überprüfung des Nachbarknotens s zu dem Resultat, dass eine Weiterleitung über diesen die ausgefallene Kante nutzen würde. Damit verbleibt für Knoten r_1 kein Nachbar mehr, der das Paket unter Ausschluss der ausgefallenen Kante in Richtung des Zielknotens y weiterleiten kann, und das Paket wird daher bei Knoten r_1 verworfen.

Steht für die Paketweiterleitung neben T_1 zusätzlich der Spannbaum T_2 bereit, so wird bei r_1 wieder die Kantenbeschreibung von (u,v) der Menge ausgefallener Kanten im Paketkopf hinzugefügt. Da der Zielknoten y sowohl in Spannbaum T_1 als auch in Spannbaum T_2 enthalten ist, werden dem Paket zwei Fehlerbeschreibungen $(T_1, r_1^{T_1}, u^{T_1})$ und $(T_2, r_1^{T_2}, u^{T_2})$ hinzugefügt. Bei der Weiterleitungsentscheidung, z. B. über Knoten n_1, wird nun zunächst wieder überprüft, ob eine erfolgreiche Weiterleitung über Spannbaum T_1 möglich ist. Da dies, wie zuvor beschrieben, nicht der Fall ist, wird nun die Weiterleitung über Knoten n_1 entlang von Spannbaum T_2 überprüft:

$$\begin{aligned} & \text{valid}(T_2, r_1^{T_2}, u^{T_2}, n_1^{T_2}, y^{T_2}) = \\ & [d_T(n_1^{T_2}, r_1^{T_2}) < d_T(n_1^{T_2}, u^{T_2}) \wedge d_T(r_1^{T_2}, y^{T_2}) < d_T(u^{T_2}, y^{T_2})] \\ \vee \quad & [d_T(n_1^{T_2}, r_1^{T_2}) > d_T(n_1^{T_2}, u^{T_2}) \wedge d_T(r_1^{T_2}, y^{T_2}) > d_T(u^{T_2}, y^{T_2})] \\ = \quad & (4 < 3 \wedge 3 < 2) \vee (4 > 3 \wedge 3 > 2) \\ = \quad & \textit{false} \vee \textit{true} \\ = \quad & \textit{true} \end{aligned} \tag{6.9}$$

Die Weiterleitung des Pakets über T_2 zum Ziel ist also möglich, ohne die ausgefallene Kante überqueren zu müssen und damit ist eine Weiterleitung über n_1 möglich. Wie in Abbildung 6.12 dargestellt, kann das Paket entlang von Spannbaum T_2 erfolgreich zugestellt werden. Ab Knoten r_2 wäre zudem Spannbaum T_1 auch wieder für eine Weiterleitung nutzbar, da sich das Paket dann bezüglich der ausgefallenen Kante im gleichen Teilbaum wie der Zielknoten y befindet.

Das vorangegangene Beispiel macht zwei Unterschiede zu Gravity-Pressure Routing deutlich: GFCP verwendet nur Pfade, über die ein Paket nach aktuellem Wissen, das sich aus den Routing-Tabellen eines Knotens und den in einem Paket enthaltenen Fehlerbeschreibungen

zusammensetzt, auch tatsächlich zugestellt werden kann. Da durch die gierig eingebetteten Spannbäume nur eine sehr eingeschränkte Sicht auf die Topologie zur Verfügung steht, kann es dadurch vorkommen, dass ein Paket nicht zugestellt werden kann, obwohl der zugrundeliegende Graph noch verbunden ist. Dieser Fall trat im obigen Beispiel auf, als nur der Spannbaum T_1 zur Verfügung stand. Dies ermöglicht aber auf der positiven Seite auch, Pakete vor Ablauf der TTL zu verwerfen, wenn kein gültiger Pfad mehr zur Verfügung steht. Bei Gravity-Pressure Routing kann der Fall auftreten, dass Pakete bis zum Ablauf der TTL im Netz umherirren ohne jemals das Ziel zu erreichen, was die Netzbelastung unnötig erhöht. Diese Abwägung zwischen der Zustellrate und der Belastung eines Netzes wird in der Evaluierung im Detail untersucht.

6.5 Anwendung auf Practical Isometric Embedding

Für die spätere Evaluierung von GFCP wird dieses nun auf das bereits in Abschnitt 2.2.2.2 vorgestellte Practical Isometric Routing Protocol angewendet. In PIE ist jeder Knoten Mitglied von genau l Spannbäumen unterschiedlicher Lokalität und die Adresse eines Knotens x ist entsprechend $x^{T_0}||x^{T_1}||...||x^{T_l}$. Der Hauptspannbaum auf Ebene 0, der alle Knoten des Graphen umfasst, stellt dabei die gegenseitige Erreichbarkeit sicher, während zusätzliche Baum-Ebenen für eine gewisse Pfaddiversität sorgen. Durch die auf höheren Ebenen zunehmende Lokalität, werden mit Annäherung des Pakets an das Ziel mit hoher Wahrscheinlichkeit immer mehr Spannbäume für die Weiterleitung verfügbar, während weit vom Ziel entfernt eventuell nur der Hauptspannbaum für die Weiterleitung zur Verfügung steht. Abbildung 6.13 stellt ein PIE Paket bei einer unoptimierten Anwendung von GFCP mit Fokus auf die für die Paketweiterleitung relevanten Informationen dar. Wie im PIE-Paketkopf sind hier die PIE-Adressen des Empfängers sowie des Senders enthalten. Jede dieser Adressen setzt sich aus virtuellen Koordinaten aus je einem Spannbaum auf jeder der l Spannbaumebenen zusammen. Da es pro Spannbaum-Ebene mehrere Spannbäume geben kann, wird zur Unterscheidung der Bäume zusätzlich zu den virtuellen Koordinaten der Identifikator der Wurzel des jeweiligen Baums mitgeführt. Da die Länge der virtuellen Koordinaten variabel ist, geht diesen jeweils ein Längenfeld voraus. Das Feld *minimale gierige Distanz* enthält die bisher vom Paket erreichte minimale Distanz zum Ziel und ermöglicht es, unterwegs festzustellen, ob ein Paket gerade gierig weitergeleitet werden kann oder nicht.

Die GFCP-Erweiterung besteht aus einem festen, kompakten GFCP Control Header und Fehlerbeschreibungen, die einem Paket auf seinem Weg dynamisch hinzugefügt werden können. Das *F-Bit* ist standardmäßig nicht gesetzt (0) und wird gesetzt, wenn durch Weiterleitung nach dem Greedy-Prinzip kein Fortschritt gegenüber der im PIE-Paketkopf enthaltenen minimalen gierigen Distanz erzielt wird. Diese Situation kann nur auftreten, wenn mindestens eine an den weiterleitenden Knoten angrenzende Kante, die diesen Fortschritt ermöglicht hätte, ausgefallen ist. Beim Eintrag einer Fehlerbeschreibung in den Paketkopf wird zunächst die *Fehleranzahl* im GFCP Control Header, die mit anfangs mit 0 initialisiert ist, inkrementiert und eine Fehlerbeschreibung der ausgefallenen Kante hinzugefügt. Diese besteht aus der *Baum-ID* als eindeutigem Identifikator für den betroffenen Baum und den virtuellen Koordi-

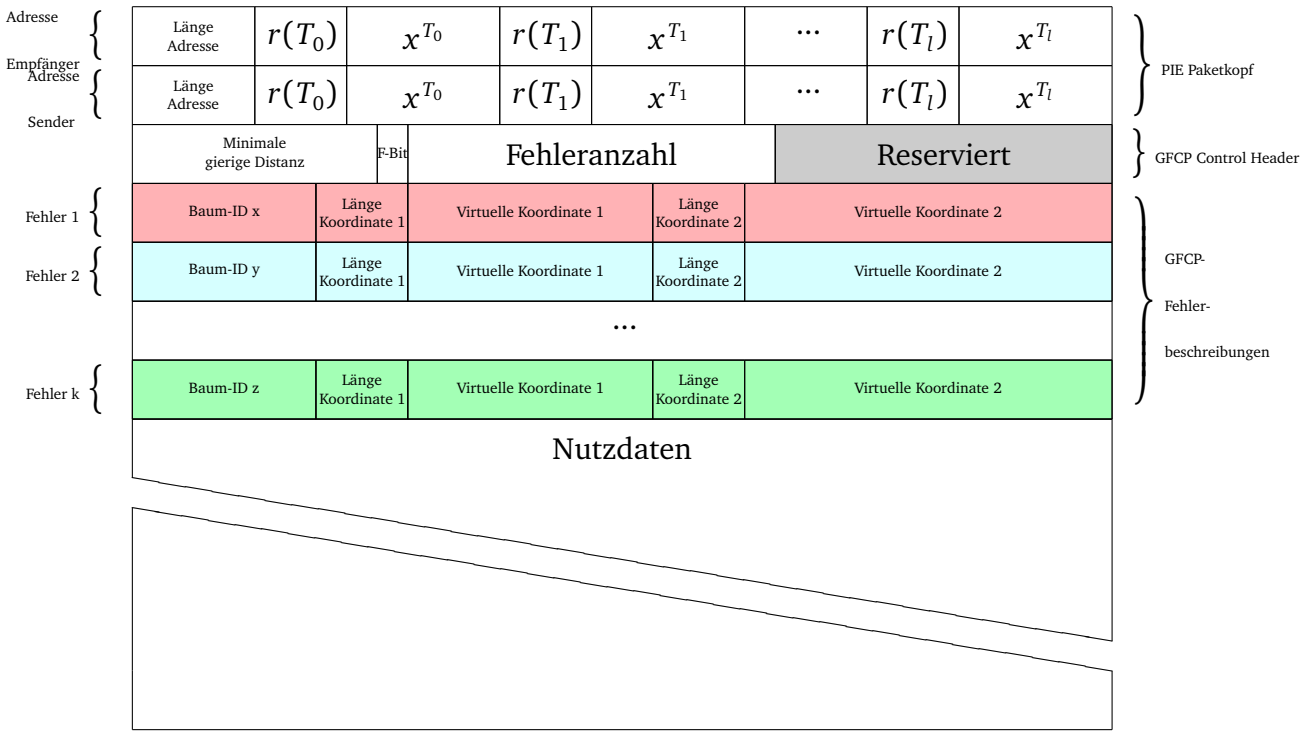


Abbildung 6.13: Unoptimiertes Paketformat für PIE mit GFCP-Erweiterung

naten der angrenzenden Knoten, die aufgrund der variablen Länge ebenfalls ein Längenfeld benötigen.

In Gegenwart vieler ausgefallener Knoten oder Kanten ist es möglich, dass die Beschreibung der ausgefallenen Kanten den restlichen Paketkopf dominiert. Es ist daher für einen praktischen Einsatz von GFCP wichtig, den für die Fehlerbeschreibungen notwendigen Platz zu reduzieren. Für das hier zum Einsatz kommende Practical Isometric Embedding wurden daher die im Folgenden vorgestellten Verfahren zur Reduktion der erforderlichen Fehlerbeschreibungen und für eine effizientere Kodierung von Fehlerbeschreibungen entwickelt.

6.5.1 Reduktion der Anzahl von Fehlerbeschreibungen

Laut bisheriger Beschreibung werden in den Paketkopf die Fehlerbeschreibungen in ein Paket eingetragen, die sich auf einen Spannbaum beziehen, in dem auch das Ziel enthalten ist. Dies ist ausreichend, da ein Paket entlang weiterer Spannbäume gar nicht weitergeleitet werden kann. Davon ausgehend, kann aber durch die folgenden Verfahren die Anzahl der notwendigen Fehlerbeschreibungen weiter reduziert werden. Die hier vorgestellten Verfahren werden nur als Konzepte zur weiteren Verbesserung vorgeschlagen, wurden aber in der späteren Evaluierung nicht verwendet.

6.5.1.1 Aggregation von Fehlerbeschreibungen

Während ein Paket seinen Weg durch das Netz zurücklegt, werden diesem verschiedene Fehlerbeschreibungen hinzugefügt. Da sich die durch diese Fehler beschriebenen, ausgeschlossenen Teilbäume gegenseitig enthalten können, ist es möglich, die Fehlerbeschreibungen während der Weiterleitung der Pakete zu aggregieren. Es müssen jeweils nur die umfassendsten Fehlerbeschreibungen im Paketkopf verbleiben, während dadurch bereits umfasste Fehlerbeschreibungen entfernt werden können.

Diese Form der Aggregation verliert keine relevante Information und erhält die Garantie der Schleifenfreiheit von GFCP.

6.5.1.2 Beschränkung der möglichen Fehlerbeschreibungen

Eine Möglichkeit, die Anzahl von Fehlerbeschränkungen zu reduzieren besteht darin, nur eine feste, maximale Anzahl im Paket zuzulassen. Durch unvollständige Fehlerbeschreibungen entsteht jedoch die Möglichkeit, dass Pakete in Schleifen geraten können. Ziel eines solchen Ansatzes muss daher sein, einerseits die maximale Anzahl zulässiger Fehlerbeschreibungen auf einen geeigneten Wert zu parametrisieren und andererseits eine geeignete Verdrängungsstrategie für Fehlerbeschreibungen zu entwickeln, die die Wahrscheinlichkeit für die Entstehung von Schleifen minimiert. Mögliche Verdrängungsstrategien, die zwar im Rahmen dieser Arbeit nicht weiter untersucht wurden, aber für dieses Szenario zum Einsatz kommen könnten, sind beispielsweise:

- Unter der Annahme, dass aktuellere Fehlerbeschreibungen wichtiger sind, könnte eine FIFO-Disziplin gewählt werden
- Bei der Verwendung von Lokalitätsebenen sind Fehler auf größeren Spannbäumen weiter sichtbar. Ein einfacher Ansatz wäre es, diese in Fehlerbeschreibungen gegenüber lokaleren Baum-Ebenen zu bevorzugen. Erweiternd könnte man abhängig je weiter sich ein Paket an das Ziel annähert lokalere Baum-Ebenen bevorzugen.

6.5.2 Effiziente Darstellung einzelner Fehlerbeschreibungen

Während die vorherigen Ansätze versuchen, die Anzahl von Fehlerbeschreibungen, die in den Paketkopf aufgenommen werden müssen, zu reduzieren, werden hier Möglichkeiten vorgestellt, um eine gegebene Fehlerbeschreibung effizienter darzustellen. Das folgende Beispiel einer unoptimierten Fehlerbeschreibung, an der die vorgestellten Verfahren auch angewendet werden, verdeutlicht mit einer Größe von 392 Bits (49 Bytes) den Bedarf einer effizienteren Darstellung:

```
1 [Baum-ID] (4 Byte)
2 [Länge c1] (1 Byte)
3 [c1] -4:-4:-4:-4:-4:-3:-3:-3:-3:3:3:2:2:-2:2:-2:-2:2:1:1 (21 Bytes)
4 [Länge c2] (1 Byte)
5 [c2] -5:-5:-5:-5:-5:-4:-4:-4:-4:4:4:3:3:-3:3:-3:-3:3:2:2:1 (22 Bytes)
6 Länge: 392 Bits
```

Block	B_1	B_2		B_3	B_4
Koordinate c	-4	3	3	2	1
Vorzeichen	$-$	$+$	$+$	$+$	$+$
Kodierung b	10	01	01	11	01

Abbildung 6.14: Struktur einer virtuellen Koordinate von PIE

6.5.2.1 Extraktion von Richtungsinformation

Die Größe einer Fehlerbeschreibung kann durch Extraktion der Richtungsinformation weiter reduziert werden. Für die Entscheidung, ob eine Kante ausgefallen ist oder nicht, sind die Gewichte der Kante nicht relevant, machen aber einen signifikanten Anteil des Speicherbedarfs aus. Um ausreichend Kantengewichte darstellen zu können, könnte man eine Koordinate beispielsweise in einem Byte darstellen (1 Bit Vorzeichen, 7 Bit Kantengewichte). In einer Fehlerbeschreibung muss aber nur die Information mitgeführt werden, die eine Beurteilung der relativen Lage der Kante ermöglicht und dies umfasst explizit nicht das Kantengewicht. Abbildung 6.14 stellt die Struktur einer virtuellen Koordinate von PIE dar. Diese Koordinate besteht aus den Blöcken $B_1, ..., B_4$ und beschreibt damit einen 4 Hops langen Weg $(- \to ++ \to + \to +)$ durch den Baum von der Wurzel bis zum Knoten mit dieser virtuellen Koordinate mit einem Kantengewicht von jeweils 1. Die minimale Information, die zur Bestimmung der relativen Lage eines Knotens relevant ist, ist einerseits die Länge der Blöcke und andererseits die Vorzeichen innerhalb der Blöcke.

Für einen Block B_i, der aus $1 \leq j \leq |B_i|$ Koordinaten c_{ij} mit betragsmäßig gleichem Wert besteht, kann diese Information mit 2 Bits b_{ij} je Koordinate c_{ij} dargestellt werden:

$$b_{ij} = i \mod 2 \,||\, f_{01}(c_{ij})$$

wobei || die Konkatenation bezeichnet mit

$$f_{01}(c_{ij}) = \begin{cases} 0 & \text{if } c_{ij} < 0; \\ 1 & \text{if } c_{ij} > 0. \end{cases}$$

Das erste Bit von b_{ij} erlaubt damit die Unterscheidung von Blöcken (Block-Bit), während das zweite Bit für die Richtungskodierung zuständig ist (Richtungs-Bit). Die virtuelle Koordinate eines Knotens u wird in der Richtungsdarstellung b^u durch die Konkatenation der b_{ij} kodiert, wie in der letzten Zeile von Abbildung 6.14 für das vorhergehende Beispiel gezeigt. Für die in Abbildung 6.14 dargestellte virtuelle Koordinate bedeutet diese Richtungsdarstellung eine Verbesserung um einen Faktor 4, da ein Byte nun durch 2 Bits dargestellt wird.

Schließlich muss noch die Funktion zur Bestimmung, ob eine Kante auf dem Weg zwischen zwei Knoten liegt, entsprechend angepasst werden. Als Hilfsfunktion wird dazu zunächst die Funktion $\text{lca}(u, v)$ definiert, die den letzten gemeinsamen Vorfahren der Knoten s und t im Baum T bestimmt und diesen in Richtungsdarstellung als Ergebnis liefert (siehe Listing 6.1). Diese Hilfsfunktion wird dazu verwendet, um den letzten gemeinsamen Vorfahren

Listing 6.1: Bestimme den letzten gemeinsamen Vorfahren der Knoten s und t im Baum T. Rückgabewert ist dessen Adresse in Richtungsdarstellung.

```
lca(s^T, t^T) {
  result ← NULL
  limit ← min(Blocks in b^s,Blocks in b^t)

  for (i=1; i <= limit; ++i) {
    if ( b_i^s == b_i^t ) // bitweise Gleichheit des Blocks i
      result ← result||b_i^s // Konkatenation
    else
      break
  }
  return result // in Richtungsdarstellung
}
```

eines möglichen Weiterleitungsknotens mit dem Ziel in Richtungsdarstellung zu bestimmen. Dies ist notwendig, um mit den in Richtungsdarstellung vorliegenden Fehlerdarstellungen arbeiten zu können.

Ein weiteres Hilfskonstrukt ist das folgende ancestor-Prädikat ancestor(b^p, b^c), das für zwei in Richtungsdarstellung gegebene Koordinaten b^p und b^c feststellt, ob der Knoten p ein Elternknoten von Knoten c ist. Dies genau dann der Fall, wenn b^p ein Präfix von b^c ist:

$$ancestor(b^p, b^c) = \begin{cases} wahr & \text{wenn } lcp(b^p, b^c) == b^p; \\ falsch & \text{sonst.} \end{cases}$$

Die Funktion lcp (eng.: longest common prefix) liefert dabei das längste gemeinsame Präfix der beiden Richtungsdarstellungen als Ergebnis und das ancestor-Prädikat ist wahr, wenn dieses Ergebnis die Richtungsdarstellung des Knotens p ist und andernfalls falsch.

Damit kann nun (analog zu Gleichung 6.3)festgestellt werden, ob eine in Richtungsdarstellung vorliegende Kante (b^u, b^v) aus dem Spannbaum T entlang dieses Spannbaums zwischen zwei Knoten s und t liegt, deren virtuelle Koordinaten in T aus dem Paketkopf bekannt sind.

$onpath(T, b^u, b^v, s^T, t^T) =$

$$\begin{array}{lll} (1) & & ancestor(\ lca(s^T, t^T), b^u\ ) \wedge ancestor(\ lca(s^T, t^T), b^v\ ) \\ (2) & \wedge & [ancestor(b^u, b^s) \wedge ancestor(b^v, b^s) \\ (3) & & \vee\ ancestor(b^u, b^t) \wedge ancestor(b^v, b^t)] \end{array}$$

Bedingung (1) stellt dabei sicher, dass sich sowohl Knoten u als auch Knoten v im Teilbaum unterhalb des letzten gemeinsamen Vorfahren der Knoten s und t befinden, was eine notwendige Voraussetzung dafür ist, dass die Kante (u,v) überhaupt auf dem Pfad liegen kann. Zusätzlich zu Bedingung (1) muss eine der Bedingungen (2) oder (3) gelten. Bedingung

(2) beschreibt dabei den Fall, dass die Knoten u und v zwischen dem letzten gemeinsamen Vorfahren von s und t und s liegen. Bedingung (3) beschreibt den Fall, dass die Knoten u und v zwischen dem letzten gemeinsamen Vorfahren von s und t und t liegen.

Analog zu Gleichung 6.4 kann dann folgendermaßen überprüft werden, ob ein Pfad zwischen zwei Knoten s und t bei Ausfall der Kante (u, v) gültig ist:

$$\text{valid}(T, b^u, b^v, s^T, t^T) := \neg\text{onpath}(T, b^u, b^v, s^T, t^T) \tag{6.10}$$

Unter Nutzung eines Verweises auf den Baum-Identifikator der Zieladresse sowie der Extraktion von Richtungsinformationen der Koordinaten ergibt sich für die eingangs beispielhaft vorgestellte Fehlerbeschreibung:

```
[Baum-ID] (4 Byte)
[Länge c1] (1 Byte)
[c1] 10 10 10 10 10 00 00 00 00 00 01 01 11 11 10 11 10 10 11 01 01 (42 Bits)
[Länge c2] (1 Byte)
[c2] 10 10 10 10 10 00 00 00 00 00 01 01 11 11 10 11 10 10 11 01 01 10 (44 Bits)
Länge: 134 Bits
```

Die Länge wurde von 392 Bits auf 134 Bits reduziert.

6.5.2.2 Ausnutzen vorhandener Information

Wie eingangs erwähnt, werden in den Paketkopf nur Fehlerbeschreibungen eingetragen, die sich auf einen Spannbaum beziehen, in dem auch das Ziel enthalten ist. Alle Spannbäume, auf die sich Fehlerbeschreibungen beziehen sind also bereits als Teil der Zieladresse im Paket enthalten. Statt nun in jede Fehlerbeschreibung einen Identifikator für den Spannbaum, auf den die Beschreibung sich bezieht, aufzunehmen, kann mit nur $\lceil \log_2 l \rceil$ Bits ein Verweis auf den relevanten Spannbaum in der Zieladresse erfolgen.

Werden, wie von PIE, Knoten-Identifikatoren von beispielsweise 4 Bytes als Identifikatoren für Spannbäume verwendet, so wäre eine Kodierung selbst für $l = 16$ in nur 4 Bits möglich. Bei der von PIE abhängig von der Anzahl Knoten n in einem Netze empfohlenen Anzahl von Baum-Ebenen $\lfloor \log_2(n) - 7 \rfloor$ wäre dies für Netze bis zur Größe von $2^{24} - 1 \approx 16$ Millionen Knoten ausreichend.

Damit lässt sich für die Beispielkoordinate eine weitere Reduktion auf 106 Bits erreichen.

```
[Baum-ID] (4 Bits)
[Länge c1] (1 Byte)
[c1] 10 10 10 10 10 00 00 00 00 00 01 01 11 11 10 11 10 10 11 01 01 (42 Bits)
[Länge c2] (1 Byte)
[c2] 10 10 10 10 10 00 00 00 00 00 01 01 11 11 10 11 10 10 11 01 01 10 (44 Bits)
Länge: 106 Bits
```

6.5.2.3 Differentielle Kodierung

Eine Fehlerbeschreibung enthält zwei virtuelle Koordinaten, die sich in ihrer Struktur ähnlich sind, da sie in diesselbe Richtung führen. In der zuvor vorgeschlagenen Kodierung der Richtungsinformation entspricht redundante Information damit einem identischen Präfix und der tiefer im Baum gelegene Knoten besitzt einen Block mehr. Um diese Redundanz

zu eliminieren, kann eine Fehlerbeschreibung bestehend aus zwei Kodierungen b^u und b^v, wovon o. B. d. A. v der tiefer im Baum gelegene Knoten sei, als b^v notiert werden. Die zugehörige Semantik bestimmt dann, dass für eine Fehlerbeschreibung mit der Kodierung b^v mit j Blöcken die zweite Koordinate die ersten $j-1$ Blöcke von b^v umfasst. Dadurch wird neben einer virtuellen Koordinate zusätzlich deren Längenfeld eingespart.

Damit ergibt sich für die als Beispiel verwendete Fehlerbeschreibung die folgende Kodierung:

```
1 [Baum-ID] (4 Bits)
2 [Länge c2] (1 Byte)
3 [c2] 10 10 10 10 10 00 00 00 00 00 01 01 11 11 10 11 10 10 11 01 01 10 (44 Bits)
4 Länge: 56 Bits
```

Die Länge der Fehlerbeschreibung wurde dabei durch die konsequente Anwendung der hier vorgestellten Kodierungsverfahren von anfangs 392 Bits (49 Bytes) auf eine Länge von nur 56 Bits (7 Bytes) deutlich reduziert und damit praxistauglicher gemacht.

Um ein Gefühl für die Größe von Fehlerbeschreibungen auf realistischen Topologien zu geben, ist die durchschnittliche Länge einer virtuellen Koordinate auf der später untersuchten DIMES Topologie 15.6 Koordinaten. Für angenommene 16 Koordinaten ergibt sich mit der hier vorgestellten Kodierung also für eine Fehlerbeschreibung durchschnittlicher Länge ein Platzbedarf von 44 Bits (5,5Ḃytes). Für die maximale beobachtete Länge einer virtuellen Koordinate von 33 ergibt sich eine Kodierung mit einer Länge von 78 Bits (9,75Ḃytes).

6.6 Evaluierung

Um die Effektivität des im Rahmen dieser Arbeit entwickelten Rerouting-Verfahrens Greedy Failure-Carrying Packets zu demonstrieren, wird dieses im Folgenden simulativ auf sein Verhalten unter Knoten- und Linkausfällen untersucht. Als zugrundeliegendes Routingprotokoll wurde PIE gewählt und, wie in Abschnitt 6.5 beschrieben, um Greedy Failure-Carrying Packets erweitert (in nachfolgenden Abbildungen bezeichnet als Rerouting: GFCP). Im Vergleich dazu wird einerseits PIE ohne Einsatz einer Rerouting-Strategie betrachtet (in nachfolgenden Abbildungen bezeichnet als Rerouting: None), um die Auswirkungen des Einsatzes einer Rerouting-Strategie zu untersuchen. Als alternative Rerouting-Strategie wird ein Vergleich mit dem in Abschnitt 6.3.2.2 vorgestellten Gravity-Pressure Routing durchgeführt, das direkt auf PIE anwendbar ist (in nachfolgenden Abbildungen bezeichnet als Rerouting: GP). Als weiterer Bezugspunkt wurde die Zustellrate unter Routing auf kürzesten Pfaden verwendet. Dazu wurde für jedes Sender-/Empfängerpaar ein kürzester Pfad berechnet, über den Pakete genau dann zugestellt werden konnten, wenn keine Teilstrecke ausgefallen war. Um eine gute Intuition für das Verhalten von GFCP unter verschiedensten Bedingungen zu erhalten, wurde in den folgenden Experimenten ein sehr breiter Parameterbereich gewählt und beispielsweise auch sehr extreme Ausfälle von bis zu 25% der Knoten beziehungsweise Kanten der Topologie untersucht. Der folgende Abschnitt gibt zunächst einen detaillierten Überblick über die verwendeten Konfigurationen bevor die Ergebnisse der Simulationen unter Knotenausfällen und anschließend unter Kantenausfällen vorgestellt und diskutiert werden.

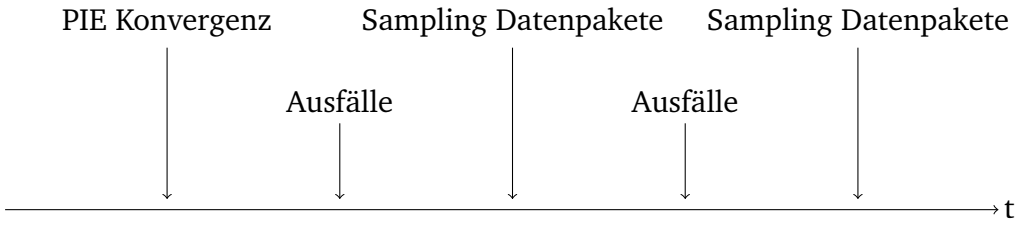


Abbildung 6.15: Zeitliche Abfolge bei der Evaluierung von GFCP: Nach der Konvergenz des zugrundeliegenden Routingprotokolls PIE finden die Ausfälle statt. Auf der resultierenden Topologie werden anschließend die Sampling-Pakete versendet.

6.6.1 Konfigurationen

Zur Evaluierung von GFCP wird die in monatlichen Abständen veröffentlichte DIMES Topologie vom März 2010 herangezogen. Diese Topologie wurde bereits in der Evaluierung von PIE zur Untersuchung von Knotenausfällen verwendet [44] und wird daher für eine gute Vergleichbarkeit der Ergebnisse hier ebenfalls verwendet. Wie zuvor werden Kanten als ungerichtet betrachtet, Schleifen und Mehrfachkanten entfernt und die größte Zusammenhangskomponente verwendet. Die resultierende Topologie enthält 26424 Knoten sowie 90267 Kanten und hat einen Durchmesser von 8 Übertragungsabschnitten.

Abbildung 6.15 skizziert den zeitlichen Ablauf einer Simulation. Zu Beginn eines Simulationslaufs werden zunächst die Wurzeln der Spannbäume auf allen l Baum-Ebenen festgelegt. Statt diese wie bei PIE durch einen Zufallsprozess mit erwarteten 2^{i-1} Wurzeln auf Ebene i festzulegen, wird eine feste Anzahl von 2^{i-1} Wurzeln auf Ebene i gewählt. Für Ebene 1 wird, wie in PIE empfohlen, der Knoten mit dem höchsten Grad als Wurzel gewählt, während auf den weiteren Ebenen exakt die erwünschte Anzahl von Wurzeln zufällig gleichverteilt aus der Menge der Knoten V gewählt wird. Gemäß PIE werden von den festgelegten Wurzeln aus nun Spannbäume aufgebaut und virtuelle Koordinaten innerhalb der Bäume bzw. Adressen and die Knoten vergeben (s. Abbildung 6.15, PIE Konvergenz). Damit können sich in Abwesenheit von Ausfällen zunächst alle Knoten erreichen und alle Pakete können zugestellt werden.

Um die Auswirkung von Ausfällen auf die jeweiligen Rerouting-Strategien zu untersuchen, wurde ein einfaches Dynamikmodell zugrundegelegt, bei dem jeweils ein Anteil δ aller Knoten (bzw. Kanten) ausfällt. Dabei werden für $\delta = \{0.01, 0.02, ..., 0.10, 0.15, 0.20, 0.25\}$ jeweils $\delta \cdot n$ Knoten (bzw. $\delta \cdot m$ Kanten) zufällig gleichverteilt deaktiviert. Der Bereich von $\delta = \{0.01, 0.02, ..., 0.10\}$ wurde dabei in 1%-Schritten genauer untersucht, da Ausfälle in dieser Größenordnung in der Praxis eher zu erwarten sind. Zusätzlich wurde der Bereich bis 25% Ausfälle noch in 5%-Schritten untersucht, um das Verhalten von GFCP unter extremen Bedingungen zu analysieren. Wie Abbildung 6.15 darstellt, werden nach jedem Ausfall Sampling-Datenpakete versendet. Für jeden Simulationslauf werden dazu fünf Ausfälle mit unterschiedlichen Knoten (bzw. Kanten) durchgeführt, wobei zwischen den Ausfällen jeweils eine Rückkehr in den Ausgangszustand mit vollständiger gegenseitiger Erreichbarkeit erfolgt. Nach jedem Ausfall werden 4000 Pakete zwischen zufällig gleichverteilt ausgewähl-

ten Sender- und Empfängerknoten gesendet, um über die Zustellrate auf die verbleibende gegenseitige Erreichbarkeit im Netz zu schließen. Damit werden pro Simulationslauf 20000 Sampling-Pakete und pro Konfiguration 100000 Sampling-Pakete versendet.

Tabelle 6.3: Untersuchter Parameterraum für Knotenausfälle auf der DIMES Topologie

Parameter	Werte
Baumebenen l	$\{1, 2, 4, 6, 8\}$
Anteil ausgefallener Knoten δ	$\{0.01, 0.02, ..., 0.10, 0.15, 0.20, 0.25\}$
Rerouting-Verfahren	{None, GP, GFCP}
Wiederholungen	5
Ausfälle pro Wiederholung	5
Sampling Datenpakete pro Ausfall	4000
Time-To-Live	64

Tabelle 6.3 zeigt die für die Untersuchung von Knotenausfällen im folgenden Abschnitt 6.6.2 verwendeten Parameter. Mit $|l| \cdot |\delta| \cdot |\text{Rerouting-Verfahren}| = 5 \cdot 13 \cdot 3$ ergeben sich 195 Konfigurationen, die jeweils mit 5 Seeds durchgeführt wurden und damit in insgesamt 975 Simulation resultieren.

Tabelle 6.4: Untersuchter Parameterraum für Kantenausfälle auf der DIMES Topologie

Parameter	Werte
Baumebenen l	$\{1, 2, 4, 6, 8\}$
Anteil ausgefallener Kanten δ	$\{0.01, 0.02, ..., 0.10, 0.15, 0.20, 0.25\}$
Rerouting-Verfahren	{None, GP, GFCP}
Wiederholungen	5
Ausfälle pro Wiederholung	5
Sampling Datenpakete pro Ausfall	4000
Time-To-Live	64

Tabelle 6.4 fasst die für die Evaluierung von GFCP unter Kantenausfällen (s. Abschnitt 6.6.3) gewählten Parameterbereiche zusammen. Damit ergeben sich $|l| \cdot |\delta| \cdot |\text{Rerouting-Verfahren}| = 5 \cdot 13 \cdot 3 = 195$ Konfigurationen, für die jeweils fünf Wiederholungen durchgeführt werden. Damit werden insgesamt 975 Simulationsläufe durchgeführt.

6.6.2 GFCP bei Knotenausfällen

Für die durchgeführten Simulationen werden zunächst die Konfigurationen mit Knotenausfällen (siehe Tabelle 6.3) untersucht und im Hinblick auf die Zustellrate analysiert. Anschlie-

ßend werden die Rerouting-Verfahren auf den durchschnittlichen und maximalen Stretch hin untersucht.

6.6.2.1 Zustellrate

Anhand von Abbildung 6.16 wird zunächst der Anteil zugestellter Pakete der Rerouting-Strategien bei Knotenausfällen betrachtet. Abbildung 6.16 gibt zunächst einen makroskopischen Überblick über die Ergebnisse der Untersuchungen mit Knotenausfällen. Dort sind die Anteile zugestellter und verworfener Pakete für die drei untersuchten Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) dargestellt. Die Abbildung gliedert sich in 15 Teil-Abbildungen, für die auf der x-Achse jeweils der Anteil ausgefallener Knoten δ und auf der y-Achse der Anteil zugestellter und verworfener Pakete aufgetragen ist. Die erste Zeile zeigt die Ergebnisse von PIE ohne Einsatz eines Rerouting-Verfahrens (None), die zweite Zeile die Ergebnisse von PIE unter Verwendung von Gravity-Pressure Routing (GP) und die dritte Zeile schließlich die Ergebnisse für Greedy Failure-Carrying Packets (GFCP). Die Spalten der Abbildung stellen die verwendete Anzahl von Baum-Ebenen von $l = 1$ bis $l = 8$ dar.

Als Ausgangspunkt der Untersuchung und Vergleichsbasis für die Rerouting-Verfahren dient **PIE ohne Rerouting**, dessen Ergebnisse in der ersten Zeile der Abbildung dargestellt sind. Dabei ist zunächst erkennbar, dass für alle Baumebenen der grün dargestellte Anteil zugestellter Pakete mit zunehmenden Knotenausfällen, wie zu erwarten, sinkt. Die Nutzung mehrerer Baum-Ebenen erhöht dabei den Anteil zugestellter Pakete im extremsten betrachteten Fall $\delta = 0.25$ von 63% ($l = 1$) zunächst auf 73% ($l = 2$) bis 76% ($l = 4$). Die Nutzung weiterer Baum-Ebenen erhöht den Anteil zugestellter Pakete nur noch mariginal auf bis zu 77% ($l = 8$). Die Erhöhung der Zustellrate mit steigender Anzahl von Spannbäumen l begründet sich durch die damit zur Verfügung stehende erhöhte Pfad-Diversität. Trifft ein Paket in einem Baum auf ein lokales Minimum, so stehen beim Vorhandensein mehrerer Spannbäume eventuell noch weitere Spannbäume bereit, die das Paket zu gleichen oder gar geringeren Kosten in Richtung des Ziels weiterleiten können. Ist dies, wie insbesondere für $l = 1$ zu beobachten, nicht der Fall, so muss PIE ohne Einsatz eines Rerouting-Verfahrens diese Pakete verwerfen.

Bei nur einer Baum-Ebene ($l = 1$) wird ein daraus resultierender Effekt sichtbar: für $\delta \in \{0.05, 0.09, 0.10, 0.20, 0.25\}$ fällt der Anteil zugestellter Pakete stark ab, während dieser Effekt mit einer erhöhten Anzahl von Baum-Ebenen deutlich abnimmt. Als Ursache dafür wurde die Ausfallshäufigkeit des höchstgradigen Knotens ausgemacht, die in Abbildung 6.17 über dem Anteil ausgefallener Knoten δ dargestellt ist. Da der höchstgradige Knoten die Wurzel des Spannbaums auf der Baum-Ebene 1 ist, führen bereits durch die Baumstruktur viele Wege über diesen Knoten. Zusätzlich ist in Internet-ähnlichen Topologien der Knotengrad stark mit der Zentralität des Knotens korreliert [106]. Dies bedeutet im Umkehrschluss, dass Pfade, die um diesen zentralen Knoten herum über alternative Knoten führen, tendentiell länger sind und damit bei gierigem Routing nicht genutzt werden können. Der Ausfall dieses höchstgradigen Knotens stellt damit bei nur einer Spannbaum-Ebene ein extremes Ereignis dar, da dort nur eine geringe Pfad-Diversität zur Verfügung steht und der Anteil zugestellter Pakete daher deutlich sinkt.

PIE ohne Rerouting verwirft gemäß dem Greedy-Prinzip Pakete sofort, wenn über keinen

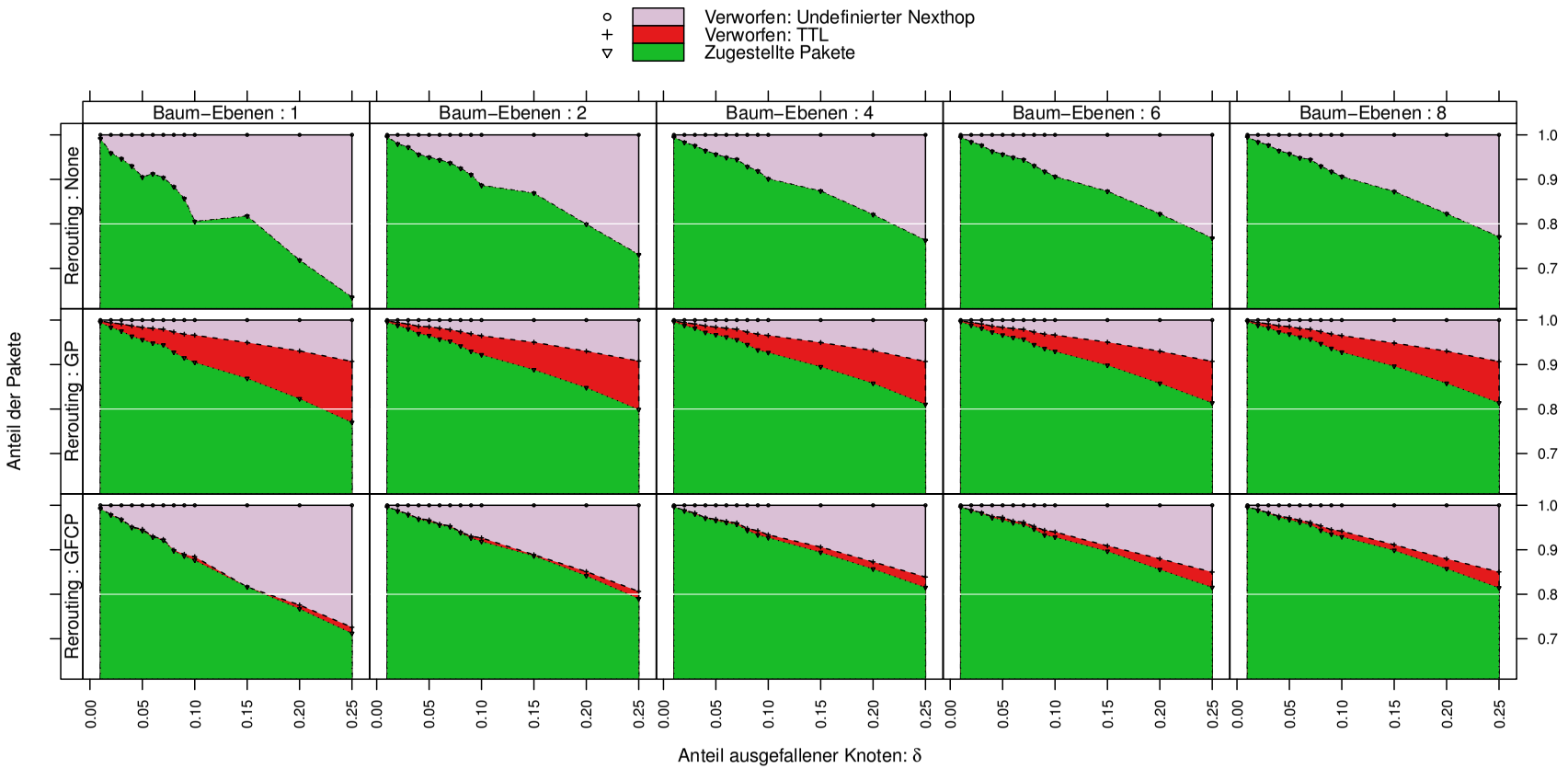


Abbildung 6.16: Anteile zugestellter und verworfener Pakete für die Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) für Knotenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

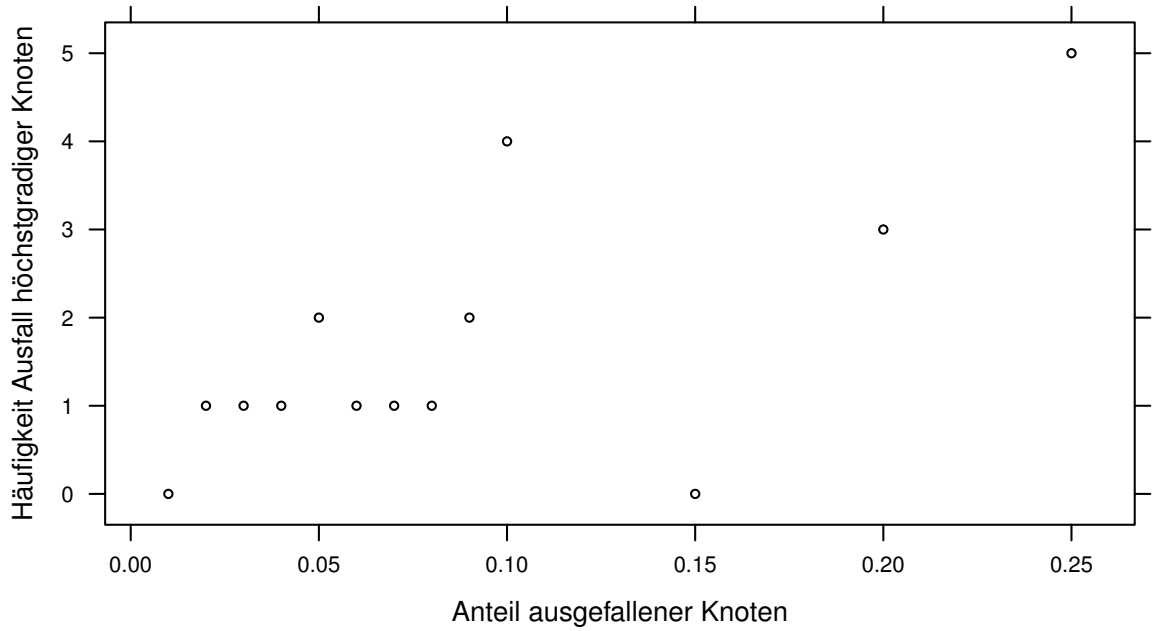


Abbildung 6.17: Anzahl der Ausfälle des Knotens mit dem höchsten Grad pro Konfiguration

Nachbarn ein Fortschritt in Richtung des Ziels mehr möglich ist. Dadurch wurde in keiner Simulation die Time-To-Live (rot dargestellter Anteil) erreicht, sondern es wurden alle Pakete an einem lokalen Minimum verworfen (grau dargestellt).

Der Einsatz von **Gravity-Pressure Routing** führt in allen Fällen zu einer Verbesserung der Paketzustellrate. Betrachtet man wieder den Extrempunkt $\delta = 0.25$, so ist diese Verbesserung bei $l = 1$ mit 15% sehr hoch, während die Verbesserung für $l \in \{2, 4, 6, 8\}$ in allen Fällen kleiner gleich 6% ausfällt. Bemerkenswert ist an dieser Stelle noch, dass der zuvor bei PIE ohne Verwendung eines Rerouting-Verfahrens aufgetretene Effekt einer starken Verschlechterung der Zustellrate in Abhängigkeit von Ausfällen des höchstgradigen Knotens für $l = 1$ bei Gravity-Pressure Routing fast nicht sichtbar ist, obwohl dort analoge Ausfälle des höchstgradigen Knotens vorlagen. Dies wird dadurch möglich, dass Pakete im Gegensatz zu rein gieriger Paketweiterleitung nun auch über längere Wege zum Ziel weitergeleitet werden können. In dem bei Internet-ähnlichen Topologien sehr dicht vermaschten Kern gelingt es Gravity-Pressure Routing dann, alternative Pfade zu finden und Pakete über diese um den ausgefallenen höchstgradigen Knoten herum zuzustellen.

Gravity-Pressure Routing nimmt im Gegensatz zu PIE ohne Rerouting Umwege in Kauf, was sich zunächst in einer Verlängerung der Pfade und idealerweise in einer erhöhten Zustellrate niederschlägt. Wie in Abbildung 6.16 dargestellt, werden für Gravity-Pressure Forwarding im Gegensatz zu PIE ohne Rerouting Pakete wegen Überschreitung der Time-To-Live von 64 Hops verworfen. In 98% aller GP-Konfigurationen war der Anteil von Paketen, die wegen einer Überschreitung der TTL verworfen wurden (rot), mindestens gleich hoch wie der Anteil der Pakete, die an einem lokalen Minimum verworfen wurden (grau). Diese Eigenschaft von Gravity-Pressure Routing, dass es kein frühzeitiges Abbruchkriterium für das Verwerfen von

Paketen gibt, spielt später bei der Betrachtung des im Netz verursachten Aufwandes eine entscheidende Rolle.

Greedy Failure-Carrying Packets steigert die Paketzustellrate im Vergleich zu PIE ohne Rerouting um bis zu 7%. Für $l \geq 2$ oder mehr Baum-Ebenen liegen Gravity-Pressure Routing und GFCP mit einem Unterschied von weniger als 1% zugestellter Pakete gleich auf. Steht nur ein einzelner Spannbaum zur Weiterleitung zur Verfügung ($l = 1$), so ist Gravity-Pressure Routing GFCP um bis zu 5% überlegen. Diese starke Abhängigkeit GFCPs von der Anzahl Baum-Ebenen l liegt darin begründet, das GFCP im Gegensatz zu GP nur Pfade in seiner Weiterleitungsentscheidung berücksichtigt, die nicht in einen unzulässigen, durch Fehlerbeschreibungen ausgeschlossenen Teilbaum führen. Ohne ausreichende, durch mehrere Baum-Ebenen bereitgestellte Pfad-Diversität führt dies zu einem schnellen Verwerfen von Paketen durch GFCP. Während dadurch für $l = 1$ zwar weniger Pakete zugestellt werden als durch Gravity-Pressure Routing, gelingt es auch GFCP die Ausfälle des höchstgradigen Knotens bei $\delta \in \{0.05, 0.09, 0.10, 0.20, 0.25\}$ auszugleichen, die bei PIE ohne Rerouting in einer deutlich reduzierten Zustellrate resultierten.

Eine interessante Beobachtung kann bezüglich der wegen einer TTL-Überschreitung verworfenen Pakete gemacht werden. Während diese bei GFCP auch auftraten, betrug deren Anteil maximal 3%. Im Gegensatz zu Gravity-Pressure Routing, bei dem mehr Baum-Ebenen zu einer Reduktion der wegen TTL-Überschreitung verworfenen Pakete führen (roter Bereich nimmt ab), steigt bei GFCP der Anteil dieser Pakete bei mehr Baum-Ebenen sogar leicht an. Dieser Anstieg bei GFCP kommt durch die erhöhte Pfad-Diversität zustande, die dazu führt, dass bis zum Ablauf der TTL noch gültige Pfade existieren, über die das Paket eventuell hätte zugestellt werden können.

Abbildung 6.18 stellt die Zustellraten der Verfahren nocheinmal unterteilt nach Baum-Ebenen dar und stellt die Ergebnisse zusätzlich dem Anteil zugestellter Pakete bei Routing entlang kürzester Pfaden gegenüber. Dabei werden zwei Effekte deutlich:

- In allen Konfigurationen mit $l \geq 2$, ist PIE sowohl ohne als auch mit Rerouting im Hinblick auf die Zustellrate einem Routing entlang kürzester Pfade überlegen. PIE kann also bereits ohne Rerouting-Verfahren den Ausfall von Knoten, wie auch in der zugehörigen Veröffentlichung [44] demonstriert, relativ gut behandeln.
- Für $l \geq 2$ liegen die beiden betrachteten Rerouting-Verfahren GP und GFCP gleich auf und können eine Verbesserung der Zustellrate gegenüber PIE ohne Rerouting um bis zu 5% erreichen.

6.6.2.2 Stretch

In diesem Abschnitt wird zunächst der durchschnittliche und anschließend der maximale Stretch der untersuchten Rerouting-Verfahren analysiert. Diese Metrik erfasst nur zugestellte Pakete, da der Stretch für nicht zugestellte Pakete nicht definiert ist.

Abbildung 6.19 zeigt den durchschnittlichen Stretch für alle *zugestellten* Pakete der untersuchten Konfigurationen. Auf der x-Achse ist dabei wieder der Anteil ausgefallener Knoten abgetragen und die y-Achse zeigt den durchschnittlichen Stretch. Die höhere Zustellrate, die Gravity-Pressure Forwarding insbesondere bei $l = 1$ im Vergleich zu PIE ohne Rerouting

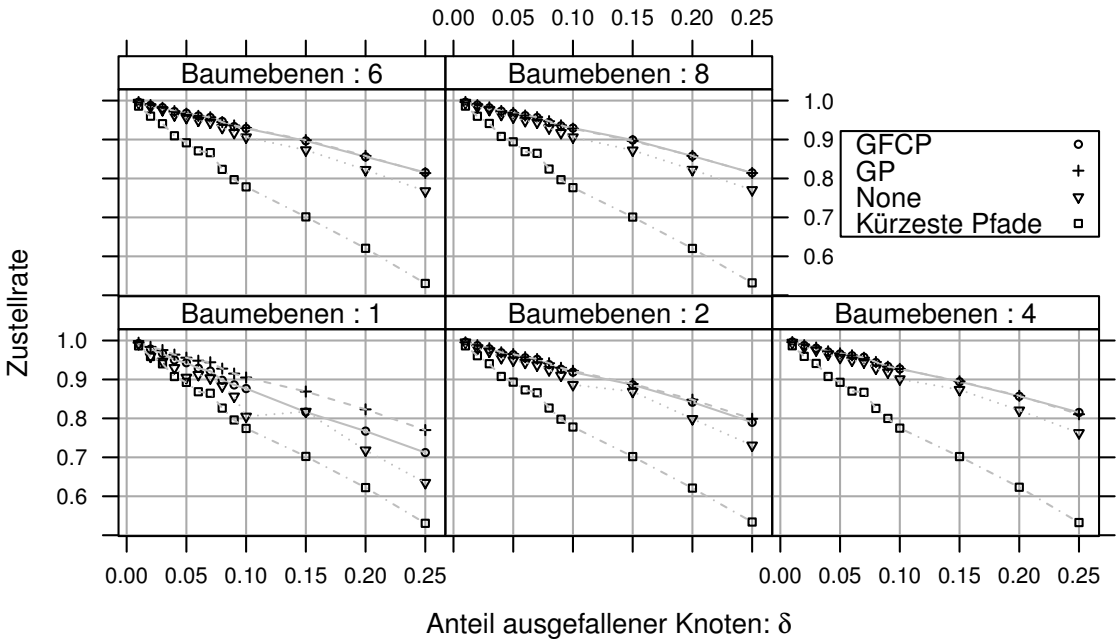


Abbildung 6.18: Anteile zugestellter Pakete für die Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) für Knotenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

und PIE mit GFCP erreicht, wird durch einen deutlich erhöhten durchschnittlichen Stretch erkauft. Diese Verschlechterung entsteht durch die Weiterleitung von Paketen im Pressure-Modus und ist auch im maximalen Stretch, der in Abbildung 6.20 dargestellt ist, deutlich sichtbar.

Abbildung 6.20 stellt den maximalen gemessenen Stretch in Abhängigkeit vom Anteil ausgefallener Knoten dar. Durch seine gierige Paketweiterleitung weist PIE hier keine nennenswerten Spitzen auf, da dieses Pakete an lokalen Minima sofort verwirft. GP und GFCP können Pakete durch die Inkaufnahme von Umwegen mit einer deutlichen Pfadverlängerung zustellen, die mit einer steigenden Anzahl von Baum-Ebenen durch eine größere Pfad-Diversität etwas zurückgeht. Bezüglich des maximalen hier beobachteten Stretch verhalten sich GP und GFCP ähnlich und der maximale Stretch sinkt leicht mit der Anzahl von Baum-Ebenen. Wie bereits zuvor anhand von Abbildung 6.17 erläutert, ist der Ausfall des höchstgradigen Knotens aufgrund seiner Zentralität von besonderer Bedeutung für die Erreichbarkeit und den Stretch zwischen Knotenpaaren. In den hier durchgeführten Simulationen wurde für $\delta = 0.15$ kein Ausfall des höchstgradigen Knotens beobachtet. Dies äußert sich auch in Abbildung 6.20 durch einen besonders guten maximalen Stretch von GFCP und zeigt die Bedeutung dieses Knotens für GFCP, falls nur ein Spannbaum zur Verfügung steht.

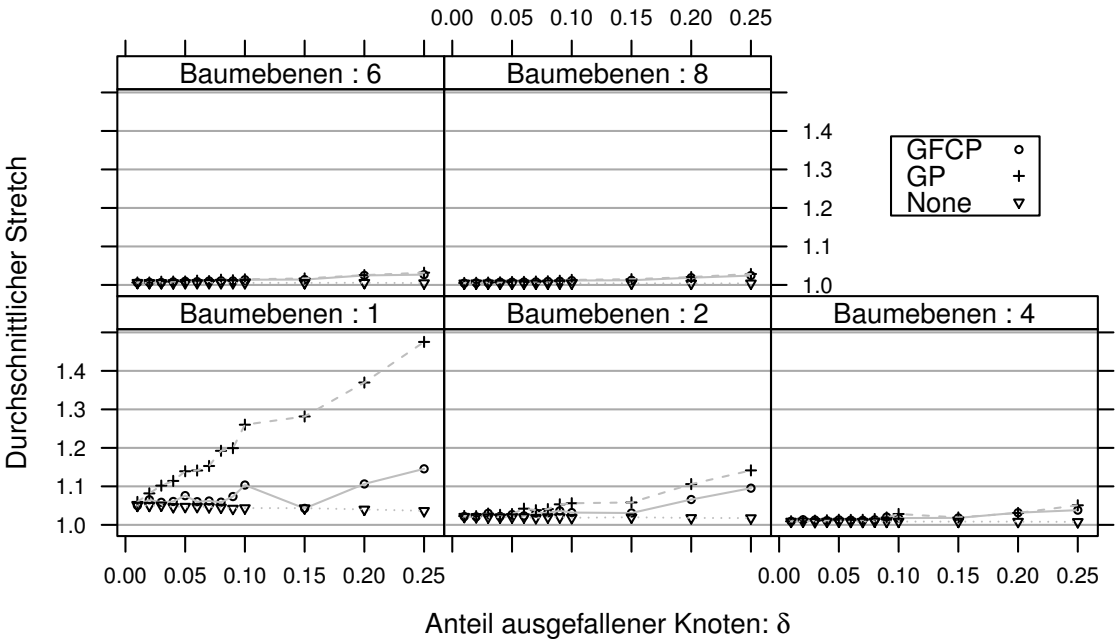


Abbildung 6.19: Durchschnittlicher Stretch für die Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) für Knotenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 6, 8\}$ Baumebenen

6.6.2.3 Zusammenfassung

Zunächst lässt sich für die untersuchten Knotenausfälle auf der DIMES-Topologie festhalten, dass bereits PIE ohne Rerouting-Verfahren deutlich besser abschneidet als eine Weiterleitung entlang kürzester Pfade. GFCP und GP können gegenüber PIE die Zustellrate leicht erhöhen und verhalten sich für $l \geq 2$ bezüglich durchschnittlichem wie auch maximalem Stretch sehr ähnlich. GFCP verwirft jedoch deutlich weniger Pakete aufgrund einer Überschreitung der TTL und belastet dadurch das Netz etwas weniger.

6.6.3 GFCP bei Kantenausfällen

Für die Experimente zum Verhalten der Rerouting-Verfahren unter Kantenausfällen (siehe Tabelle 6.4) wird zunächst wieder die Zustellrate untersucht. Anschließend wird wie zuvor der Stretch betrachtet, bevor der durch die Rerouting-Verfahren verursachte Mehraufwand untersucht wird.

6.6.3.1 Zustellrate

Abbildung 6.21 zeigt zunächst wieder für den gesamten Parameterraum das makroskopische Verhalten der Rerouting-Verfahren unter Kantenausfällen. Dort sind die Anteile zugestellter und verworfener Pakete für die drei untersuchten Rerouting-Verfahren GFCP, GP und ohne

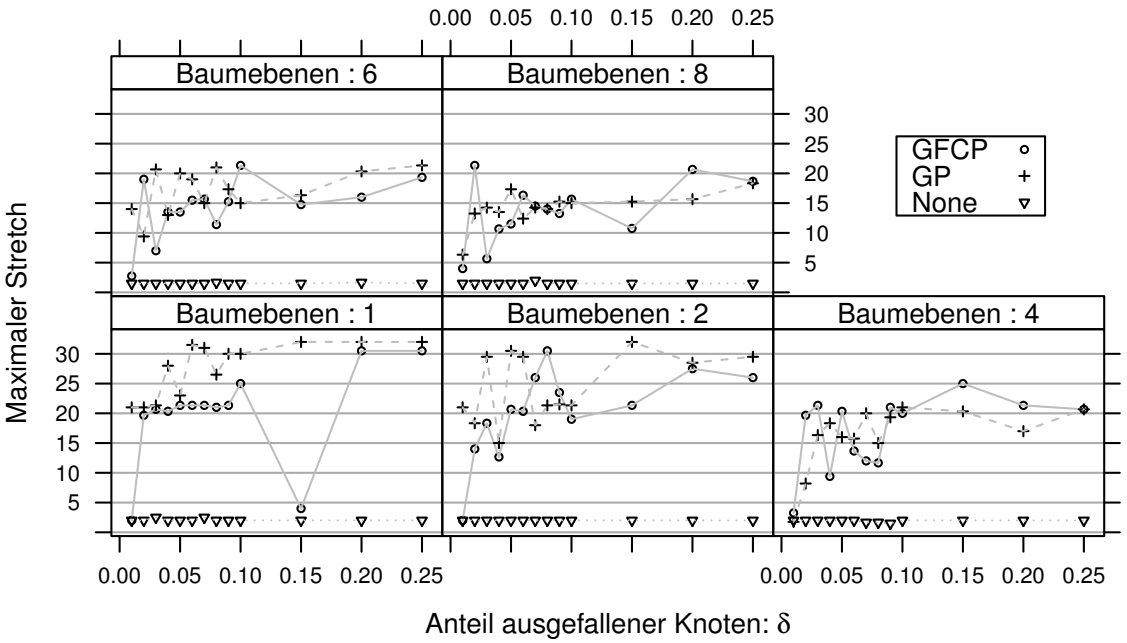


Abbildung 6.20: Maximaler Stretch für die Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) für Knotenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

Rerouting (None) dargestellt. Die Abbildung gliedert sich, wie auch bei den Knotenausfällen, in 15 Teil-Abbildungen, für die auf der x-Achse jeweils der Anteil ausgefallener Knoten δ und auf der y-Achse der Anteil zugestellter und verworfener Pakete aufgetragen ist. Die erste Zeile zeigt die Ergebnisse von PIE ohne Einsatz eines Rerouting-Verfahrens, die zweite Zeile die Ergebnisse von PIE unter Verwendung von Gravity-Pressure Routing und die dritte Zeile schließlich die Ergebnisse für Greedy Failure-Carrying Packets. Die Spalten der Abbildung stellen die verwendete Anzahl von Baum-Ebenen von $l = 1$ bis $l = 8$ dar.

Für **PIE ohne Einsatz eines Rerouting-Verfahrens** (erste Zeile), das hier wieder als Vergleichsbasis dient, ist zunächst wieder festzustellen, dass eine erhöhte Anzahl von Baum-Ebenen zu einer Erhöhung des Anteils der zugestellten Pakete führt. Dabei liegt die Zustellrate jedoch insgesamt auf einem sehr niedrigen Niveau und überschreitet beispielsweise für $\delta = 0.25$ in keinem Fall 40%. Dies ist insbesondere beim Vergleich mit den Knotenausfällen interessant, da dort eine deutlich höhere Zustellrate erreicht wird. Kantenausfälle sind also für die gierige Paketweiterleitung schwerwiegender als Knotenausfälle.

Der Einsatz von **Gravity-Pressure Rerouting** (zweite Zeile) resultiert im Vergleich zu PIE ohne Rerouting in einer deutlichen Erhöhung des Anteils der zugestellten Pakete. Betrachtet man $\delta = 0.25$, so führt der Einsatz von Gravity-Pressure Rerouting bereits für $l = 1$ zu mehr als einer Verdopplung der zugestellten Pakete und erhöht den Anteil von 28% auf 60%. Von einer erhöhten Anzahl an Baum-Ebenen kann Gravity-Pressure Rerouting zwar ebenfalls profitieren, allerdings ist die dadurch erreichte Verbesserung nicht besonders stark.

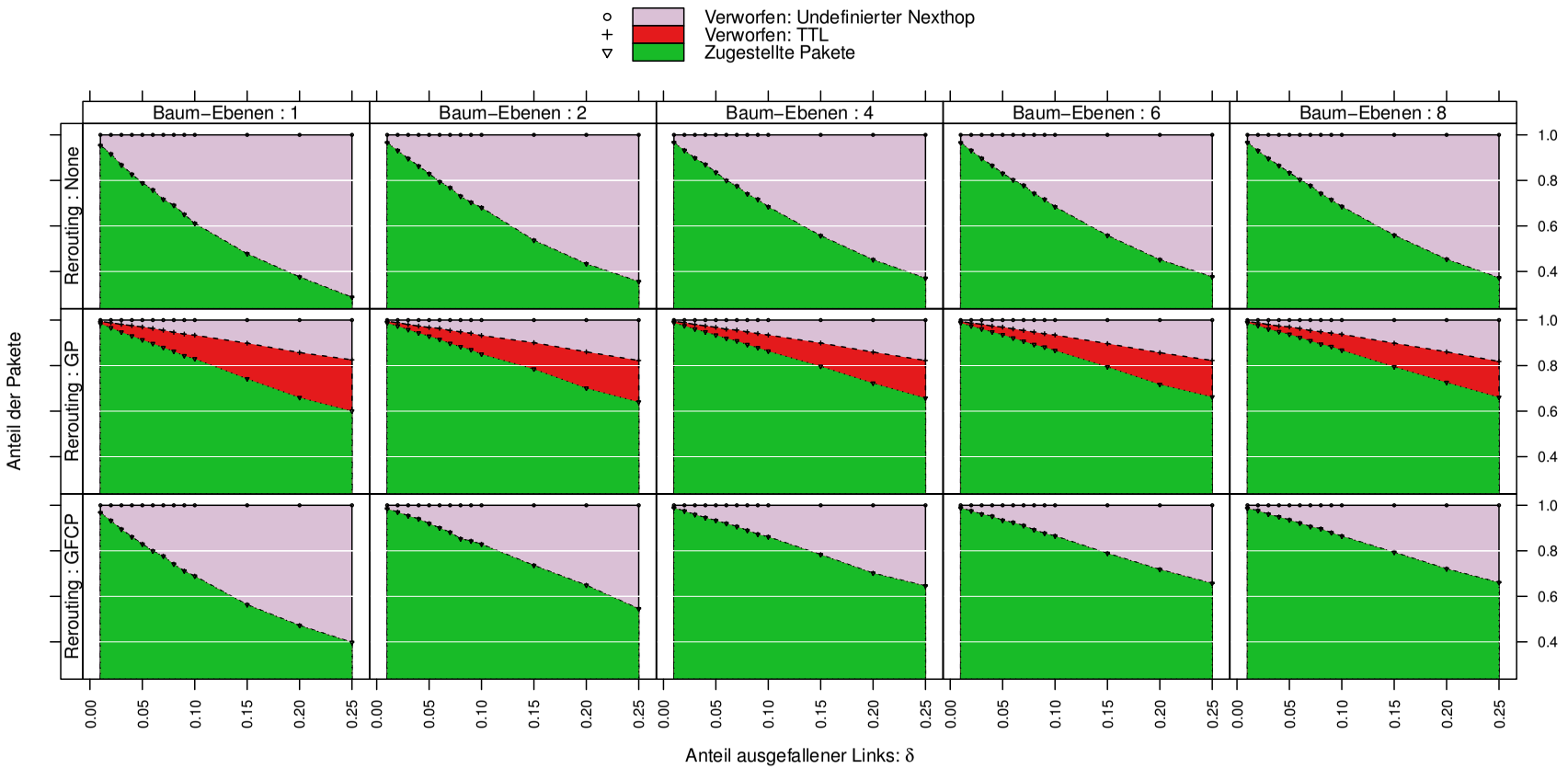


Abbildung 6.21: Anteile zugestellter und verworfener Pakete für die Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

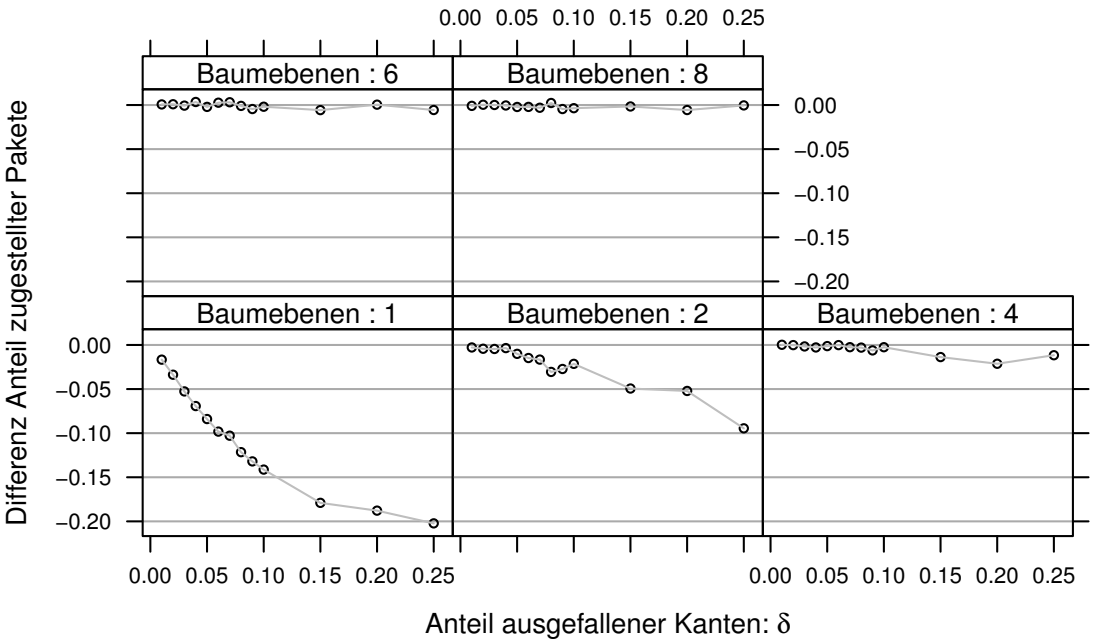


Abbildung 6.22: Differenz der Anteile zugestellter Pakete (GFCP − GP) für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

Für $\delta = 0.25$ werden beispielsweise statt 60% ($l = 1$) schließlich 66% ($l = 8$) der Pakete zugestellt.

Bei **Greedy Failure-Carrying Packets** (dritte Zeile) hängt der Anteil zugestellter Pakete wieder stark von der Anzahl der Baum-Ebenen ab. Durch die mit zunehmenden Baum-Ebenen steigende Pfad-Diversität kann GFCP deutlich mehr Pakete zustellen. Bei nur einer Baum-Ebene und $\delta = 0.25$ können mit 39% etwa 11% mehr Pakete zugestellt werden als ohne Einsatz eines Rerouting-Verfahrens. Der Anteil zugestellter Pakete erhöht sich jedoch deutlich mit hinzukommenden Baum-Ebenen auf bis zu 66% ($l = 8$) während ohne Rerouting nur weniger als 40% der Pakete zugestellt werden.

Abbildung 6.22 stellt zum Vergleich zwischen den Rerouting-Verfahren GFCP und GP die Differenz des Anteils zugestellter Pakete zwischen beiden Verfahren dar. Auf der x-Achse ist dabei der Anteil ausgefallener Kanten abgetragen und auf der y-Achse die Differenz der beiden Zustellraten, wobei negative Werte bedeuten, dass GFCP weniger Pakete als GP zustellt. Dabei wird deutlich, dass GP-Rerouting insbesondere bei nur einem einzelnen Spannbaum sehr viel bessere Zustellraten erreicht. Während GFCP bei nur einer Baum-Ebene dem Gravity-Pressure Routing deutlich unterlegen ist, verringert sich der Abstand zwischen beiden mit weiteren hinzukommenden Baum-Ebenen und ist bereits bei $l \geq 6$ nicht mehr vorhanden bzw. fällt der Vergleich teilweise leicht zu Gunsten von GFCP aus.

Mit der für PIE für eine Topologie dieser Größe empfohlenen Anzahl von $\lfloor \log_2(26424) \rfloor = 7$ Baum-Ebenen arbeiten also sowohl GFCP als auch GP sehr gut und erreichen eine deutlich erhöhte Zustellrate gegenüber PIE ohne Rerouting.

Abbildung 6.21 stellt neben den zugestellten Paketen (grün) auch die verworfenen Pakete dar und unterteilt diese in zwei Kategorien: Die erste Kategorie (grau) umfasst Pakete, die verworfen wurden, weil nach dem Greedy-Prinzip kein gültiger Knoten für die Weiterleitung der Pakete existiert. Ohne Einsatz eines Rerouting-Verfahrens bedeutet dies, dass keiner der Nachbarn mehr einen Fortschritt in Richtung des Ziels ermöglicht. Für GFCP könnten sogar noch Nachbarn vorhanden sein, die bei gieriger Weiterleitung zwar noch einen Fortschritt in Richtung des Ziels bedeuten könnten, aber durch GFCP Fehlerbeschreibungen ausgeschlossen sind und das Paket dadurch verworfen wird. Die zweite Kategorie (rot) beinhaltet Pakete, für die weitere Weiterleitungsmöglichkeiten existiert hätten, die aber aufgrund einer Überschreitung der Time-To-Live verworfen werden mussten. Diese Kategorie tritt bei der Untersuchung von Kantenausfällen *ausschließlich bei Gravity-Pressure Rerouting auf* und macht dort zwischen 46% und 65% aller verworfenen Pakete aus. Diese Pakete verweilen über die volle Time-To-Live (hier 64 Übertragungsabschnitte) im Netz und werden dann unzugestellt verworfen. GFCP hingegen verwirft alle Pakete vor Ablauf der Time-To-Live durch Ausschluss ungeeigneter Weiterleitungsmöglichkeiten.

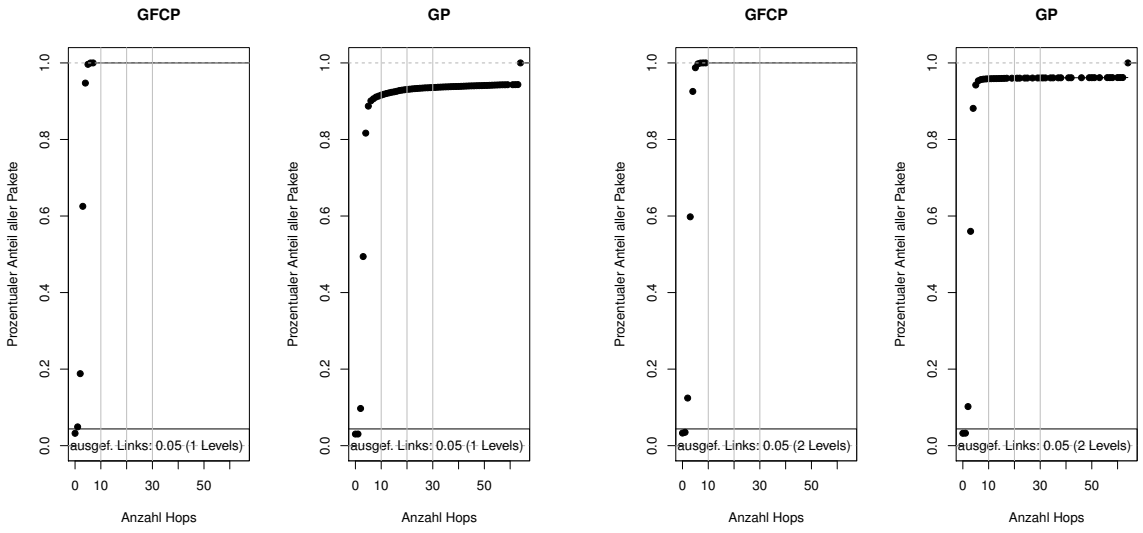
Um exemplarisch zu untersuchen, wann bei GP bzw. GFCP Pakete verworfen werden, zeigt Abbildung 6.23 bei 5% ausgefallenen Kanten die kumulierten relativen Häufigkeiten für die Distanz, nach der ein Paket entweder zugestellt oder verworfen wurde. Während für GFCP in allen hier dargestellten Fällen ($l \in \{1, 2, 4, 8\}$) kein Paket mehr als 13 Hops im Netz verweilt, werden bei Verwendung von Gravity-Pressure Routing 3% ($l = 8$) bis 5% ($l = 1$) aller Pakete erst durch Ablauf der TTL verworfen. Dies liegt daran, dass Gravity-Pressure Rerouting sehr aggressiv arbeitet und selbst über keinerlei Abbruchbedingung verfügt, wenn es ein Paket nicht zustellen kann (siehe Abschnitt 6.3.2.2). Dies ist eine große Schwäche von Gravity-Pressure Routing, die später bei der Betrachtung des Overhead einer genaueren Analyse unterzogen wird.

6.6.3.2 Stretch

Die im Vergleich hohen Paketzustellraten von GP, insbesondere bei nur wenigen Spannbaum-Ebenen, werden durch Inkaufnahme längerer Pfade ermöglicht. Abbildung 6.24 zeigt den durchschnittlichen Stretch (y-Achse), den zugestellte Pakete in Abhängigkeit vom Anteil ausgefallener Kanten (x-Achse) erfahren; verworfene Pakete werden durch den Stretch nicht erfasst.

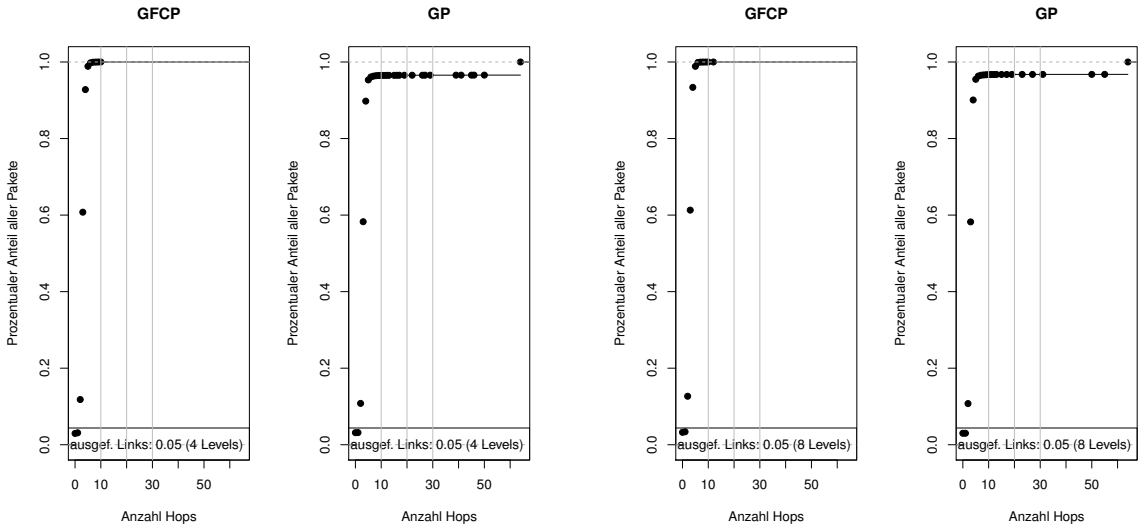
Für **PIE ohne Rerouting** ist der durchschnittliche Stretch dabei am niedrigsten, da Pakete an einem lokalen Minimum verworfen werden und kein zusätzlicher Aufwand für die Zustellung auf einem alternativen Weg ensteht. Pakete werden also strikt nach dem Greedy-Prinzip zugestellt oder an einem lokalen Minimum verworfen. Der Einsatz eines der beiden betrachteten Rerouting-Verfahren erhöht von diesem Niveau aus den Stretch. Dies resultiert daher, dass Pakete, die bei PIE verworfen werden und damit im Stretch nicht erfasst werden, durch den Einsatz der Rerouting-Verfahren auf Umwegen zum Ziel kommen können und dann im Stretch erfasst werden.

Betrachtet man das Verhalten von **Gravity-Pressure Routing** über die verschiedenen Baum-Ebenen, so ist zu beobachten, dass insbesondere bei nur einer Baum-Ebene der durchschnittliche Stretch mit zunehmenden Kantenausfällen ansteigt. Die Wahrscheinlichkeit, dass GP



(a) 1 Spannbaum-Ebene

(b) 2 Spannbaum-Ebenen



(c) 4 Spannbaum-Ebenen

(d) 8 Spannbaum-Ebenen

Abbildung 6.23: Anzahl zurückgelegter Hops aller Pakete (zugestellt und verworfen) bei 5% ausgefallenen Links

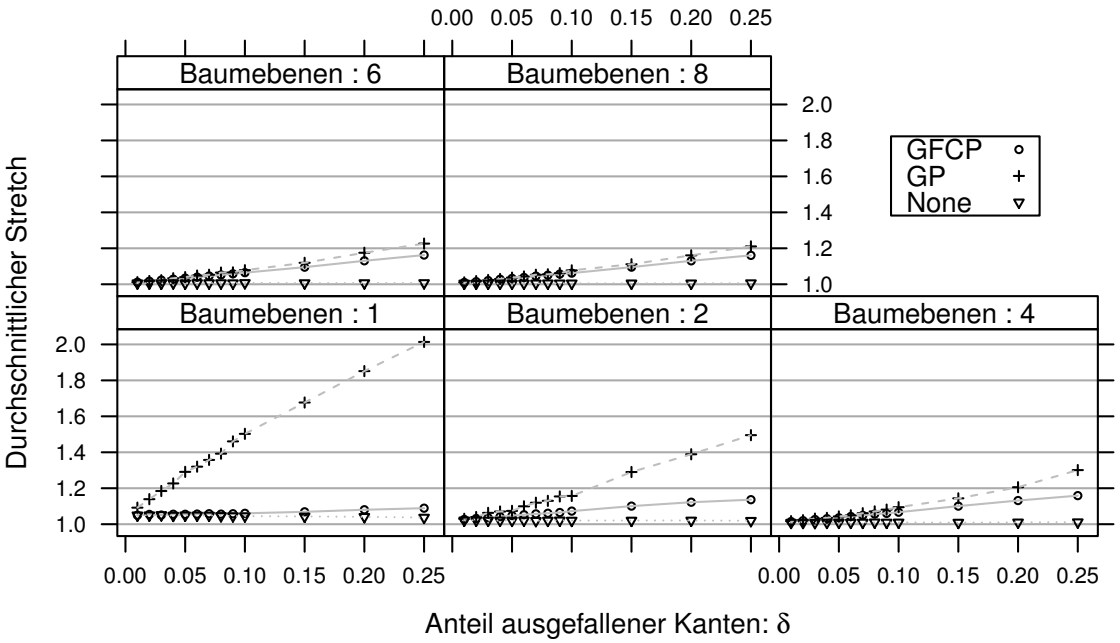


Abbildung 6.24: Durchschnittlicher Stretch für die Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

Rerouting schnell wieder auf einen funktionierenden Pfad zurückfindet und das Paket nach dem Greedy-Prinzip zugestellt werden kann, ist für $l = 1$ gering und so werden Pakete eher im ineffizienten Pressure-Modus weitergeleitet. Erhöht sich die Anzahl der für die Paketweiterleitung zur Verfügung stehenden Baum-Ebenen, so reduziert sich auch der durchschnittliche Stretch.

Während **Greedy Failure-Carrying Packets** in allen Fällen einen niedrigeren durchschnittlichen Stretch als GP Rerouting erreicht, so wird dies insbesondere bei nur wenigen Baum-Ebenen durch eine deutliche Verschlechterung der Zustellrate erreicht. Für die Konfigurationen mit $l = 1$, in denen GFCP eine nur leichte Verbesserung gegenüber der Verwendung von PIE ohne Rerouting-Mechanismus erreicht, resultiert auch nur ein minimal erhöhter Stretch gegenüber diesem. Wenn durch mehr Baum-Ebenen auch eine größere Pfaddiversität zur Verfügung steht, erhöht sich durch umfangreichere Umleitungsmöglichkeiten auch der Stretch von GFCP während der von GP parallel sinkt, da Pakete seltener im ineffizienten Pressure-Modus weitergeleitet werden müssen. Für Konfigurationen mit $l \geq 6$, in denen GFCP und GP gleiche Zustellraten erreichen, nähert sich der durchschnittliche Stretch der beiden Verfahren an und ein Vergleich fällt leicht zu Gunsten von GFCP aus.

Betrachtet man hingegen den in Abbildung 6.25 dargestellten maximalen Stretch, so fällt das Urteil hier deutlich zu Gunsten von GFCP aus, das durch seine zielgerichtete Weiterleitung von Paketen in allen Fällen einen maximalen multiplikativen Stretch von weniger als fünf erreicht. Der maximale Stretch von Gravity-Pressure Routing hingegen liegt in fast allen Fällen über zehn.

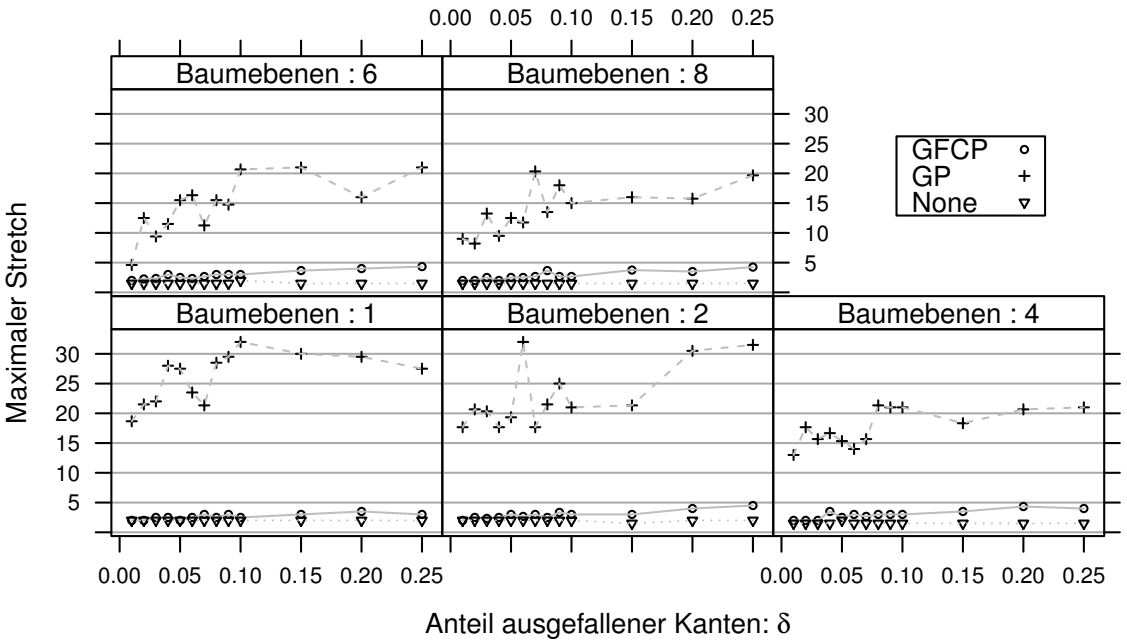


Abbildung 6.25: Maximaler Stretch für die Rerouting-Verfahren GFCP, GP und ohne Rerouting (None) für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

6.6.3.3 Mehraufwand

Um die, im Vergleich zu PIE ohne Rerouting, höheren Zustellraten zu erreichen, müssen Rerouting-Strategien einen erhöhten Aufwand in Kauf nehmen. Da sowohl GFCP als auch GP Pakete zunächst nach dem Greedy-Prinzip weiterleiten bis diese auf ein lokales Minimum treffen und von dort aus dann einen alternativen Pfad suchen, verlängern sich natürlich die Pfade gegenüber PIE ohne Rerouting-Strategie. In diesem Abschnitt wird der Mehraufwand durch die beiden Rerouting-Strategien im Detail untersucht. Dazu wird zunächst die Mehrbelastung des Netzes durch die Inkaufnahme von Umwegen betrachtet. Abschließend wird die Größe der im Paketkopf enthaltenen Fehlerbeschreibungen untersucht und darüber die Praxistauglichkeit von GFCP bewertet.

Als erste Metrik wird die **Belastung des Netzes** definiert, die eine gemeinsame Betrachtung von zugestellten sowie verworfenen Paketen erlaubt. Dazu werden die zurückgelegten Hop-Distanzen aller Pakete eines Rerouting-Verfahrens aufsummiert und anschließend mit zurückgelegten Hop-Distanzen von PIE ohne Rerouting normiert; für letzteren Fall ist die Netzbelastung damit immer 1 während die Netzbelastung für Reroutingverfahren größer gleich 1 ist. Abbildung 6.26 stellt die Netzbelastung durch GP und GFCP aufgetrennt nach Baum-Ebenen dar. Die Netzbelastung durch GP liegt dabei in allen Fällen über der von GFCP. Während dies insbesondere für $l \leq 4$ noch mit einer höheren Zustellrate von GP zu begründen ist, so ist der Aufwand von GP auch bei $l \geq 6$ deutlich höher als der von GFCP, während beide vergleichbare Paketzustellraten erreichen. Da der Mehraufwand auf der Seite von GP

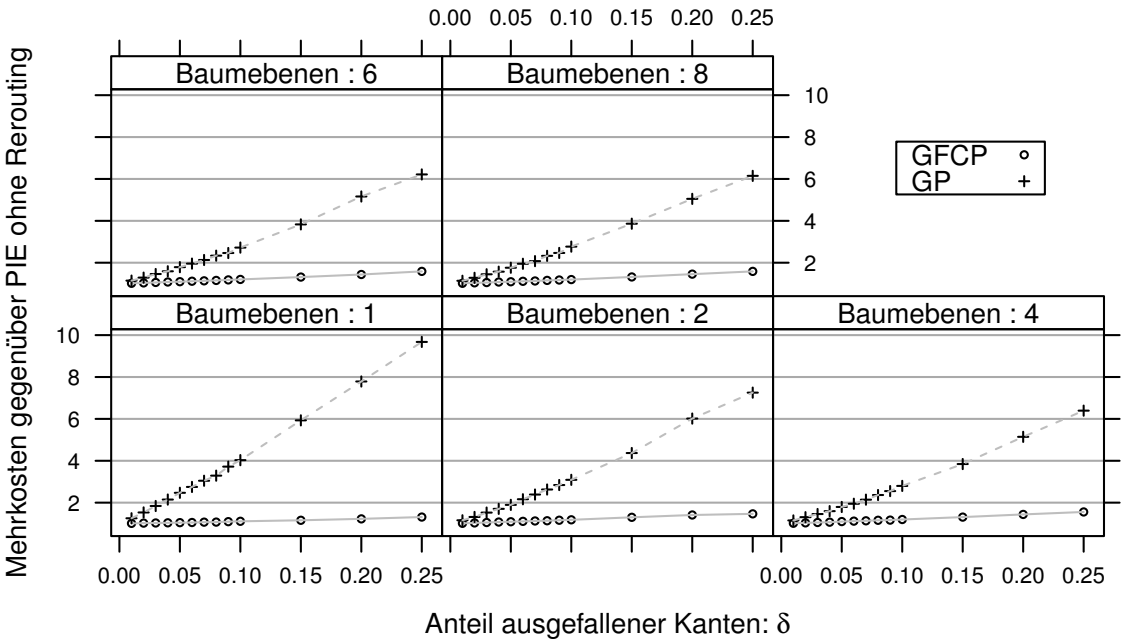


Abbildung 6.26: Netzbelastung für die Rerouting-Verfahren GP und GFCP gegenüber PIE ohne Rerouting für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

zu einem nicht unerheblichen Teil durch die Pakete verursacht wird, die aufgrund einer Überschreitung der Time-To-Live verworfen werden, wurde in einem weiteren Experiment noch untersucht, wie sich GP bei deutlich geringeren TTL verhält.

Da GFCP in keinem Fall über 13 Hops benötigte, um ein Paket zuzustellen oder zu verwerfen, wurde die TTL zu Gunsten von GP auf 13 gesetzt und die Veränderung im Mehraufwand von GP sowie dessen Zustellrate betrachtet. Den resultierenden Mehraufwand stellt Abbildung 6.27 dar. Trotz einer praktisch nicht wünschenswerten, topologie-abhängigen Anpassung der Time-To-Live ist die Netzbelastung durch GP immer noch höher als durch GFCP. Zugleich resultiert die Senkung der TTL in einer reduzierten Zustellrate; diese ist in Abbildung 6.28 dargestellt. Während also die Netzbelastung durch GP weiterhin signifikant höher ist als die durch GFCP verbessert sich insbesondere für $l \geq 6$ die Zustellrate zum Vorteil von GFCP (vergleiche Abbildung 6.22)

Als zweite Metrik wird der **im Paketkopf notwendige Speicheraufwand** der GFCP Fehlerbeschreibungen untersucht. In den Betrachtungen zur effizienten Kodierung von Fehlerbeschreibungen in Paketen wurde bereits die mittlere (5.5 Bytes) und maximale Länge (9.75 Bytes) einer einzelnen Fehlerbeschreibung für die DIMES Topologie bestimmt (siehe Abschnitt 6.5.2).

An dieser Stelle wird daher die Anzahl der im Paketkopf eingetragenen Fehlerbeschreibungen untersucht, anhand derer sich in Kombination mit dem zuvor ermittelten Speicherbedarf für einzelne Fehlerbeschreibungen der gesamte Speicherbedarf im Paketkopf erschließt.

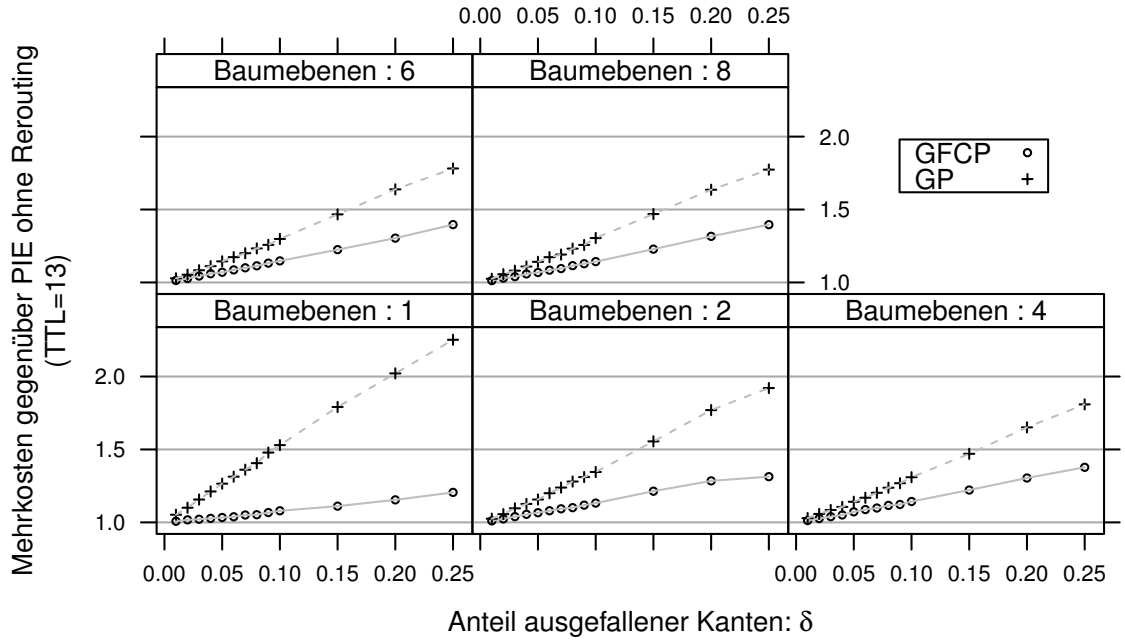


Abbildung 6.27: Mehraufwand für die Rerouting-Verfahren GP und GFCP bei TTL=13 gegenüber PIE ohne Rerouting für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

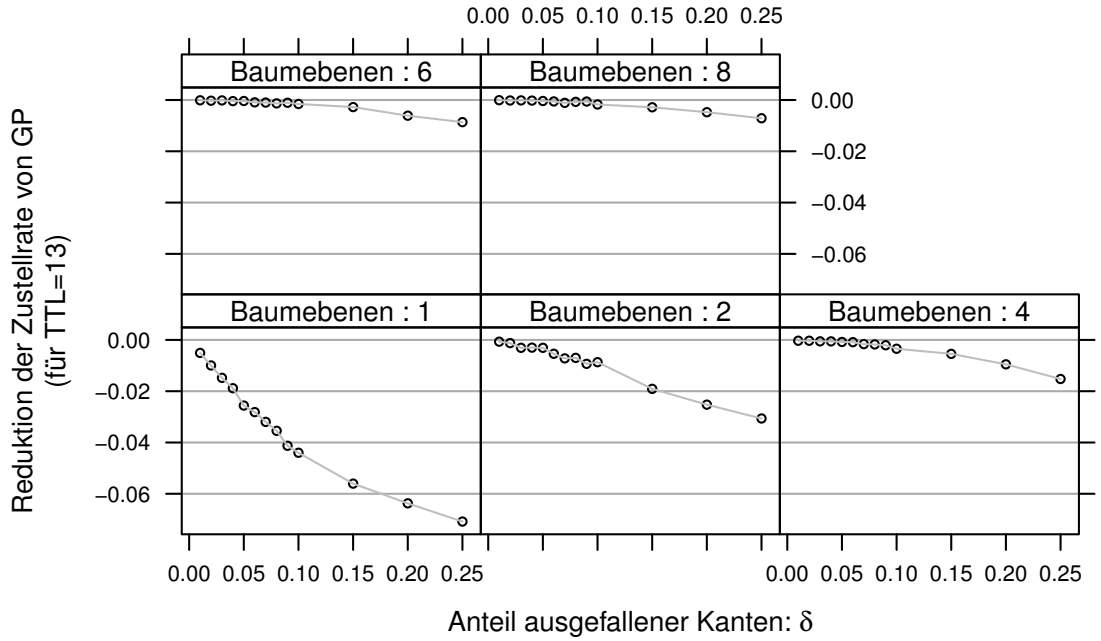


Abbildung 6.28: Reduktion der Zustellrate von GP Rerouting bei TTL=13 für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen.

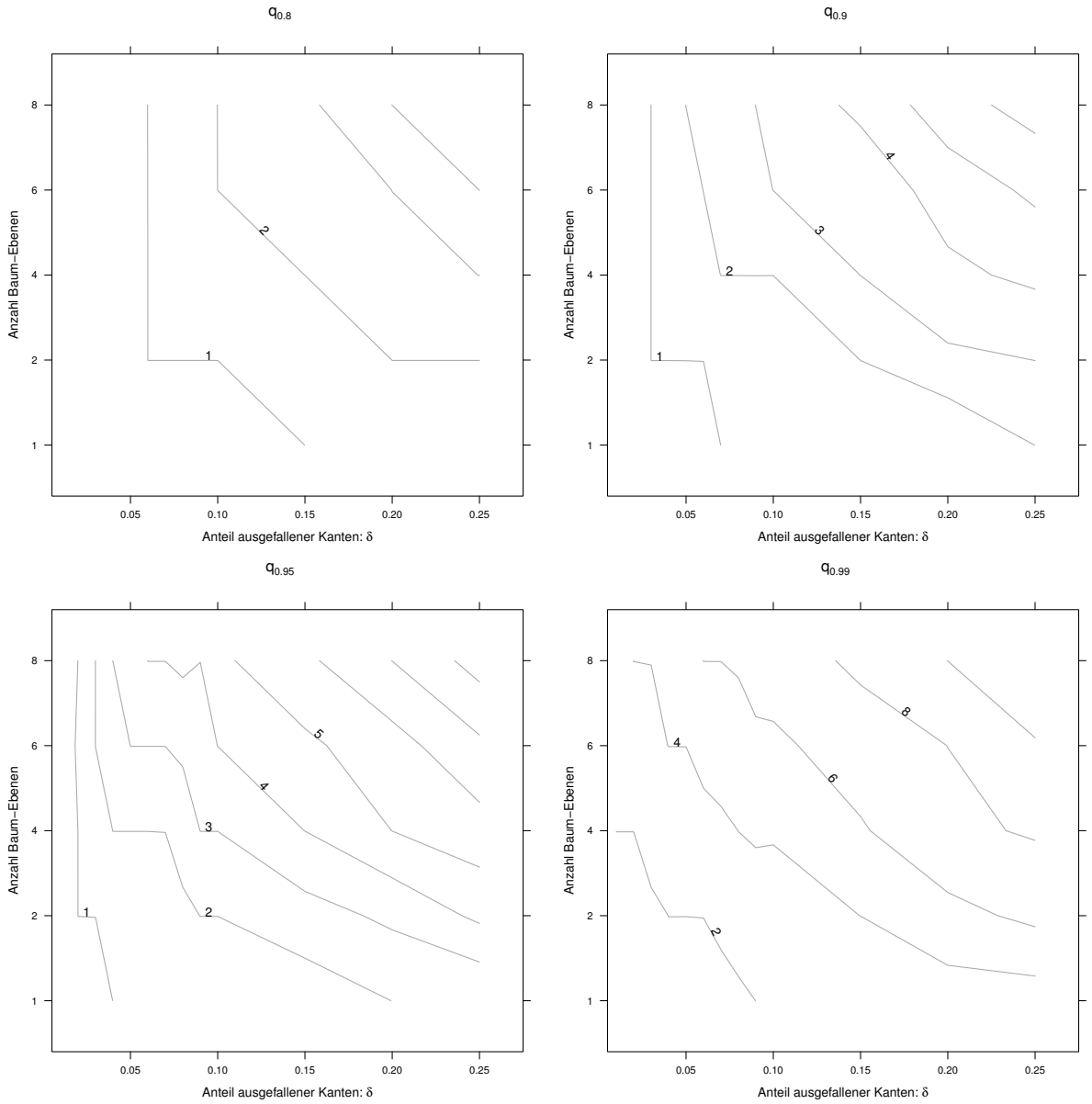


Abbildung 6.29: Quantile $q_{0.80}$, $q_{0.90}$, $q_{0.95}$, $q_{0.99}$ für die Anzahl von GFCP-Fehlerbeschreibungen im Paketkopf für Kantenausfälle von $\delta = \{0.01, ..., 0.25\}$ sowie $l = \{1, 2, 4, 6, 8\}$ Baumebenen

Abbildung 6.29 stellt die Quantile aus Tabelle 6.5 in einem Konturplot dar und bietet eine Übersicht über die Ergebnisse. Anhand der Abbildung für jedes der Quantile $q_{0.80}$, $q_{0.90}$, $q_{0.95}$ und $q_{0.99}$ ablesbar, wieviele Fehlerbeschreibungen für ein gegebenes l und δ zu erwarten sind.

Die detaillierten Ergebnisse für die durchgeführten Versuche stellt Tabelle 6.5 dar. Diese gibt für die untersuchten Konfigurationen von GFCP in Abhängigkeit von der Anzahl Baum-Ebenen l und dem Anteil ausgefallener Kanten δ die 0.80-, 0.90-, 0.95- und 0.99-Quantile sowie die maximal beobachtete Anzahl von Fehlerbeschreibungen in einem Paket an.

Für 95% aller Pakete in allen Konfigurationen sind Fehlerbeschreibungen im Mittel kleiner als $8 \cdot 5.5\text{Bytes} = 44\text{Bytes}$. Im schlechtesten Fall, der für $l = 6$ und $\delta = 0.20$ eintritt, ergibt sich unter der unwahrscheinlichen Annahme, dass alle Fehlerbeschreibungen maximale Länge hätten, eine Länge von $29 \cdot 9.75\text{Bytes} = 282.75\text{Bytes}$ für die Fehlerbeschreibung. Bei dieser Abschätzung ist aber zu bedenken, dass ein Anteil von $\delta = 0.20$ an ausgefallenen Kanten ein sehr extremer Fall ist. Nimmt man für die Praxis geringere Linkausfallraten von beispielsweise bis zu maximal 10% an, so ergibt sich für die beim Einsatz von GFCP interessanten Konfigurationen mit $l \in \{6, 8\}$ im schlechtesten Fall eine maximale Länge der Fehlerbeschreibungen von $11 \cdot 9.75\text{Bytes} = 107.25\text{Bytes}$ ($l = 6$) beziehungsweise $18 \cdot 9.75\text{Bytes} = 175.5\text{Bytes}$. Im Mittel resultieren Fehlerbeschreibungen mit $11 \cdot 5.5\text{Bytes} = 60.5\text{Bytes}$ ($l = 6$) beziehungsweise $18 \cdot 5.5\text{Bytes} = 99\text{Bytes}$. Für 99% der Pakete ist die Länge der Fehlerbeschreibungen aber in diesen Fällen selbst bei der maximalen Länge einer einzelnen Fehlerbeschreibung von 9.75 Bytes kleiner gleich $6 \cdot 9.75\text{Bytes} = 58.5\text{Bytes}$ ($l = 6$) beziehungsweise $7 \cdot 9.75\text{Bytes} = 68.25\text{Bytes}$. Unter Verwendung der mittleren Länge einer einzelnen Fehlerbeschreibung von 5.5 Bytes ist die Größe der Fehlerbeschreibung eines Paketes für das 0.99-Quantil und $\delta \leq 0.10$ sogar kleiner gleich 33 Bytes ($l = 6$) beziehungsweise 38.5 Bytes ($l = 8$) und damit zur Kodierung in Paketköpfen geeignet.

6.7 Zusammenfassung

Das im Rahmen dieser Arbeit entwickelte Greedy Failure-Carrying Packets ist ein neuartiger Rerouting-Ansatz für spannbaumbasierte Routingprotokolle, die Pakete nach dem Greedy-Prinzip weiterleiten. Viele neuartige, skalierbare Routing-Protokolle arbeiten auf dieser Grundlage und besitzen bisher oft keine ausreichende Unterstützung zur Behandlung von Topologie-Änderungen. Mit Greedy Failure-Carrying Packets können solche bestehenden Routing-Protokolle um ein Rerouting-Verfahren ergänzt werden, da GFCP unabhängig von der konkreten gierigen Einbettung anwendbar ist.

Vor dem Hintergrund des hier entwickelten Greedy Failure-Carrying Packets werden abschließend die zu Beginn des Kapitels eingeführten Fragestellungen wieder aufgegriffen und beantwortet.

Tabelle 6.5: Quantile Anzahl Fehlerbeschreibungen GFCP (unaggregiert)

l	δ	q_{80}	q_{90}	q_{95}	q_{99}	Maximum
1	0.01	0	0	0	1	2
1	0.05	0	0	1	1	3
1	0.10	0	1	1	2	5
1	0.15	1	1	1	2	5
1	0.20	1	1	2	3	6
1	0.25	1	2	2	3	7
2	0.01	0	0	0	1	4
2	0.05	0	1	1	2	5
2	0.10	1	1	2	3	9
2	0.15	1	2	3	4	9
2	0.20	2	3	3	5	10
2	0.25	2	3	4	6	14
4	0.01	0	0	0	2	5
4	0.05	0	1	2	3	7
4	0.10	1	2	3	4	9
4	0.15	2	3	4	6	12
4	0.20	3	4	5	8	21
4	0.25	3	4	6	8	18
6	0.01	0	0	0	3	6
6	0.05	1	2	3	4	8
6	0.10	2	3	4	6	11
6	0.15	2	3	5	7	15
6	0.20	3	4	6	8	29
6	0.25	4	5	7	10	21
8	0.01	0	0	0	3	7
8	0.05	1	2	3	5	12
8	0.10	2	3	5	7	18
8	0.15	3	4	6	8	24
8	0.20	4	6	7	10	22
8	0.25	4	6	8	12	25

R | Wie können spannbaum-basierte gierige Routing-Protokolle zur Vermeidung einer aufwändigen Reparatur robuster gestaltet werden, so dass die Auswirkungen von kurzlebigen Ausfällen auf die Qualität des Routings, gemessen durch die Anzahl zustellbarer Pakete, möglichst gering sind?

GFCP arbeitet ohne eine Anpassung der vorhandenen Routing-Tabellen und vermeidet den sehr hohen Aufwand für eine Aktualisierung der Routing-Tabellen, der sonst anfällt. Dazu kodiert GFCP Beschreibungen von Fehlern, die ein Paket unterwegs angetroffen hat, im Paketkopf. Diese Fehlerbeschreibungen werden von GFCP in der Weiterleitungsentscheidung genutzt, um Pfade, die nicht mehr zum Ziel führen können, auszuschließen und Pakete über eventuell längere, aber gültige Pfade zum Ziel zu leiten.

In der Evaluierung auf der DIMES Topologie konnte in einer Gegenüberstellung mit der alternativen Rerouting-Strategie Gravity-Pressure Routing gezeigt werden, dass GFCP ab 6 Baum-Ebenen eine vergleichbar gute Zustellrate erreicht. Beide Verfahren erhöhen insbesondere bei Kantenausfällen die Zustellrate deutlich und erzielen damit eine deutliche Verbesserung der Robustheit.

R-1 | Was sind die Kosten eines solchen Verfahrens?

Zunächst entsteht durch GFCP nur dann ein Mehraufwand gegenüber dem zugrundeliegenden Routing-Protokoll, wenn auch tatsächlich Fehler aufgetreten sind. Dieser Mehraufwand entsteht einerseits in Form der im Paket mitzuführenden Fehlerbeschreibungen und andererseits in Form einer Verlängerung der Pfade durch die Umleitung der Pakete.

Für GFCP wurde zunächst am Beispiel von PIE eine sehr effiziente Kodierung für Fehlerbeschreibungen entwickelt. Mit dieser Kodierung kann eine ausgefallene Kante der DIMES Topologie im Mittel mit 5.5 Bytes beschrieben werden und benötigt maximal 9.75 Bytes. Für bis zu 10% ausgefallener Kanten ergibt sich über alle Pakete eine maximale Länge einer Fehlerbeschreibung von 175.5 Bytes, während für 99% der Pakete maximal 68.25 Bytes benötigt werden. Während der Aufwand zur Kodierung der Fehlerbeschreibung in diesen Extremfällen hoch ist, betrifft dies nur einen sehr kleinen Teil aller Pakete.

Durch die präzise Fehlerbeschreibung kann GFCP Pakete sehr effizient auf kurzen Wegen zustellen und erzeugt damit nur einen sehr geringen Stretch. Zudem kann es Pakete frühzeitig verwerfen, für die kein gültiger Pfad mehr sichtbar ist. Dadurch belastet GFCP das Netz deutlich weniger als das zum Vergleich herangezogene Gravity Pressure Routing , während es vergleichbar gute Zustellraten erreicht.

R-2 | Inwieweit ist ein solches Verfahren praktisch einsetzbar?

GFCP ist als Erweiterung konzipiert und kann mit jedem spannbaum-basierten gierigen Routing-Protokoll genutzt werden. In allen Szenarien, wie beispielsweise virtuelle Netzen, in denen solche neuartigen skalierbaren Routing-Protokolle eingesetzt werden können, ist

damit auch GFCP einsetzbar. Durch die erreichte Erhöhung der Robustheit ist Greedy Failure-Carrying Packets eine gute Erweiterung für solche Protokolle und erhöht deren praktische Einsatzfähigkeit.

KAPITEL 7

Zusammenfassung und Ausblick

Das enorme Wachstum des Internets in den letzten zwanzig Jahren ging einher mit einem starken Wachstum der Routing-Tabellen. Wie auch das Internet Architecture Board feststellt, ist die Skalierbarkeit des Routings ein zentrales Problem, dem sich das heutige Internet stellen muss [1]. Vor diesem Hintergrund wurden im Rahmen dieser Arbeit anhand zweier zentraler Fragestellungen neuartige Ansätze für skalierbares und robustes Routing auf Internet-ähnlichen Topologien entwickelt. Als Vorarbeiten für die hier entworfenen Ansätze wurde mit einer **Architektur für Netzvirtualisierung** ein mögliches künftiges Einsatz-Szenario für diese vorgestellt. Zur Untersuchung der Ansätze wurde basierend auf OMNeT++ das skalierbare Rahmenwerk **RoutingSim** entwickelt, das eine Untersuchung von Routing-Protokollen unter Dynamik mit Fokus auf deren algorithmisches Verhalten ermöglicht.

7.1 Beiträge dieser Arbeit

Mit dem BC Routing-Verfahren wurde ein Ansatz aus dem Bereich des kompakten Routings identifiziert, der bei skalierbaren Routing-Tabellen eine parametrisierbare additive Beschränkung des Stretch auf Internet-ähnlichen Topologien erlaubt. Das BC Routing-Verfahren setzt aber eine zentralisierte Entität voraus, die über eine vollständige Sicht auf die Topologie verfügt, und ist damit nicht für den Einsatz in einem verteilten Routing-Protokoll vorgesehen. Diese Beobachtung führte zu der ersten übergeordneten Fragestellung, die in dieser Arbeit behandelt wurde:

V	Kann das zentralisierte Routing-Verfahren von Brady und Cowen als verteiltes Routing-Protokoll umgesetzt werden?

Mit **Sprinkles** wurde ein Routing-Protokoll entwickelt, das die parametrisierbare additive Stretch-Schranke des BC Routing-Verfahrens verteilt realisiert. Sprinkles konstruiert die

dafür erforderlichen Spannbäume mit ihren Abhängigkeiten und nimmt die notwendige Partitionierung der Topologie in Kern und Rand vor. Zugunsten einer verteilten Bestimmung der lokationsabhängigen Adressen wurde bei Sprinkles auf die im zentralisierten Fall mögliche Beschränkung der Adresslänge verzichtet und stattdessen ein geeignetes verteiltes Verfahren verwendet. Dieses erreicht in der Praxis ebenfalls polylogarithmische Adresslängen.

Mit dem neu eingeführten Sparse Mode wurde gemäß des Prinzips „Trennung der Belange“ eine klare Unterscheidung zwischen für die Einhaltung der Stretch-Schranke notwendigen Spannbäumen und optionalen, zur Reduktion des durchschnittlichen Stretchs einsetzbaren Spannbäumen vorgenommen. Durch den Sparse Mode kann die Anzahl der erforderlichen Spannbäume im Vergleich zum BC Routing-Verfahren deutlich reduziert werden. Dies führt zu einer Reduktion der Adresslänge und der Routing-Tabellengröße, während aber theoretisch wie auch empirisch gezeigt wurde, dass die Stretch-Schranke auch im Sparse Mode stets eingehalten wird. Wie sich bei der Evaluierung auf der CAIDA Topologie mit fast zweihunderttausend Knoten zeigte, werden durch die im Sparse Mode reduzierten Adresslängen deutlich bessere Stretch-Schranken praktisch einsetzbar als im BC-ähnlichen Dense Mode von Sprinkles.

Ausgehend von Sprinkles wurde eine Beobachtung gemacht, die so auch für die gesamte Klasse spannbaum-basierter gieriger Routing-Protokolle gilt. Veränderungen der Topologie durch Knoten- oder Kantenausfälle können durch eine Partitionierung der zugrundeliegenden Spannbaum-Strukturen zu einer schwerwiegenden Einschränkung der Erreichbarkeit der Knoten führen. Durch die von der Lage im Spannbaum abhängig vergebenen Adressen erfordert eine Reparatur der Routing-Tabellen nach einem Ausfall einen hohen Aufwand. Davon ausgehend wurde die zweite übergeordnete Fragestellung dieser Arbeit formuliert:

R Wie können spannbaum-basierte gierige Routing-Protokolle zur Vermeidung einer aufwändigen Reparatur robuster gestaltet werden, so dass die Auswirkungen von kurzlebigen Ausfällen auf die Qualität des Routings, gemessen durch die Anzahl zustellbarer Pakete, möglichst gering sind?

Das im Rahmen dieser Arbeit entwickelte **Greedy Failure-Carrying Packets** ist eine Rerouting-Strategie, die als Erweiterung für spannbaum-basierte gierige Routing-Protokolle einsetzbar ist. Anstatt die aufwändige Reparatur in Kauf zu nehmen, werden die Routing-Tabellen nach Ausfällen bewusst nicht angepasst. Stattdessen wird die Robustheit dieser Routing-Protokolle gegenüber Ausfällen dadurch erhöht, dass eine Beschreibung von Fehlerstellen, die ein Paket während seiner Weiterleitung antrifft, im Paketkopf kodiert wird. Damit werden in nachfolgenden Weiterleitungsentscheidungen ungültige Pfade ausgeschlossen.

In einer ausführlichen Evaluierung wurde gezeigt, dass Greedy Failure-Carrying Packets eine vergleichbare Paketzustellrate wie das dem Stand der Technik entsprechende Gravity Pressure Routing erreicht und zu einer hohen Robustheit gegenüber Ausfällen führt. Der im Paketkopf durch Fehlerbeschreibungen verursachte Mehraufwand tritt dabei nur auf, falls tatsächlich Ausfälle auf dem Pfad des Pakets aufgetreten sind. Die Kodierung der Fehlerbeschreibungen im Paketkopf ist selbst auf großen Topologien möglich. Ein entscheidender Vorteil von Greedy Failure-Carrying Packets gegenüber dem Gravity Pressure Routing ist,

dass dieses die Netzbelastung deutlich reduziert. Die im Vergleich geringere Netzbelastung erreicht Greedy Failure-Carrying Packets dadurch, dass Pakete, die ihr Ziel erreichen, auf Alternativpfaden mit geringem Stretch zugestellt werden. Pakete, für die in der GFCP Weiterleitungsentscheidung kein gültiger Pfad mehr existiert, werden frühzeitig verworfen. Es wurde gezeigt, dass die Paketweiterleitung durch Greedy Failure-Carrying Packets durch den konsequenten Ausschluss von Teilbäumen durch Fehlerbeschreibungen schleifenfrei ist.

7.2 Ausblick auf weiterführende Arbeiten

- Sprinkles ermöglicht die verteilte Umsetzung einer additiven Beschränkung des Stretch basierend auf dem Kerndurchmesser d. Wie im Rahmen der hier durchgeführten Evaluierungen gezeigt wurde, ist selbst bei einer Reduktion der Adresslängen durch den Sparse Modus eine *gute Parametrisierung von d* unbedingt erforderlich. Wird d zu klein gewählt, resultieren Adresslängen, die nicht mehr in Paketköpfen verwendbar sind. Wird d hingegen zu hoch angesetzt, liegt die Stretch-Schranke höher als notwendig. Damit stellt sich die Frage, welche Informationen über eine Topologie im Voraus bekannt sein müssen, so dass die *Bestimmung eines guten Kerndurchmessers* möglich ist.
- Greedy Failure-Carrying Packets verursacht in der *Weiterleitungsentscheidung einen erhöhten Berechnungsaufwand*. Bei der Bestimmung eines Weiterleitungsknotens muss geprüft werden, dass keine der im Paket enthaltenen Fehlerbeschreibungen den dadurch gewählten Pfad betrifft. Eine Möglichkeit, den zeitlichen Aufwand zu reduzieren, ist eine Parallelisierung der Überprüfung. Es sind jedoch auch Ansätze denkbar, die, im Gegensatz zu der hier beschriebenen zustandslosen Form von GFCP, die in den Datenpaketen transportierten Fehlerbeschreibung zum Aufbau von Zustand auf den Knoten nutzen. Statt einer Neuberechnung der GFCP Weiterleitungsfunktion pro Paket kann beispielsweise dem für die Weiterleitung relevanten Tupel aus Empfänger und Fehlerbeschreibungen über eine Hash-Tabelle der dann nur noch einmalig zu bestimmende Weiterleitungsknoten zugeordnet werden. Neben möglichen Kollisionen ist dabei auch zu bedenken, wie lange solche Einträge bestehen bleiben sollen.
- Wie bereits in Abschnitt 6.5.1 angedeutet, können weitere Verfahren zur *Reduktion der erforderlichen Fehlerbeschreibungen* untersucht werden. Da Fehlerbeschreibungen bei GFCP jeweils Teilbäume beschreiben, ist es möglich, dass ein Teilbaum einen anderen enthält und dadurch nur die umfassendere Fehlerbeschreibung im Paket kodiert werden muss. Davon würde der Speicheraufwand im Paket und auch die Berechnung der Weiterleitungsentscheidung profitieren. Diese Aggregation ist zudem verlustlos, da keine für die Weiterleitung relevante Information verlorengeht.

  Eine alternative Möglichkeit besteht im Festlegen einer maximalen Anzahl von Fehlerbeschreibungen, die in einem Paket kodiert werden dürfen. Soll einem Paket mit bereits maximaler Anzahl von Fehlerbeschreibungen nun eine zusätzliche Fehlerbeschreibung hinzugefügt werden, gehen in diesem Fall Informationen verloren und die Entstehung von Schleifen wäre theoretisch möglich. Hier ist also die Untersuchung von geeigneten Verdrängungsstrategien für die Fehlerbeschreibungen nötig, anhand

derer entschieden werden kann, welches die wichtigsten Fehlerbeschreibungen sind und welche verworfen werden sollen.

ANHANG A

Greedy Failure-Carrying Packets

A.1 Einfluss von Topologieänderungen

Um zu motivieren, wieso GFCP einen gewissen Aufwand in Kauf nimmt, um eine Reparatur der Routing-Tabellen zu verhindern bzw. zu verzögern, wird hier der aus Topologieänderungen resultierende Aufwand zur Aktualisierung der Routing-Tabellen analysisiert. Dabei wird zunächst eine Erweiterung des Netzes durch neu hinzukommende Knoten oder Kanten betrachtet und anschließend die Auswirkungen eines Ausfalls von Kanten oder Knoten untersucht.

A.1.1 Hinzufügen eines Links

Wird ein neuer Link zwischen zwei Routern hinzugefügt, so hat dies zunächst keine negativen Auswirkungen auf die gierige Einbettung. Es können immer noch alle Pakete entlang der vorhandenen Spannbäume zugestellt werden. Die neue Kante kann sogar sofort, nachdem die beiden Knoten ihre Adressen ausgetauscht haben, für die Weiterleitung von Paketen genutzt werden; zunächst aber nur als Abkürzung. Dies wird auf Grundlage der Topologie aus Abbildung 2.8 erläutert, die nun über eine zusätzliche Kante e_{neu} zwischen Knoten w und Knoten t verfügt. Abbildung A.1 zeigt den Spannbaum T_1 mit der zusätzlichen Kante, die nun von Knoten w und Knoten t als Abkürzung genutzt werden kann. Ein Paket von Knoten t nach u wird nun nicht mehr entlang des Spannbaums weitergeleitet (Distanz 3) sondern über den kürzeren Weg über e_{neu} zugestellt (Distanz 2). Da die neu hinzugekommene Kante nur verwendet wird, wenn einer der angrenzenden Knoten feststellt, dass eine Weiterleitung über die neue Kante eine Abkürzung in Richtung des Ziels gegenüber einer Weiterleitung entlang des Spannbaumes darstellt, können dadurch auch keine Schleifen entstehen. Diese einfache Integration einer neuen Kante ist zunächst vorteilhaft. Eine neue Kante kann aber gleichzeitig dazu führen, dass nicht mehr alle Knoten im Spannbaum die minimale Distanz zur Wurzel

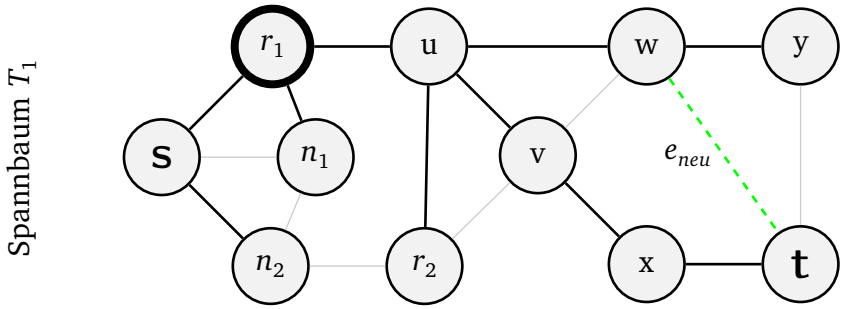


Abbildung A.1: Auswirkungen einer neu hinzukommenden Kante (vor Anpassung)

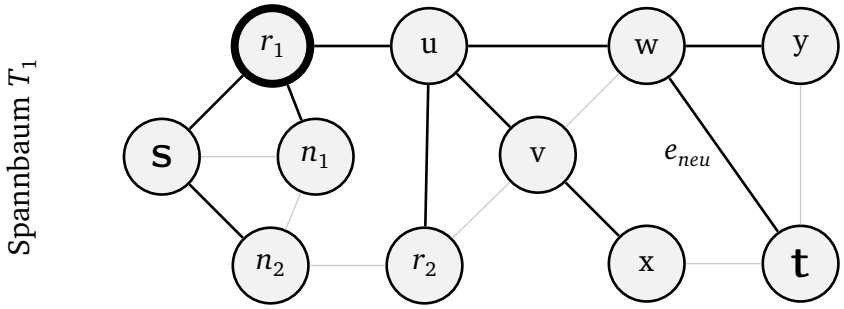


Abbildung A.2: Auswirkungen einer neu hinzukommenden Kante (nach Anpassung)

haben, wie Knoten t in dem vorliegenden Beispiel. Knoten t's kürzester Pfad zur Wurzel wäre nun statt zuvor r_1–u–v–x–t (Distanz 4) der Pfad r_1–u–w–t (Distanz 3). Ohne eine Anpassung der Routing-Tabellen erfahren beispielsweise Pakete von Knoten r_1 an Knoten t damit einen höheren Stretch als nach einer Anpassung der Routing-Tabellen bzw. des Spannbaums, wie in Abbildung A.2 dargestellt. Ändern sich durch die neu hinzugekommene Kante die Spannbäume nicht, ist keinerlei Anpassung notwendig.

A.1.2 Hinzufügen eines Routers

Tritt ein neuer Knoten dem Netz bei, verfügt dieser über Verbindungen zu einem oder mehreren Knoten des bestehenden Netzes. Der Unterschied zu einer neu hinzukommenden Kante ist jedoch, dass dieser Knoten noch keine Adresse in dem Netz besitzt und damit nicht erreichbar ist. Um diesen Knoten in ein bestehendes Netz zu integrieren gibt es mehrere Möglichkeiten. Der Neuaufbau aller Routingstrukturen ist auch hier der einzige Weg, um in allen Fällen wieder bestmöglichen Stretch zu erreichen, da nur so gewährleistet ist, dass jeweils die kürzesten Wege zu den Wurzeln der Spannbäume gewählt werden und so kein unnötiger Stretch durch nicht angepasste Routingstrukturen entstehen kann. Da ein Neuaufbau aller Routingstrukturen aufwändig ist, sind hier Lösungen zu bevorzugen, die diesen Aufwand vermeiden oder verzögern. Wie bereits in Abschnitt 6.1 erwähnt, geht GFCP davon aus, dass eine Anpassung an langfristige Änderungen der Topologie durch einen periodischen Neuaufbau der Routingstrukturen geschieht. Eine Strategie, um neue Knoten bereits vor dem nächsten periodischen Neuaufbau erreichbar zu machen (fast join), ist es, diesen abhängig von Ihren Nachbarn unter rein lokaler Kommunikation eine temporäre Adresse

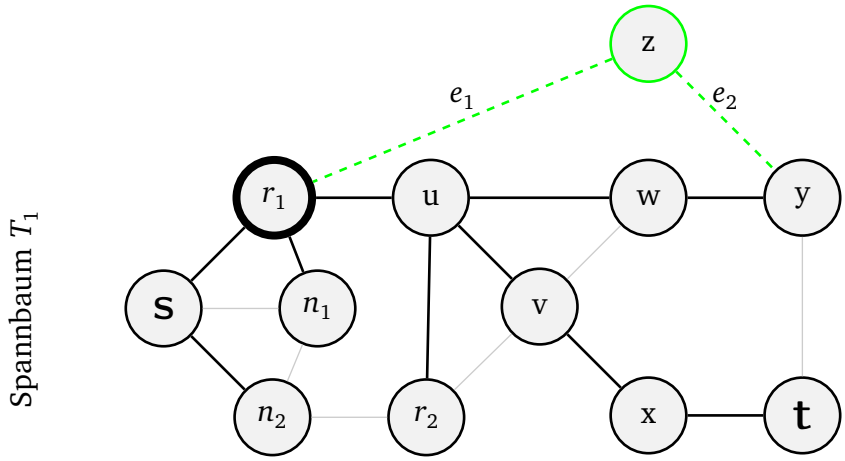


Abbildung A.3: Auswirkungen eines neu hinzukommenden Knotens (vor Anpassung)

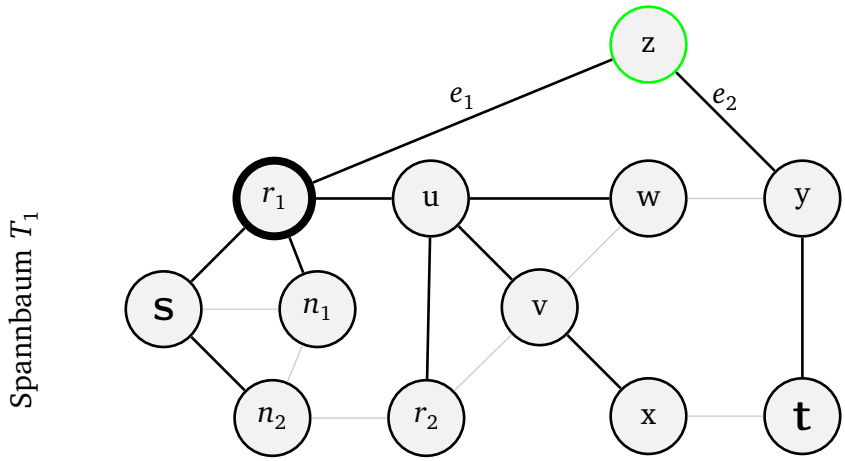


Abbildung A.4: Auswirkungen eines neu hinzukommenden Knotens (nach Anpassung)

zuzuweisen. Wie zuvor beim Hinzufügen eines neuen Links können auch durch das Hinzufügen eines Knotens mit mehr als einer Kante Abkürzungen entstehen, denen mit einer Anpassung der Spannbäume Rechnung getragen werden müsste, um bestmöglichen Stretch zu erreichen. Nimmt man jedoch suboptimalen Stretch bis zum nächsten Wartungsintervall in Kauf, können neue Knoten so effizient dem Netz beitreten.

Abbildung A.3 zeigt am Spannbaum T_1 aus Abbildung 2.8 die Auswirkungen des Hinzufügens eines Knotens zur Topologie. Der neue Knoten z ist dabei über die Kanten e_1 und e_2 mit den Knoten r_1 und y verbunden. Unter Verwendung von Algorithmus 5 kann Knoten z sich nun, sofern möglich, sofort in das Netz integrieren. In dem konkreten Beispiel würde Knoten z die Wurzel r_1 als Elternknoten präferieren und nur wenn diese keine weiteren virtuellen Koordinaten vergeben kann, wird Knoten y als Elternknoten gewählt. Mit den erhaltenen Koordinaten ist Knoten z nun erreichbar.

In der nächsten Iteration der periodischen Reparatur wird Knoten z dann, wie in Abbildung A.4 dargestellt, vollständig ins Netz integriert. Dies umfasst im vorliegenden Beispiel einen teilweisen Neuaufbau des Spannbaums T_1. Durch den neu hinzugekommen Knoten

Algorithmus 5 Schneller Beitritt zum Netz

```
1: procedure FastJoin(z)                          ▷ Knoten z tritt dem Netz bei
2:     parents[] ← null
3:     mindist[] ← ∞
4:     for n ∈ Γ(z) do                            ▷ Für jeden Nachbarknoten
5:         for i = 1; i ≤ l; ++i do
6:             if mindist[i] > d(n^{T_i}, r(T_i)^{T_i}) & incrementallyEmbeddable(n) then
7:                 parents[i] ← n
8:                 mindist[i] ← d(n^{T_i}, r(T_i)^{T_i})
9:             end if
10:        end for
11:    end for
12:    Join(parents)                  ▷ Netzbeitritt über zuvor bestimmte Elternknoten
13:    Announce(ID-Locator)             ▷ Registrierung der Adresse im Namensdienst
14: end procedure
```

z verfügen die Knoten y und t nun über bessere Pfade zur Wurzel des Spannbaums und wechseln daher ihre Elternknoten wie dargestellt. Im Gegensatz zu dem zuvor vorgestellten schnellen Netzbeitritt enthält der Spannbaum nun wieder die kürzesten Pfade zwischen allen Knoten und der Wurzel. Während nach dem schnellen Netzbeitritt von Knoten z über die Wurzel r_1 ein Paket von r_1 nach y den Weg r_1–u–w–y (Distanz 3) wählt, ist damit nun der bessere Weg r_1–z–y (Distanz 2) nutzbar.

A.1.3 Ausfall einer Kante

Im Gegensatz zum Hinzufügen von Links oder Knoten kann der Ausfall einer Kante die Weiterleitung von Paketen nach dem Greedy-Prinzip unmöglich machen. Auch hier ist zunächst wieder zu unterscheiden, ob die ausgefallene Kante Teil eines Spannbaums war oder nur als Abkürzung genutzt wurde. War die Kante Teil eines Spannbaums (und hat kein Blatt des Baumes angebunden), partitionert der Ausfall einer Kante den Baum in zwei Teile. Allgemeiner teilt der Ausfall von k Spannbaumkanten den Spannbaum T in $k+1$ Teilbäume. Die Zustellrate innerhalb der Teilbäume ist auch in Gegenwart der Ausfälle perfekt; wohl aber sind Pakete, die zwischen Teilbäumen zugestellt werden müssen, nun nicht mehr garantiert zustellbar. Gegebenenfalls ist es möglich über sonst als Abkürzungen genutzte Nicht-Spannbaumkanten zwischen Teilbäumen zu wechseln und darüber Pakete zuzustellen. Um jedoch eine perfekte Zustellrate zu erreichen, muss der Spannbaum an die geänderte Topologie so angepasst werden, dass verbundene Komponenten von G auch wieder in T verbunden sind. Ausfälle von Kanten, die nicht Teil des Spannbaums sind, haben keinen Einfluss auf die Zustellrate und erfordern daher keine Wartung der Spannbaumstruktur. Während solch ein Ausfall zu einer Erhöhung des Stretch beitragen kann, ist bei intaktem Spannbaum T jederzeit eine perfekte Zustellrate möglich.

Abbildung A.5 zeigt am Spannbaum T_1 aus Abbildung 2.8 die Auswirkungen des Ausfalls der rot gestrichtelten Kante zwischen den Knoten r_1 und u. Dieser Ausfall teilt den Spannbaum

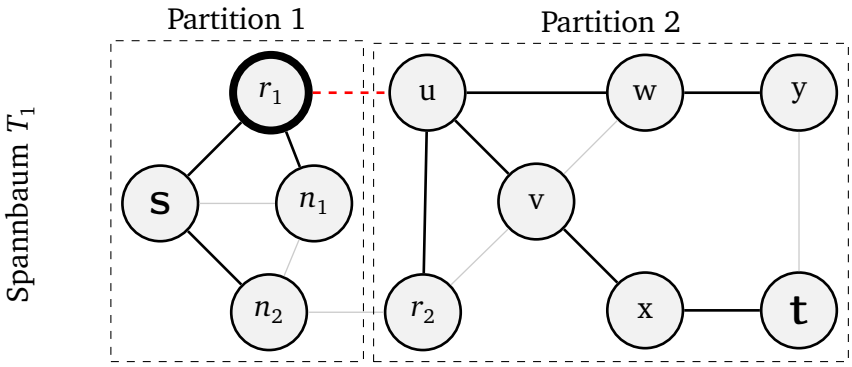


Abbildung A.5: Auswirkungen eines Kantenausfalls (vor Anpassung)

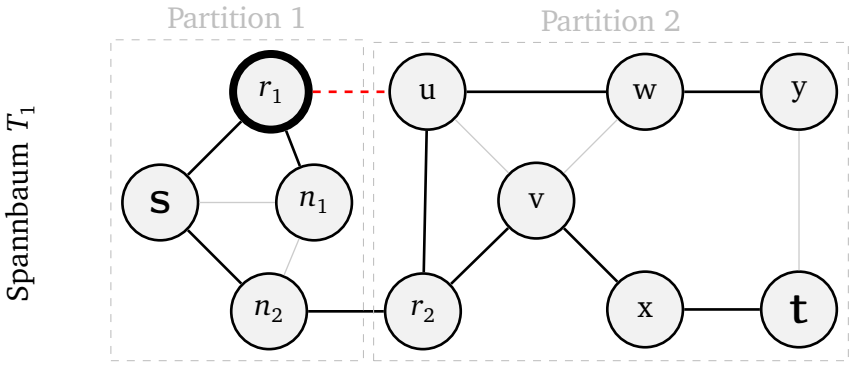


Abbildung A.6: Auswirkungen eines Kantenausfalls (nach Anpassung)

T_1 in die beiden eingezeichneten Partitionen. Zwei Knoten innerhalb derselben Partition können sich weiterhin erreichen; die Kommunikation zwischen Partitionen ist nur noch für die Knoten n_2 und r_2 möglich, die über eine Nicht-Spannbaumkante verbunden sind. Gelangt ein jeweils für die andere Partition bestimmtes Paket damit zu den Knoten n_2 oder r_2, kann dieses erfolgreich zugestellt werden, sofern keine weiteren Fehler aufgetreten sind. Obwohl der zugrundeliegende Graph also noch verbunden ist, hat der Kantenausfall massive Auswirkungen auf die gegenseitige Erreichbarkeit der Knoten. Ein Neuaufbau der Routingstrukturen resultiert in dem in Abbildung A.6 Spannbaum. Die Einbettung aller Knoten in der vorherigen Partition 2, die hier noch dezent angedeutet ist, ändert sich durch die Änderung der Spannbaumstruktur. Allgemeiner lässt sich festhalten, dass der Ausfall einer Kante sich folgendermaßen auswirkt: Der Teil des ursprünglichen Spannbaums oberhalb der ausgefallenen Kante bleibt inklusive Einbettung unverändert gültig. Jeder Knoten des Teilbaums unterhalb der ausgefallenen Kante muss, um die gierige Einbettung zu reparieren, zunächst einen neuen Elternknoten finden, der ihn auf dem aktuellen kürzestem Weg mit der Wurzel verbindet und danach seine Einbettung in dem neu entstandenen Spannbaum aktualisieren. Je näher die ausgefallene Kante daher an der Wurzel liegt und je größer der Teilbaum unterhalb der ausgefallenen Kante ist, desto höher ist der Aufwand für die Reparatur.

Für $l > 1$ Ebenen von Spannbäumen ist die Schwere des Ausfalls einer Kante zusätzlich davon abhängig, wieviele Spannbaumkanten über die ausgefallenene Kante geführt haben. Im schlimmsten Fall müssen l (Teil-)Spannbäume neu aufgebaut und eingebettet werden. Während mehrere Bäume zwar den durchschnittlichen Stretch reduzieren, erhöhen sie hier linear den erforderlichen Reparaturaufwand.

Obwohl der Spannbaum zerteilt ist, besteht selbst in dem vereinfachten Beispiel über die bisher als Abkürzung genutzte Kante n_2–r_2 die Möglichkeit zur Kommunikation zwischen den Teilbäumen. Das Vorhandensein einer größeren Menge von Abkürzungskanten oder gar mehreren Ebenen von Spannbäumen erhöht die Wahrscheinlichkeit, dass ein alternativer Pfad zur Verfügung steht.

A.1.4 Ausfall eines Knotens

Der Ausfall eines Knotens kann schwerwiegendere Folgen haben als der Ausfall einer Kante, da ein Knotenausfall letztendlich einem koordinierten Linkausfall entspricht. Dabei können Knoten, die mit dem ausgefallenen Knoten zuvor noch verbunden waren, anhand ihrer lokalen Sicht nicht feststellen, ob der Knoten tatsächlich ausgefallen ist oder lediglich die verbindende Kante. Abbildung A.7 stellt die im Vergleich zu einem Kantenausfall schwerwiegenderen Folgen eines Knotenausfalls am Beispiel von Knoten u dar. Ein Knoten, der nicht Wurzel oder Blatt des Spannbaums ist, hat eine Verbindung in Richtung Wurzel und $k \geq 1$ Kinder im Spannbaum sowie ggf. $s \geq 0$ Kanten, die nur als Abkürzung genutzt wurden und bei der Betrachtung des Reparaturaufwands damit keine Rolle spielen. Der Spannbaum wird durch den Knotenausfall dann entsprechend in $k + 1$ Partitionen unterteilt. Für den im Beispiel ausgefallenen Knoten u bedeutet dies eine Aufteilung des Spannbaums in die vier eingezeichneten Partitionen. Dadurch ist die Erreichbarkeit im Vergleich zu dem zuvor diskutierten Ausfall einer einzelnen Kante bereits viel deutlicher eingeschränkt. Abbildung A.8 zeigt eine gültige Spannbaumstruktur nach erfolgtem Neuaufbau. Durch den Knotenausfall

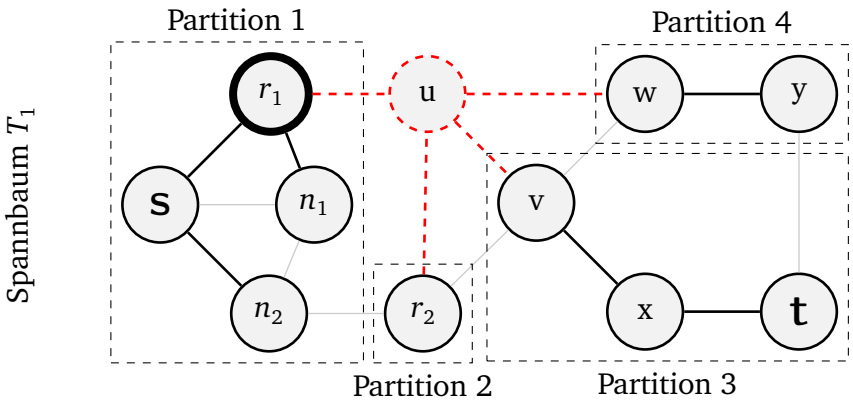


Abbildung A.7: Auswirkungen eines Knotenausfalls (vor Anpassung)

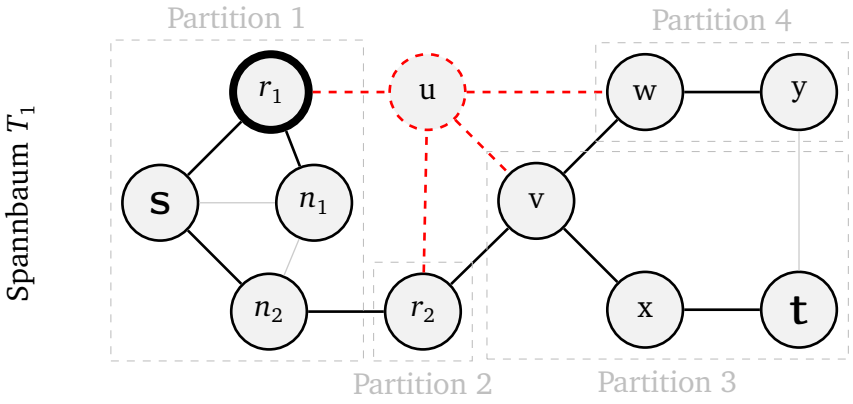


Abbildung A.8: Auswirkungen eines Knotenausfalls (nach Anpassung)

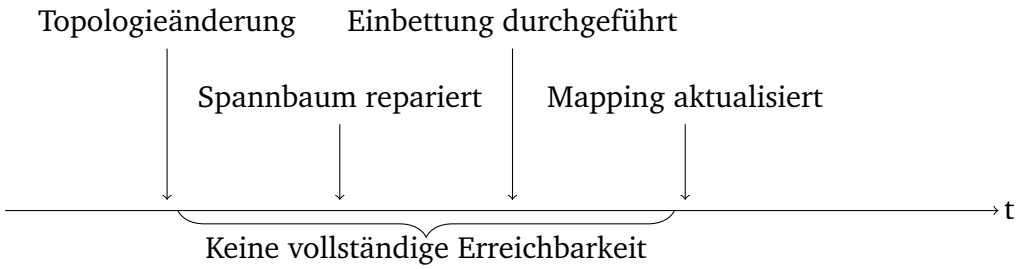


Abbildung A.9: Folgen einer Topologieänderung

müssen alle drei ehemaligen Kindbäume von Knoten u (Partition 2, Partition 3 und Partition 4) bei der Einbettung durch den erfolgten Wechsel des Elternknotens neue virtuelle Koordinaten erhalten. Die Kante vom ausgefallenen Knoten Richtung Wurzel erzeugt hier keinen zusätzlichen Aufwand.

Werden mehrere Ebenen von Spannbäumen eingesetzt, so vervielfacht sich dieses Problem, da nun im schlimmsten Fall $\sum_{i=1}^{l} ik_i$ disjunkte Spannbaumkanten (und damit Teilbäume) gleichzeitig ausfallen, wobei k_i die Anzahl der im Spannbaum i zu Kindern führenden Kanten ist.

Als Heuristik zur Auswahl der Wurzel eines Spannbaums wird häufig empfohlen, den Knoten h mit dem höchsten Grad $\deg(h)$ zu wählen. Bei Einheitskantengewichten, die hier bis auf weiteres angenommen werden, bedeutet dies, dass alle $\deg(h)$ Kanten Teil des Spannbaums sind und der Ausfall der Wurzel somit einen besonders schweren Reparaturaufwand erfordert. Zusätzlich wird zuvor die Auswahl einer neuen Wurzel erforderlich, die dann wieder den Aufbau des Spannbaums und die Einbettung initiieren kann.

A.1.5 Folgen aus Topologieänderungen

In den vorangegangenen Abschnitten war der Fokus auf den Folgen von Topologieänderungen auf die als Grundlage für die Einbettung verwendete Spannbaumstruktur gerichtet. Während der vollständige oder teilweise Neuaufbau von Spannbäumen einen gewissen Nachrichtenaufwand verursacht, ist dieser jedoch in den meisten Fällen im Vergleich zu den daraus resultierenden Folgekosten vernachlässigbar. Die Veränderung der Spannbaumstruktur führt zunächst zu einer inkonsistenten gierigen Einbettung, innerhalb derer nicht mehr alle Pakete erfolgreich zugestellt werden können. Dazu muss nach der Reparatur des Spannbaums zunächst eine erneute Einbettung durchgeführt werden. Dies resultiert darin, dass sich für möglicherweise große Teile der Knoten die virtuellen Koordinaten und damit die Adresse der Knoten ändert, über die diese erreichbar sind. Als Konsequenz daraus müssten die Knoten zusätzlich die Abbildung von ihrer Identität auf die neue, als Lokator dienende Adresse im Namensdienst aktualisieren. Da die Realisierung des Namensdienstes auf vielfältige Art und Weise erfolgen kann, wird im Rahmen dieser Arbeit nicht der dafür notwendige Nachrichtenaufwand betrachtet sondern als korrelierte Metrik die Anzahl der Adressen, die sich durch eine Anpassung geändert haben, herangezogen.

A.1.6 Zusammenfassung

Die vorangegangene Diskussion der verschiedenen Arten betrachteter Topologieänderungen zeigt, dass eine Erweiterung der Topologie um zusätzliche Knoten oder Kanten einerseits keine negativen Auswirkungen auf die bestehende Einbettung hat. So können nach wie vor alle Pakete zugestellt werden, wenn auch mit gegebenenfalls etwas schlechterem Stretch, da durch die Topologieerweiterung eventuell neue, bessere Pfade verfügbar wären. Hinzugekommene Kanten werden von den angrenzenden Knoten auch ohne Anpassung der Einbettung als Abkürzungen verwendet. Die vorgeschlagene *FastJoin*-Prozedur erlaubt es, hinzugekommene Knoten in die vorhandene Einbettung zu integrieren, ohne dass diese angepasst werden muss. Dadurch wird nur ein geringer Nachrichtenaufwand verursacht, der dadurch möglich wird, dass bis zur nächsten periodischen Neueinbettung eventuell höherer Stretch in Kauf genommen werden muss. Während zum Erreichen des bestmöglichen Stretch nach einer Erweiterung der Topologie zwar eine erneute Einbettung durchgeführt werden sollte, besteht keine zwingende Notwendigkeit, dies sofort zu tun. Da jederzeit alle Pakete zugestellt werden können, kann die erneute Einbettung auf die nächste periodische Neueinbettung verschoben werden.

Der Ausfall von Komponenten ist deutlich schwerwiegender, da dadurch die Weiterleitung nach dem Greedy-Prinzip in der Regel nicht mehr möglich ist. Wie zuvor gezeigt, kann der Reparaturaufwand für diese Änderungen beträchtlich sein. Neben der zuvor diskutierten Anpassung der Spannbäume an die geänderte Topologie sowie die erneute Einbettung dieses Spannbaums kommt hinzu, dass nachgelagert für jede Adressänderung der Namensdienst aktualisiert werden müsste. Ändert sich also die Adresse eines Knotens, d.h. eine der virtuellen Koordinaten, so muss der Knoten die neue Abbildung von seiner Identität auf die aktualisierte Adresse im Namensdienst registrieren, um für andere Knoten erreichbar zu sein.

Literaturverzeichnis

[1] D. Meyer, L. Zhang und K. Fall. *Report from the IAB Workshop on Routing and Addressing*. RFC 4984 (Informational). Internet Engineering Task Force, Sep. 2007 (siehe S. 1, 147).

[2] A. Brady und L. Cowen. „Compact routing on power-law graphs with additive stretch“. In: *in ALENEX*. 2006, S. 119–128 (siehe S. 2, 12, 20, 21, 70).

[3] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali und C. Diot. „Characterization of failures in an operational IP backbone network“. In: *IEEE/ACM Trans. Netw.* 16.4 (Aug. 2008), S. 749–762 (siehe S. 2, 90, 92).

[4] G. Schaffrath, **C. Werle**, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel und L. Mathy. „Network Virtualization Architecture: Proposal and Initial Prototype“. In: *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*. VISA '09. Barcelona, Spain: ACM, 2009, S. 63–72 (siehe S. 4, 36, 37).

[5] R. Bless und **C. Werle**. „Network Virtualization from a Signaling Perspective“. In: *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*. 2009, S. 1–6 (siehe S. 4).

[6] S. Mies. „Autonomous interconnection of heterogeneous networks“. Magisterarb. Juni 2012 (siehe S. 4).

[7] **C. Werle**, S. Mies und M. Zitterbart. „On benchmarking routing protocols“. In: *Networks (ICON), 2011 17th IEEE International Conference on*. 2011, S. 305–310 (siehe S. 4, 45).

[8] **C. Werle** und O. Waldhorst. „Everything is better with Sprinkles: Greedy Routing with Bounded Stretch“. In: *Second International Conference on Future Generation Communication Technologies*. London, UK, Dez. 2013, S. 305–310 (siehe S. 4, 56).

[9] **C. Werle** und O. Waldhorst. „Greedy Failure-Carrying Packets“. In: *International Conference on Computing, Networking, and Communications*. Honolulu, HI, USA, Feb. 2014, S. 1016–1022 (siehe S. 4, 90).

[10] D. Farinacci, V. Fuller, D. Meyer und D. Lewis. *The Locator/ID Separation Protocol (LISP)*. RFC 6830 (Experimental). Internet Engineering Task Force, Jan. 2013 (siehe S. 5).

[11] C. Gavoille und D. Peleg. „Compact and Localized Distributed Data Structures“. In: *Distrib. Comput.* 16.2-3 (Sep. 2003), S. 111–120 (siehe S. 8).

[12] D. Peleg. „Proximity-preserving labeling schemes“. In: *Journal of Graph Theory* 33.3 (2000), S. 167–176 (siehe S. 8, 21).

[13] C. Gavoille, D. Peleg, S. Pérennes und R. Raz. „Distance labeling in graphs“. In: *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. SODA '01. Washington, D.C., USA: Society for Industrial und Applied Mathematics, 2001, S. 210–219 (siehe S. 8).

[14] R. Kleinberg. „Geographic Routing Using Hyperbolic Space“. In: *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. 2007, S. 1902–1909 (siehe S. 11, 24, 89).

[15] L. Lu. „The diameter of random massive graphs“. In: *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. SODA '01. Washington, D.C., USA: Society for Industrial und Applied Mathematics, 2001, S. 912–921 (siehe S. 12, 13, 20).

[16] Y. Shavitt und E. Shir. „DIMES: Let the Internet Measure Itself“. In: *SIGCOMM Comput. Commun. Rev.* 35.5 (Okt. 2005), S. 71–74 (siehe S. 13).

[17] R. Cohen, M. Gonen und A. Wool. „Bounding the Bias of Tree-Like Sampling in IP Topologies“. In: *CoRR* abs/cs/0611157 (2006) (siehe S. 13).

[18] K. Claffy, T. Monk und D. McRobb. „Internet Tomography“. In: *Nature* (Jan. 1999) (siehe S. 13).

[19] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, K. Claffy und A. Vahdat. „The Internet AS-level Topology: Three Data Sources and One Definitive Metric“. In: *SIGCOMM Comput. Commun. Rev.* 36.1 (Jan. 2006), S. 17–26 (siehe S. 13).

[20] Y. XU und Z. WANG. „On Internet Topology Modeling and an Improved BA Model“. In: *Journal of Networks* 6.3 (2011) (siehe S. 14).

[21] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh und J. van der Merwe. „The case for separating routing from routers“. In: *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*. FDNA '04. Portland, Oregon, USA: ACM, 2004, S. 5–12 (siehe S. 14).

[22] V. Kotronis, X. Dimitropoulos und B. Ager. „Outsourcing the routing control logic: better internet routing based on SDN principles“. In: *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. HotNets-XI. Redmond, Washington: ACM, 2012, S. 55–60 (siehe S. 14).

[23] D. Peleg und E. Upfal. „A Tradeoff between Space and Efficiency for Routing Tables (Extended Abstract)“. In: *STOC*. 1988, S. 43–52 (siehe S. 15).

[24] D. Peleg und E. Upfal. „A trade-off between space and efficiency for routing tables“. In: *J. ACM* 36.3 (Juli 1989), S. 510–530 (siehe S. 15, 16).

[25] P. Fraigniaud und C. Gavoille. „Universal routing schemes“. English. In: *Distributed Computing* 10.2 (1997), S. 65–78 (siehe S. 16).

[26] C. Gavoille und M. Gengler. „Space-efficiency for routing schemes of stretch factor three“. In: *J. Parallel Distrib. Comput.* 61.5 (Mai 2001), S. 679–687 (siehe S. 16, 17).

[27] M. Thorup und U. Zwick. „Approximate distance oracles“. In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. STOC '01. Hersonissos, Greece: ACM, 2001, S. 183–192 (siehe S. 16, 17).

[28] L. J. Cowen. „Compact routing with minimum stretch“. In: *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*. SODA '99. Baltimore, Maryland, USA: Society for Industrial und Applied Mathematics, 1999, S. 255–260 (siehe S. 16).

[29] Y. Mao, F. Wang, L. Qiu, S. Lam und J. Smith. „S4: Small State and Small Stretch Compact Routing Protocol for Large Static Wireless Networks“. In: *Networking, IEEE/ACM Transactions on* 18.3 (2010), S. 761–774 (siehe S. 19).

[30] B. A. Ford. „UIA: A Global Connectivity Architecture for Mobile Personal Devices“. Diss. Massachusetts Institute of Technology, Sep. 2008 (siehe S. 19).

[31] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone und S. Ratnasamy. „Scalable routing on flat names“. In: *Proceedings of the 6th International COnference*. Co-NEXT '10. New York, NY, USA: ACM, 2010, 20:1–20:12 (siehe S. 19).

[32] M. Faloutsos, P. Faloutsos und C. Faloutsos. „On power-law relationships of the Internet topology“. In: *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*. SIGCOMM '99. Cambridge, Massachusetts, USA: ACM, 1999, S. 251–262 (siehe S. 20).

[33] T. Bu und D. Towsley. „On distinguishing between Internet power law topology generators“. In: *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Bd. 2. 2002, 638–647 vol.2 (siehe S. 20).

[34] S. Jaiswal, A. L. Rosenberg und D. Towsley. „Comparing the Structure of Power-Law Graphs and the Internet AS Graph“. In: *Proceedings of the 12th IEEE International Conference on Network Protocols*. ICNP '04. Washington, DC, USA: IEEE Computer Society, 2004, S. 294–303 (siehe S. 20).

[35] A. Clauset, C. Shalizi und M. Newman. „Power-Law Distributions in Empirical Data“. In: *SIAM Review* 51.4 (2009), S. 661–703 (siehe S. 20).

[36] W. Chen, C. Sommer, S.-H. Teng und Y. Wang. „A compact routing scheme and approximate distance oracle for power-law graphs“. In: *ACM Trans. Algorithms* 9.1 (Dez. 2012), 4:1–4:26 (siehe S. 20).

[37] M. Thorup und U. Zwick. „Compact routing schemes“. In: *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*. SPAA '01. Crete Island, Greece: ACM, 2001, S. 1–10 (siehe S. 21).

[38] G. G. Finn. *Routing and Addressing Problems in Large Metropolitan-scale Internetworks*. Techn. Ber. ISI/RR-87-180. 4676 Admiralty Way, Marina del Rey, CA 90292: USC/Information Sciences Institute, März 1987 (siehe S. 23).

[39] E. Kranakis, H. Singh und J. Urrutia. „Compass Routing on Geometric Networks“. In: *IN PROC. 11 TH CANADIAN CONFERENCE ON COMPUTATIONAL GEOMETRY*. 1999, S. 51–54 (siehe S. 24).

[40] A. Cvetkovski und M. Crovella. „Hyperbolic Embedding and Routing for Dynamic Graphs“. In: *INFOCOM 2009, IEEE*. 2009, S. 1647–1655 (siehe S. 24, 89, 97).

[41] D. Eppstein. „Manhattan orbifolds“. In: *Topology and its Applications* 157.2 (2010), S. 494–507 (siehe S. 24, 89).

[42] D. Eppstein und M. Goodrich. „Succinct Greedy Geometric Routing Using Hyperbolic Geometry“. In: *Computers, IEEE Transactions on* 60.11 (2011), S. 1571–1580 (siehe S. 24, 89).

[43] P. Maymounkov. *Greedy Embeddings, Trees, and Euclidean vs. Lobachevsky Geometry*. Techn. Ber. Author's Manuscript. MIT, 2006 (siehe S. 24, 89).

[44] J. Herzen, C. Westphal und P. Thiran. „Scalable routing easy as PIE: A practical isometric embedding protocol“. In: *Network Protocols (ICNP), 2011 19th IEEE International Conference on*. Okt. 2011, S. 49–58 (siehe S. 24, 26, 68, 87, 89, 124, 129).

[45] A. Cvetkovski und M. Crovella. „On the choice of a spanning tree for greedy embedding of network graphs“. English. In: *Networking Science* (2013), S. 1–11 (siehe S. 26).

[46] R. Perlman. „An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN“. In: *Proceedings of the Ninth Symposium on Data Communications*. SIGCOMM '85. Whistler Moutain, British Columbia, Canada: ACM, 1985, S. 44–53 (siehe S. 29).

[47] A. Feldmann. „Internet Clean-slate Design: What and Why?“ In: *SIGCOMM Comput. Commun. Rev.* 37.3 (Juli 2007), S. 59–64 (siehe S. 35, 36).

[48] R. Braden, T. Faber und M. Handley. „From Protocol Stack to Protocol Heap: Role-based Architecture“. In: *SIGCOMM Comput. Commun. Rev.* 33.1 (Jan. 2003), S. 17–22 (siehe S. 35).

[49] X. Yang. „NIRA: A New Internet Routing Architecture“. In: *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*. FDNA '03. Karlsruhe, Germany: ACM, 2003, S. 301–312 (siehe S. 35).

[50] I. Stoica, D. Adkins, S. Zhuang, S. Shenker und S. Surana. „Internet Indirection Infrastructure“. In: *IEEE/ACM Trans. Netw.* 12.2 (Apr. 2004), S. 205–218 (siehe S. 35).

[51] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan und P. Steenkiste. „XIA: Efficient Support for Evolvable Internetworking“. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. San Jose, CA: USENIX Association, 2012, S. 23–23 (siehe S. 35).

[52] S. Deering und R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946. Internet Engineering Task Force, Dez. 1998 (siehe S. 35).

[53] J. Czyz, A. Mark, J. Zhang, S. Iekel-Johnson, E. Osterweil und M. Bailey. *Measuring IPv6 Adoption*. Techn. Ber. TR-13-004. 1947 Center Street, Ste. 600, Berkeley, CA 94704: International Computer Science Institute, Aug. 2013 (siehe S. 35).

[54] T. Anderson, L. Peterson, S. Shenker und J. Turner. „Overcoming the Internet impasse through virtualization“. In: *Computer* 38.4 (Apr. 2005), S. 34–41 (siehe S. 35).

[55] S. Bryant und P. Pate. *Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture*. RFC 3985 (Informational). Updated by RFC 5462. Internet Engineering Task Force, März 2005 (siehe S. 36).

[56] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe und J. Rexford. „Virtual Routers on the Move: Live Router Migration As a Network-management Primitive“. In: *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. SIGCOMM '08. Seattle, WA, USA: ACM, 2008, S. 231–242 (siehe S. 36).

[57] **C. Werle**, P. Papadimitriou, I. Houidi, W. Louati, D. Zeghlache, R. Bless und L. Mathy. „Building Virtual Networks Across Multiple Domains“. In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM '11. Toronto, Ontario, Canada: ACM, 2011, S. 412–413 (siehe S. 36).

[58] L. Völker, D. Martin, I. El Khayaut, **C. Werle** und M. Zitterbart. „A Node Architecture for 1000 Future Networks“. In: *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*. Juni 2009, S. 1–5 (siehe S. 36).

[59] L. Völker, D. Martin, **C. Werle**, M. Zitterbart und I. El-Khayat. „Selecting Concurrent Network Architectures at Runtime“. In: *Communications, 2009. ICC '09. IEEE International Conference on*. Juni 2009, S. 1–5 (siehe S. 36).

[60] N. Feamster, L. Gao und J. Rexford. „How to Lease the Internet in Your Spare Time“. In: *SIGCOMM Comput. Commun. Rev.* 37.1 (Jan. 2007), S. 61–64 (siehe S. 37).

[61] Y. Zhu, R. Zhang-Shen, S. Rangarajan und J. Rexford. „Cabernet: Connectivity Architecture for Better Network Services“. In: *Proceedings of the 2008 ACM CoNEXT Conference*. CoNEXT '08. Madrid, Spain: ACM, 2008, 64:1–64:6 (siehe S. 37).

[62] D. D. Clark, J. Wroclawski, K. R. Sollins und R. Braden. „Tussle in Cyberspace: Defining Tomorrow's Internet“. In: *IEEE/ACM Trans. Netw.* 13.3 (Juni 2005), S. 462–475 (siehe S. 37).

[63] R. Bless und **C. Werle**. „Network Virtualization from a Signaling Perspective“. In: *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*. Juni 2009, S. 1–6 (siehe S. 38).

[64] G. Schaffrath. „Virtual Network Management“. Diss. Technische Universität Berlin, Nov. 2012 (siehe S. 38, 42, 43).

[65] M. Yu, Y. Yi, J. Rexford und M. Chiang. „Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration“. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (März 2008), S. 17–29 (siehe S. 38).

[66] S. Paul, J. Pan und R. Jain. „Architectures for the future networks and the next generation Internet: A survey“. In: *Computer Communications* 34.1 (2011), S. 2–42 (siehe S. 40).

[67] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink und M. Wawrzoniak. „Operating System Support for Planetary-scale Network Services“. In: *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. NSDI'04. San Francisco, California: USENIX Association, 2004, S. 19–19 (siehe S. 40).

[68] T. Anderson und T. Roscoe. „Learning from PlanetLab“. In: *Proceedings of the 3rd Conference on USENIX Workshop on Real, Large Distributed Systems - Volume 3*. WORLDS'06. Seattle, WA: USENIX Association, 2006, S. 6–6 (siehe S. 40).

[69] A. Bavier, N. Feamster, M. Huang, L. Peterson und J. Rexford. „In VINI Veritas: Realistic and Controlled Network Experimentation“. In: *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '06. Pisa, Italy: ACM, 2006, S. 3–14 (siehe S. 40).

[70] J. Dike. „A User-mode Port of the Linux Kernel“. In: *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*. ALS'00. Atlanta, Georgia: USENIX Association, 2000, S. 7–7 (siehe S. 41).

[71] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson und J. Rexford. „Trellis: A Platform for Building Flexible, Fast Virtual Networks on Commodity Hardware“. In: *Proceedings of the 2008 ACM CoNEXT Conference*. CoNEXT '08. Madrid, Spain: ACM, 2008, 72:1–72:6 (siehe S. 41).

[72] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb und A. Joglekar. „An Integrated Experimental Environment for Distributed Systems and Networks“. In: *OSDI02*. USENIXASSOC. Boston, MA, Dez. 2002, S. 255–270 (siehe S. 41).

[73] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu und H. Yu. „Advances in Network Simulation“. In: *Computer* 33.5 (Mai 2000), S. 59–67 (siehe S. 41, 48).

[74] *GENI: Global Environment for Network Innovation*. `http://www.geni.net/` (zugegriffen am 13.12.2013) (siehe S. 41).

[75] N. Feamster, L. Gao und J. Rexford. „How to Lease the Internet in Your Spare Time“. In: *SIGCOMM Comput. Commun. Rev.* 37.1 (Jan. 2007), S. 61–64 (siehe S. 42).

[76] Y. Zhu, R. Zhang-Shen, S. Rangarajan und J. Rexford. „Cabernet: Connectivity Architecture for Better Network Services“. In: *Proceedings of the 2008 ACM CoNEXT Conference*. CoNEXT '08. Madrid, Spain: ACM, 2008, 64:1–64:6 (siehe S. 42).

[77] A. Wundsam. „Towards Improved Control and Troubleshooting for Operational Networks“. Diss. Technische Universität Berlin, Juli 2011 (siehe S. 43).

[78] R. Bless, M. Röhricht und **C. Werle**. „Authenticated Setup of Virtual Links with Quality-of-Service Guarantees“. In: *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*. 2011, S. 1–8 (siehe S. 43).

[79] R. Bless, M. Röhricht und **C. Werle**. „Authenticated Quality-of-Service Signaling for Virtual Networks“. In: *Journal of Communications* 7.1 (Jan. 2012), S. 17–27 (siehe S. 43).

[80] *G-Lab: German Lab*. `http://www.german-lab.de` (zugegriffen am 13.12.2013) (siehe S. 43).

[81] R. Bless, M. Schöller, P. Smith, F. Pallas und J. Horneber. „An Architectural Model for Deploying Critical Infrastructure Services in the Cloud“. In: *5th IEEE International Conference on Cloud Computing Technology and Science*. CloudCom. CloudCom, Dez. 2013 (siehe S. 43).

[82] B. Quoitin und S. Uhlig. „Modeling the Routing of an Autonomous System with C-BGP“. In: *Netwrk. Mag. of Global Internetwkg.* 19.6 (Nov. 2005), S. 12–19 (siehe S. 45).

[83] I. Baumgart, B. Heep und S. Krause. „OverSim: A Flexible Overlay Network Simulation Framework“. In: *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*. Mai 2007, S. 79–84 (siehe S. 45).

[84] B. Schiller, D. Bradler, I. Schweizer, M. Mühlhäuser und T. Strufe. „GTNA: a framework for the graph-theoretic network analysis“. In: *Proceedings of the 2010 Spring Simulation Multiconference*. SpringSim '10. Orlando, Florida: Society for Computer Simulation International, 2010, 111:1–111:8 (siehe S. 46).

[85] L. Hogie, D. Papadimitriou, I. Tahiri und F. Majorczyk. „Simulating Routing Schemes on Large-Scale Topologies“. In: *Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*. PADS '10. Washington, DC, USA: IEEE Computer Society, 2010, S. 132–141 (siehe S. 46).

[86] *INET Rahmenwerk*. `http://inet.omnetpp.org/` (zugegriffen am 23.1.2014) (siehe S. 46).

[87] L. Hogie, A. Lancin und I. Tahiri. *Mascsim*. `http://www-sop.inria.fr/mascotte/software/mascsim/` (zugegriffen am 23.1.2014) (siehe S. 47).

[88] G. Riley und T. Henderson. „The ns-3 Network Simulator“. In: *Modeling and Tools for Network Simulation*. Hrsg. von K. Wehrle, M. Güneş und J. Gross. Springer Berlin Heidelberg, 2010, S. 15–34 (siehe S. 48).

[89] A. Varga und R. Hornig. „An Overview of the OMNeT++ Simulation Environment“. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. Simutools '08. Marseille, France: ICST (Institute for Computer Sciences, Social-Informatics und Telecommunications Engineering), 2008, 60:1–60:10 (siehe S. 48).

[90] R. Barr, Z. J. Haas und R. van Renesse. „JiST: An Efficient Approach to Simulation Using Virtual Machines: Research Articles“. In: *Softw. Pract. Exper.* 35.6 (Mai 2005), S. 539–576 (siehe S. 48).

[91] E. Weingärtner, H. vom Lehn und K. Wehrle. „A Performance Comparison of Recent Network Simulators“. In: *Communications, 2009. ICC '09. IEEE International Conference on*. 2009, S. 1–5 (siehe S. 48).

[92] R Core Team. *R: A Language and Environment for Statistical Computing*. ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria, 2012 (siehe S. 50).

[93] *The Boost Graph Library: User Guide and Reference Manual*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002 (siehe S. 51).

[94] F. Geisberger. „Implementierung und Optimierung des Routingprotokolls PIE für den Einsatz in dynamischen Netzen“. Diplomarbeit. Karlsruher Institut für Technologie, 2012 (siehe S. 91).

[95] M. Shand und S. Bryant. *IP Fast Reroute Framework*. RFC 5714 (Informational). Internet Engineering Task Force, Jan. 2010 (siehe S. 93).

[96] A. Atlas und A. Zinin. *Basic Specification for IP Fast Reroute: Loop-Free Alternates*. RFC 5286 (Proposed Standard). Internet Engineering Task Force, Sep. 2008 (siehe S. 94).

[97] C. Filsfils, P. Francois, M. Shand, B. Decraene, J. Uttaro, N. Leymann und M. Horneffer. *Loop-Free Alternate (LFA) Applicability in Service Provider (SP) Networks*. RFC 6571 (Informational). Internet Engineering Task Force, Juni 2012 (siehe S. 94).

[98] S. Bryant, S. Previdi und M. Shand. *A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses*. RFC 6981 (Informational). Internet Engineering Task Force, Aug. 2013 (siehe S. 94).

[99] J. Feigenbaum, B. Godfrey, A. Panda, M. Schapira, S. Shenker und A. Singla. „Brief announcement: on the resilience of routing tables“. In: *PODC*. 2012, S. 237–238 (siehe S. 94).

[100] S. Nelakuditi, S. Lee, Y. Yu und Z.-L. Zhang. „Failure Insensitive Routing for Ensuring Service Availability“. In: *Proceedings of the 11th International Conference on Quality of Service*. IWQoS'03. Berkeley, CA, USA: Springer-Verlag, 2003, S. 287–304 (siehe S. 94).

[101] A. Farrel, A. Satyanarayana, A. Iwata, N. Fujita und G. Ash. *Crankback Signaling Extensions for MPLS and GMPLS RSVP-TE*. RFC 4920 (Proposed Standard). Internet Engineering Task Force, Juli 2007 (siehe S. 94).

[102] „Private Network-Network Interface Specification Version 1.0 (PNNI 1.0)“. In: *The ATM Forum Technical Committee* (März 1996) (siehe S. 94).

[103] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker und I. Stoica. „Achieving convergence-free routing using failure-carrying packets“. In: *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. SIGCOMM '07. Kyoto, Japan: ACM, 2007, S. 241–252 (siehe S. 95).

[104] P. Bose, P. Morin, I. Stojmenović und J. Urrutia. „Routing with Guaranteed Delivery in Ad Hoc Wireless Networks“. In: *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*. DIALM '99. Seattle, Washington, USA: ACM, 1999, S. 48–55 (siehe S. 97).

[105] S. Sahhaf, W. Tavernier, D. Colle, M. Pickavet und P. Demeester. „Link failure recovery technique for greedy routing in the hyperbolic plane“. In: *Computer Communications* 36.6 (2013). Reliable Network-based Services, S. 698–707 (siehe S. 101).

[106] P. Pantazopoulos, M. Karaliopoulos und I. Stavrakakis. „On the Local Approximations of Node Centrality in Internet Router-level Topologies“. In: *7th International Workshop on Self-Organizing Systems (IFIP IWSOS'13), Palma de Mallorca.* Mai 2013 (siehe S. 126).