Skalierbare Datenverwaltung in einem auf Zufallsgraphen basierten, verteilten System für interaktive Anwendungen

Inaugural-Dissertation

zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Andreas Barbian
geb. Jägermann

aus Heessen, jetzt Hamm

Düsseldorf, November 2016

aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Erstgutachter: Univ.-Prof. Dr. Michael Schöttner
Zweitgutachter: Jun.-Prof. Dr. Kalman Graffi

Tag der mündlichen Prüfung: 09.01.2017

Meiner Familie zur Freude und zum Dank.

"Hypothesen sind Netze; nur der wird fangen, der auswirft."

Novalis, deutscher Dichter
($\star$ 02. Mai 1772, † 25. März 1801)

Kurzfassung

Im Rahmen dieser Arbeit wurde mit Omentum das erste Peer-to-Peer-System entwickelt, das sich auf virtuelle Mikroskopie fokussiert – als Beispielanwendung für verteilte interaktive Lernplattformen. Als besondere Herausforderungen sind zum einen die sehr große Datenmenge von rund 50 GB je Datensatz bei mehreren tausend Datensätzen und zum anderen der hohe Grad an Interaktionen zwischen sehr vielen Benutzern zu nennen.

Als wichtigste Beiträge dieser Arbeit sind die Entwicklung eines effizienten, unstrukturierten Overlays mit zweistufiger Pfadkompression für optimiertes Routing und einer innovativen Authentifizierungsstrategie im Prinzip einer Single-Sign-On-Strategie anzuführen. Weiterhin ist das neuartige portable User-Space Dateisystem USPFS mit Fokus auf der Speicherung sehr vieler kleiner Dateien bei geringstem Metadatenoverhead von lediglich 0,3% zu nennen.

Durch eine effektive Datenaufbereitung konnte eine optimale Partitionierungsstrategie realisiert werden, die gleichzeitig eine Datenanforderung aus multiplen Quellen unterstützt und damit auch in schwachen oder sehr heterogenen Netzwerkumgebungen sehr gute Ergebnisse erzielt. Eine neukonzipierte Bestimmung der verfügbaren Bandbreite mittels passive probing über TCP mit Congestion Control mit einer Messgenauigkeit von 98% erlaubt es nicht nur die Überlastung einzelner Peers zu erkennen, sondern ermächtigt jeden Peer bei anhaltender Belastung aktiv Unterstützung für seine Aufgaben anzufordern.

Das neu entwickelte Overlay bietet durch seinen Zufallsgraphen eine effiziente Basis zur Unterstützung komplexer Suchen nach Benutzer-generierten Inhalten durch die Lösung eines reduzierten Rendezvous-Problems wie auch für einen sehr schnellen Lookup einzelner Datensätze über ihre ID. Gleichzeitig besteht durch die äußerst robuste Architektur kein Bedarf an der Änderung von Routing-Tabellen bei einem Ausfall von Knoten. So lässt sich durch die Verknüpfung der Peers untereinander selbst ein Ausfall von 80% der Knoten eines Netzwerks schnell kompensieren ohne den Verwaltungsaufwand der Peers überproportional zu steigern.

Das Overlay mit seinen innovativen Aspekten ist mit dem aktuellen Forschungsstand verglichen worden und es konnten für den skizzierten Anwendungsfall sehr gute Ergebnisse in Simulationen mit 10^6 Knoten erzielt werden. In der medizinischen Lehre wird Omentum bereits erfolgreich in curricularen Veranstaltungen an der HHU mit insgesamt mehreren Hundert Benutzern eingesetzt.

Abstract

Within the framework of this dissertation Omentum was developed, the first peer-to-peer-system solely focusing on virtual microscopy as a sample application for a distributed interactive learning platform. The particular challenges are large data volumes of several thousands of datasets (up to 50 GB each) and very high levels of user interaction.

The two main contributions of this work are the development of an efficient, unstructured overlay with a two-stage path compression for optimal routing and the innovative advancement of a single-sign-on authentication strategy. In addition, USPFS was established; a novel portable user-space file system for data storage of very small datasets at the lowest metadata overhead possible (0.3%).

The implemented partitioning strategy uses an effective data processing method and provides optimal support for data requests from multiple sources and performs very well in weak and highly heterogeneous network environments. Passive probing over TCP with Congestion Control, a newly designed estimation of available bandwidths, achieved 98% measurement accuracy identifying individual peer overload and even more authorizing each peer to actively request assistance for its tasks in case of persisting loads.

The random graph of the newly developed overlay forms an efficient support base for complex searches of user-generated content as it solves the reduced rendezvous-problem and at the same time granting very rapid Lookups of single datasets using their ID. Furthermore, the architecture is highly robust and therefore there is no need for the routing tables to be modified, even in case of node failure. Linking the peers amongst themselves, quickly compensates pervasive network failures. Even a 80% node failure is offset without disproportional increase of administrate expenses of the peers.

The Omentum overlay with its cutting-edge aspects has been compared to the current state of research and, for the outlined application case, obtained very good results in simulations with 10^6 nodes. Furthermore, Omentum has already been successfully deployed in medical teaching at HHU Dusseldorf with several hundreds of users.

Danksagung

Meinen herzlichsten Dank möchte ich Herrn Prof. Dr. Michael Schöttner für die Betreuung dieser Dissertation aussprechen. Die zahlreichen konstruktiven Diskussionen und seine vorbehaltlose Unterstützung haben wesentlichen Anteil an dieser Arbeit.
Weiterhin gilt mein Dank Herrn Jun.-Prof. Dr. Kalman Graffi für die Supervision und Unterstützung als Mentor.

Bei Herrn Prof. Dr. Timm J. Filler möchte ich mich für die uneingeschränkte Unterstützung dieser Dissertation ganz besonders bedanken.

Elisabeth Wesbuer danke ich sehr herzlich für die Unterstützung in der Endphase dieser Arbeit, da sie mir mit ihrem unermüdlichen Engagement den Rücken in der Elektronenmikroskopie freigehalten hat.

Ein unverzichtbarer Dank gilt Angela Rennwanz, die mir an vielen Stellen den Weg durch den bürokratischen Dschungel geebnet hat.

Ebenfalls möchte ich Dennis Malenica, Michael Morosow, Stefan Nothaas, Daniel Ott, Ralph-Gordon Paul und Dominik Weinhold danken, die durch Ihre Qualifikationsarbeiten einen Beitrag zu dieser Dissertation geleistet haben. Insbesondere bedanke ich mich bei Dennis Malenica für die vielen fruchtbaren Diskussionen, die offenen Worte und das unermüdliche Lektorat.

Ganz besonderer Dank gilt meiner Frau Birte für die Unterstützung in allen Phasen dieser besonderen Zeit und die vielen konstruktiven Gespräche zu den unmöglichsten Zeiten.
Meiner Familie, insbesondere meiner Mutter Ranghild, möchte ich ganz herzlich für den Rückhalt und die Geduld während meiner gesamten Promotionszeit danken.

Andreas Barbian

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Peer-to-Peer-Systeme sind seit vielen Jahren ein fester Bestandteil informationsverarbeitender Systeme. So werden sie z.B. im Bereich des „File Sharing“ (z.B. BitTorrent), der Kommunikation (z.B. Skype [14]) oder für Audio- und Video-Streaming [10, 175] eingesetzt. Cloud-Systeme beschreiben ein Modell für bedarfsorientiert, überall und jederzeit bereitgestellte Pools von konfigurierbaren Systemressourcen [120]. Sie sind in der Regel innerhalb eines Datacenters dezentral organisiert, verwenden aber häufig einen zentralen Punkt für die Kommunikation mit den Benutzern, der jedoch meist über virtuelle Instanzen in kürzester Zeit repliziert werden kann und damit gegen die meisten Ausfälle abgesichert ist.

Im Vergleich zu Cloud-Systemen sind traditionelle Peer-to-Peer-Systeme kostengünstiger, da die Systemressourcen von den Nutzern bereit gestellt werden und nicht von einem Anbieter gegen Entgelt zur Verfügung gestellt werden müssen.

Der Vorteil des Peer-to-Peer-Ansatzes ist neben den Kosten ganz eindeutig das Fehlen des kritischen Punktes, an dessen Funktion die Operationsfähigkeit des gesamten Netzwerks hängt.

1.1 Motivation

Diese Arbeit beschäftigt sich mit einer verteilten Architektur für eine interaktive Lehr- und Lernplattform im Bereich virtueller Mikroskopie. Als primärer Anwendungsfall im Rahmen der medizinischen Aus-, Fort- und Weiterbildung ist hier die dezentrale Akquise, Speicherung und Verwaltung von Wissen in Form von Annotationen (Beschriftungen, Ergänzungen) zu den bereitgestellten Bildern zu sehen. Herausfordernd an dem gewählten Arbeitskontext sind die großen Datenmengen, die im verteilten System gespeichert werden müssen, wie auch das hohe Maß an Interaktion, dass den Nutzern zur Verfügung steht.

Der Unterricht in Mikroskopischer Anatomie (auch: Histologie) ist seit dem späten 18. Jahrhundert einem Prozess kontinuierlicher Überprüfung und Restrukturierung unterworfen [76, 102]. Besonders beeinflusst wurde dieser Prozess durch die Entwicklung virtueller Mikroskope gegen Ende des 20. Jahrhunderts, die vor allem durch den Bedarf an histopathologischem Konsiliar motiviert wurde [170].

Der Begriff „virtuelles Mikroskop“ ist nicht eindeutig definiert und wird in starker Abhängigkeit zum Kontext der Fragestellung eingesetzt. Erstmals wurde der Begriff im Jahr 1997 verwendet, als *Grimes et al.* eine Möglichkeit demonstrierten, mit der ein Roboter-gesteuertes Mikroskop, an dem eine digitale Kamera angebracht war, aus der Ferne bedient werden konnte [73]. Diese Anwendung basierte auf den Anforderungen von Pathologen, die sich den bedarfsorientierten Zugriff auf externes Expertenwissen wünschten, und ist ein Beispiel für Telepathologie[1]. Historisch gesehen werden zwei verschiedene Arten der Telepathologie unterschieden. Es wurde zum einen von dynamischer Telepathologie gesprochen, wenn der Experte aktiv an der Steuerung des Mikroskops zur Auswahl des Bildausschnitts beteiligt war. Im Gegensatz dazu stand die statische Telepathologie, in der dem Experten lediglich unterschiedliche, im Vorfeld vom Untersucher digitalisierte Aufnahmen zur Befundung vorgelegt wurden [3].

Im weiteren Verlauf des Jahres 1997 definierten *Ferreira et al.* ein virtuelles Mikroskop als ein Softwaresystem, das eine realistische digitale Emulation eines leistungsfähigen Lichtmikroskops bietet. Sie beschrieben eine Client-Server-basierte Architektur, die im Vorfeld digitalisierte, histopathologische Präparate bereitstellen konnte [62]. Nach nur einem Jahr erweiterten die Autoren ihre Definition und ergänzten sie um „Funktionen, die mit einem konventionellen Lichtmikroskop nicht möglich sind“ [3].

Im Rahmen dieser Entwicklung erfolgte der Transfer von einer konsiliarischen Telepathologie hin zu curricularen Veranstaltungen an Universitäten. Studien aus den Jahren 1999 bis 2001 zogen aus verschiedenen Fachbereichen Vergleiche zwischen virtuellen Mikroskopen und herkömmlichen Lichtmikroskopen [74]. Unterschiedliche Anwendungsszenarien von CD-ROM-basierter Bildverteilung bis hin zu den typischen Client-Server-Architekturen lassen sich in den folgenden Jahren dokumentieren [18, 75].

Der Funktionsumfang virtueller Mikroskope wurde ausgebaut und überschritt die Leistungsfähigkeit herkömmlicher Lichtmikroskope erstmals dokumentiert im

[1] Der amerikanische Pathologe Ronald S. Weinstein [48] führte den Begriff der Telephatologie 1986 ein [170].

Jahr 1998. Zu diesem Zeitpunkt wurden erste Annotationen in Form von eingebetteten Webseiten eingeführt [3]. Später folgten weitere, teils modifizierbare Annotationen, die von einfachen geometrischen Formen über Texte und Verknüpfungen zu anderen virtuellen Präparaten oder Ressourcen im Internet reichen [64].

Motiviert durch die stetige Weiterentwicklung der Möglichkeiten virtueller Mikroskope führten *Rojo et al.* im Jahr 2006 eine neue Begriffsbestimmung für virtuelle Mikroskope ein. Sie definierten ein virtuelles Mikroskop als ein Konzept, das verschiedene Aspekte von der Bildakquisition bis zu darstellenden Systemen umfasst [138]. Sie argumentierten, dass der korrekte Terminus „digitales Mikroskop“ lauten müsse, da die digitalen Bilder nicht weniger real seien als die von einem konventionellen Lichtmikroskop erzeugten. Diese Argumentation hat sich nicht durchgesetzt; in der Literatur wird bis heute von virtuellen Mikroskopen gesprochen.

Aufgrund des kontinuierlich wachsenden Funktionsumfangs virtueller Mikroskope, entschieden sich in den folgenden Jahren viele Universitäten für einen Wechsel von der konventionellen Lichtmikroskopie zur virtuellen Mikroskopie [47, 98]. Gleichzeitig begann eine Diskussion in den medizinischen Fakultäten über die Vor- und Nachteile virtueller Mikroskope [40].

So reduzieren virtuelle Mikroskope die Zeit für den curricularen Unterricht sowohl in Histologie als auch Pathologie [51] und steigern die Lerneffizienz wie auch die bessere Zusammenarbeit unter Studierenden [26]. Nach *Pratt et al.* können sie herkömmliche Mikroskope nicht ersetzen [132], vor allem aber nicht die Grundlagen im praktischen Umgang mit Mikroskopen vermitteln. Sie können daher lediglich unterstützend eingesetzt werden.

Aus Sicht der Dozierenden besitzt der konzeptionelle Transfer virtueller Mikroskopie in die Elektronenmikroskopie[2] ein außerordentliches Potential für Studierende der Medizin, da die Verfügbarkeit erforschbarer elektronenmikroskopischer Aufnahmen hierdurch sehr stark erhöht werden kann. Gewebe und ihre ultrastrukturellen Elemente können so nahtlos miteinander verknüpft werden [85, 105]. Zum Zeitpunkt dieser Arbeit ist die Erzeugung großflächiger elektronenmikroskopischer Meander-Scans häufig noch sehr zeitaufwendig und komplex, da z.B. die Teilbilder meist noch manuell zusammengesetzt werden müssen.

Neben der konzeptionellen Weiterentwicklung war die fortschreitende Hardware-Entwicklung moderner Computersysteme, sowie die Verbreitung von Breit-

[2] Auflösungsvermögen: menschliches Auge ca. $2 \cdot 10^{-4}$ m, Lichtmikroskop ca. $2 \cdot 10^{-7}$ m, Elektronenmikroskop ca. $2 \cdot 10^{-10}$ m [96]

band-Internetverbindungen wegbereitend für eine globale Verbreitung virtueller Mikroskopie. Sowohl Anwendungsszenarien als auch die Leistungsfähigkeit virtueller Mikroskope wurden ausgebaut. Besondere Relevanz haben Präparatescanner, die Präparate in mehr als einer Fokusebene digitalisieren, was einen deutlich höheren Speicherbedarf bedeutet, gleichzeitig aber auch neue Funktionen zulässt.

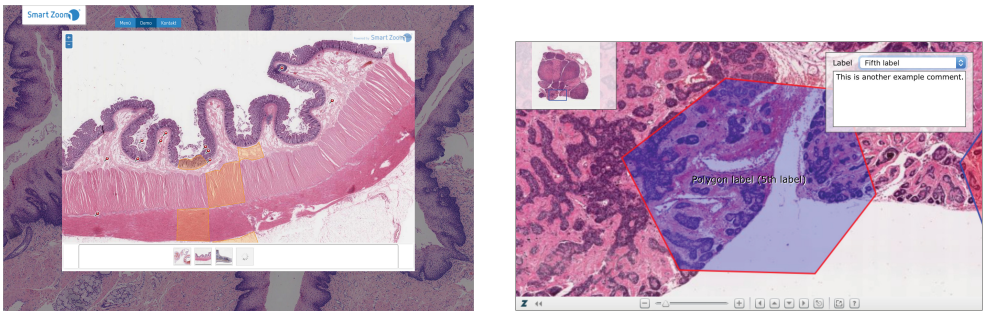
Virtuelle Mikroskope ermöglichen eine Nutzung wissenschaftlicher Expertise an vielen Orten. Gleichzeitig können im Bereich medizinischer Ausbildung einer breiten Masse von Studierenden und Wissenschaftlern digitalisierte Präparate zur Verfügung gestellt werden, zu denen sie ohne diese Entwicklung kaum Zugang hätten. Insbesondere Färbungen mit fluoreszierenden Farbstoffen sind aufgrund ihrer begrenzten Haltbarkeit, die sich durch kontinuierliches Verblassen unter Lichteinfall zusätzlich mit jeder Betrachtung reduziert, in den Fokus der Scanner-Hersteller gerückt. Weiterhin sind diese Präparate der Immunfluoreszenz äußerst kostenintensiv in der Herstellung und damit für Gruppen in Klassen- oder Semestergröße ungeeignet.

Ein Peer-to-Peer-System ist nach den bisher eingesetzten Client-Server-Varianten der virtuellen Mikroskopie eine leistungsstarke Innovation, da es bei hohem Datenaufkommen und ebensolchen Nutzerzahlen sehr gut skaliert. Es erweitert überdies das klassische virtuelle Mikroskop, indem es dem datenintensiven Bereich der Lebenswissenschaften mit einer kooperativen Plattform begegnet, die gleichzeitig als interdisziplinäre Lernumgebung genutzt werden kann.

1.2 Stand der Technik virtueller Mikroskope

Nationale und internationale Medizinische Fakultäten setzen in der histo(patho)logischen Ausbildung auf virtuelle Mikroskopie; sei es als Ergänzung zur curricularen Lehre oder als vollständigen Ersatz für konventionelles Mikroskopieren. In der Regel werden dazu Internet-basierte Client-Server-Lösungen verwendet, die größtenteils auf Silverlight, HTML (z.B. SmartZoom[3]) [171] oder Flash (z.B. zoomify Framework[4]) [121] basieren (vgl. Abb. 1.1). Der Anwendungsbereich beschränkt sich üblicherweise auf Zytologie, Histologie und Histopathologie. Vereinzelt finden sich auch Onkologie und Hämatologie als Themenfelder. Vielfach sind vorgefertigte Beschriftungen zu finden und Werkzeuge zur Erstellung eigener Annotationen vorhanden (z.B. PATE [27], NYU Virtual Microscopy [127] oder

[3]SmartZoom Webseite: `http://www.smartzoom.com/de/system/`
[4]zoomify Webseite: `http://www.zoomify.com`



(a) SmartZoom (b) zoomify

Abbildung 1.1: Darstellung der Benutzeroberfläche von zwei häufig verwendeten Lösungen bzw. Frameworks zur Konstruktion virtueller Mikroskope.

HistoViewer[5] [144]). Existierende Systeme stellen in der Regel ein leicht zugängliches Nachschlagewerk für ausgewählte Lehrbuchpräparate dar, z.B. die Mainzer Histo Maps [171]. Die Präparatedatenbanken der bisher genannten Anwendungen beinhalten wenige Dutzend bis einige Hundert Datensätze. Lediglich die School of Medicine der New York University (NYU), die eine Lösung über die Google-Maps API zur Darstellung von histologischen Präparaten [127] verwendet, kann auf einen Datenbestand von rund 1.500 Datensätzen[6] zurückgreifen.

Bis auf die Anwendung HistoViewer bietet keine Lösung die native Möglichkeit mehrere Präparate vergleichend zu betrachten. HistoViewer ermöglicht die einfache Betrachtung von zwei Präparaten (vgl. Abb. 1.2).

Ein überwiegend im Forschungsbereich angesiedelter Ansatz virtueller Mikroskopie wird vom OME[7] (Open Mircroscopy Environment) verfolgt. Hier geht es primär um ein plattform- und herstellerunabhängiges Dateiformat [68] zur quantitativen Analyse von Bilddaten in den Lebenswissenschaften [157] mit dem Ziel der Austauschbarkeit und Datensicherheit.

Ein wesentlicher Aspekt der Interaktion ist das Anfertigen und der Austausch von Annotationen zwischen den Nutzern. Dies wird von den vorgestellten Anwendungen jedoch häufig nur fragmentär umgesetzt und Annotationen lediglich als statische Lehrinhalte oder private Beschriftungen verwendet. Die Bereitstellung unterschiedlicher Werkzeuge für die Beschriftung von Datensätzen ermöglicht hoch detaillierte Konturen. Die NYU Virtual Microscopy hingegen erlaubt

[5] HistoViewer Webseite: `http://anat-microscopy.ana.au.dk`
[6] Stand: 23. November 2015, 12:54 Uhr
[7] OME Webseite: `http://www.openmicroscopy.org/`

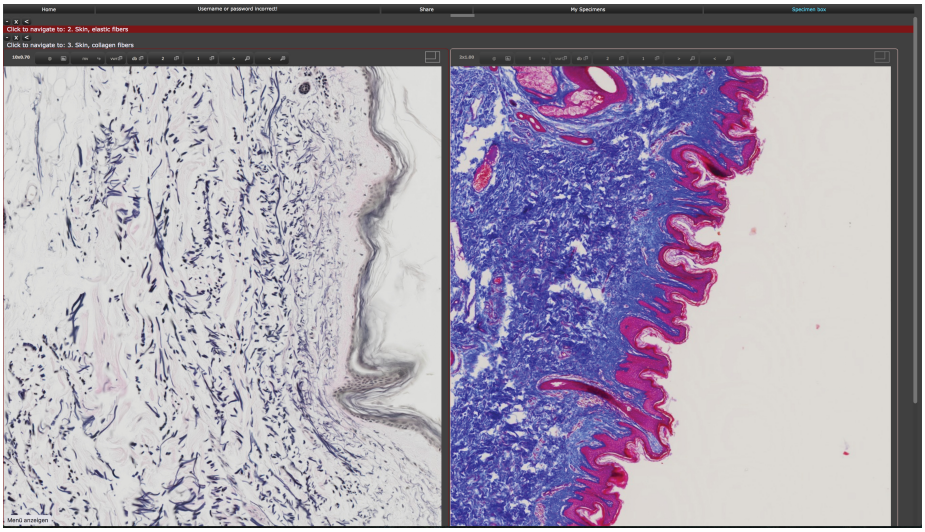


Abbildung 1.2: Der HistoViewer [144] ermöglicht die vergleichende Betrachtung von zwei Präparaten.

lediglich das Platzieren von Markierungen an beachtenswerten Strukturen. In keiner der erwähnten Lösungen können Annotationen untereinander oder auch mit anderen Datensätzen verknüpft werden. Wertvolle, kontextübergreifende Informationen werden so nicht erfasst. Jedoch ist genau diese stetig wachsende Sammlung an Verknüpfungen ein anzustrebendes Ziel einer multifokalen Wissendarbietung.

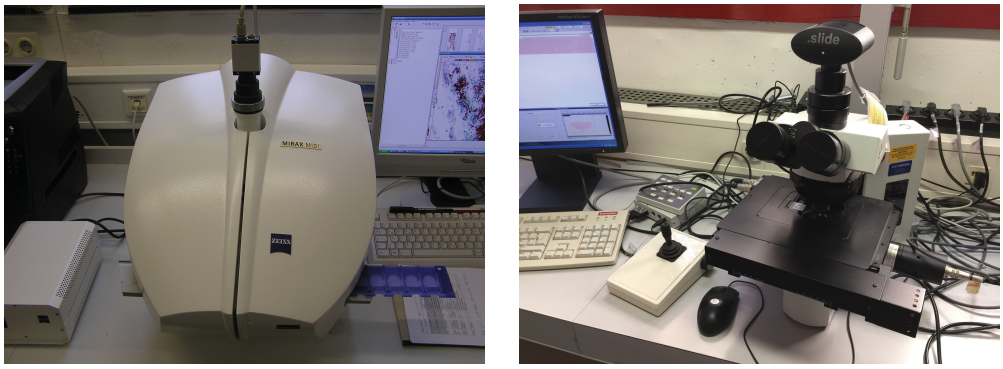
Neben Annotationsfunktionen bieten die eingangs dieses Abschnitts erwähnten Lösungen eine Navigation in 3 Ebenen durch die digitalisierten Präparate, wie sie von Geo-Applikationen (z.B. GoogleMaps) bekannt sind. Eine darüber hinaus gehende, native Unterstützung für mehrere Fokusebenen bietet lediglich zoomify [121]. Diese Navigationsmöglichkeiten lassen sich nach entsprechender Erweiterung auch für die Navigation durch die Zeitachse radiologischer Aufnahmen verwenden.

Betrachtet man die Verfügbarkeit der einzelnen Lösungen, so erfordern alle Systeme zwangsläufig einen zentralen Server zur Datenverteilung und Organisation in den Lehrinstitutionen und unterstützen je nach verwendeter Technik auch mobile Endgeräte. Andere Datenformate aus den Bereichen Radiologie, Chirurgie oder Orthopädie werden für einen interdisziplinären Ansatz von keiner Lösung dezidiert betrachtet.

1.2.1 Digitalisierung der Präparate

Histologische Präparate können mit speziellen Scannern oder auch mit Scanner-Mikroskopen (vgl. Abb. 1.3) digitalisiert werden. Slide Scanner arbeiten unter konstanten Beleuchtungsbedingungen zur schnellen und automatischen Sequenz-Digitalisierung vieler Objektträger, wohingegen Scanner-Mikroskope eher für abweichend große Objektträger oder kleine Ausschnitte mit sehr hochauflösenden Objektiven geeignet sind. Speziell diese hochauflösenden Objektive erfordern mitunter Öl als optisches Medium, das in den üblicherweise geschlossenen Systemen von Scannern nicht verfügbar ist, da ihre Mechanik sehr empfindlich auf Verschmutzungen reagiert.

Grundsätzlich werden zur Digitalisierung von histologischen Präparaten CCD Chips in digitalen Kameras verwendet, die mit einer entsprechend hochauflösenden Optik sich überlappende Teilbilder fester Größe des Objektträgers aufnehmen. Dabei variiert das Auflösungsvermögen je nach Gerät zwischen 0,12 und 0,45 μm je Pixel für ein 40x Objektiv [138]. Die Überlappung ist durch die mechanische Beanspruchung des Gerätes und der sich daraus ergebenden Ungenauigkeit bei der Digitalisierung mit stark vergrößernden Objektiven unabdingbar. Ein Stitching-Prozess entfernt die Überlappung nach der Digitalisierung und richtet die Teilbilder möglichst passend aus. Dessen ungeachtet entstehen auf diese Weise, vor allem bei kontrastarmen Aufnahmen, durch unterschiedliche Feinfo-



(a) Mirax Midi Slide Scanner des Instituts für Anatomie der Westfälischen Wilhelms-Universität, Münster. Foto: A. Barbian

(b) Olympus Scanning Microscope BX51 des Instituts für Anatomie I der Heinrich-Heine-Universität Düsseldorf. Foto: A. Barbian

Abbildung 1.3: Unterschiedliche Scanner zur Digitalisierung histologischer Präparate.

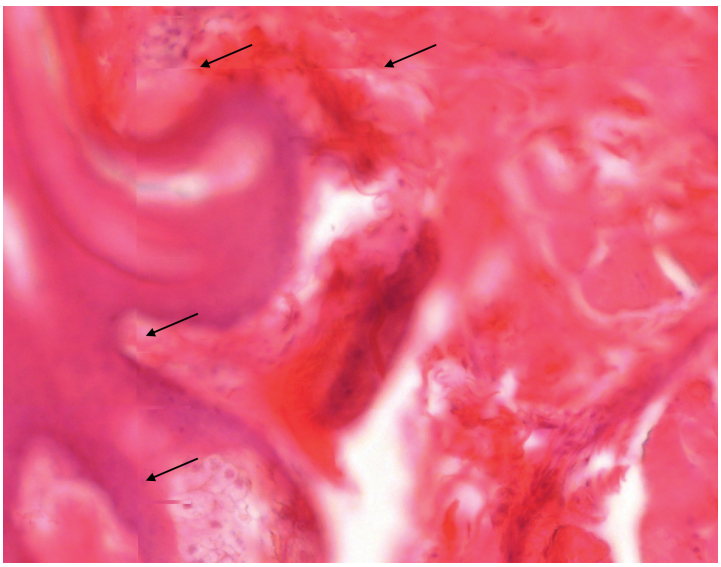


Abbildung 1.4: Beispiel eines Stitching-Artefakts (schwarze Pfeile) durch unterschiedliche Feinfokussierung in einem HE gefärbten Präparat einer menschlichen Lippe mit geringem Kontrast und inhomogener Adhäsion. Foto: A. Barbian

kussierung der Teilbilder mitunter Artefakte, die von der Scannersoftware nicht beseitigt werden können (vgl. Abb. 1.4).

Objektträger der labordiagnostischen Routine sind nach DIN ISO 8037-1 76 x 26 mm groß. Häufig wird an einer Seite ein etwa 20 mm breiter Bereich zur Beschriftung verwendet, der von den meisten Scannern ebenfalls gelesen werden kann. Ein vollständiger Scan des Gewebebereiches eines Präparats umfasst bei den meisten Geräten daher eine Fläche von 50 x 25 mm [138]. Unter Berücksichtigung der Auflösung ergibt sich je nach Scanner eine Bildgröße von bis zu 417.000 x 208.000 Pixel ($\approx$ 82 GigaPixel).

Die ausgerichteten Teilbilder werden in der Regel JPEG codiert (mit oder ohne Kompression) in einem proprietären Dateiformat gespeichert und können von der passenden Software des Herstellers in nahezu beliebigen Ausschnitten angezeigt werden. Dazu muss jeweils der passende Ausschnitt in der Datei gefunden, entpackt und neu aus den einzelnen Teilbildern zusammengesetzt werden.

Zusätzlich können viele Scanner die Aufnahmen in mehreren Ebenen fokussieren – analog einem klassischen Lichtmikroskop, das eine stufenlose Fokussierung ermöglicht [138]. Entsprechend des möglichen Fokusbereichs, der gerätespezifischen Fokusschrittlänge und der Schnittdicke des Präparates sind bis zu 15 wei-

tere Fokusebenen möglich, was das Datenvolumen in der genannten Vergrößerung auf bis zu 1,2 TeraPixel erhöht. Kleinere Vergrößerungen werden meist algorithmisch interpoliert und nicht separat digitalisiert. Häufig werden hierzu Potenzen von Zwei verwendet, um die neue Vergrößerung mit lediglich einem Viertel der ursprünglichen Größe zu berechnen.

Die unkomprimierte Größe S in Bits eines zu speichernden Präparates p, mit gegebener Ausdehnung $(w \cdot h)$ in μm, lässt sich für jeden Scanner anhand von Gleichung 1.1 aus der Anzahl an Farbkanälen c, ihrer Farbtiefe bpc in Bit je Kanal, der Auflösung d, der Anzahl der digitalisierten Fokusebenen l und der Menge abrufbarer Vergrößerungen z ermitteln.

$$S(p) = c \cdot bpc \sum_{n=1}^{z} \left(\frac{1}{2^{n-1}} \cdot \frac{w_p h_p}{d} \cdot l \right) \tag{1.1}$$

1.2.2 Virtuelle Mikroskopie an der HHU

Der Autor verfügt durch die Entwicklung einer neuartigen Client-Server-basierten Lösung an der Westfälischen Wilhelm-Universität Münster über fundierte Erfahrung im Umfeld virtueller Mikroskope. Bereits hier konnte ein virtuelles Mikroskop für mehr als zweihundert Studierende unter Verwendung ressourcenarmer, nativer ThinClients ohne Erfordernis eines leistungsfähigen Backends realisiert werden [84]. Im Jahr 2011 realisierte er die erstmalige, interdisziplinäre Übertragung auf die Elektronenmikroskopie. Damit wurde eine Verknüpfung von histologischen und ultrastrukturellen Aufnahmen anhand von benutzergenerierten Beschriftungen etabliert [85].

Kurz darauf begann im Rahmen dieser Dissertation die Konzeption einer verteilten Architektur des virtuellen Mikroskops *Omentum*[8], das in der folgenden Zeit anhand unterschiedlicher Anwendungsszenarien zu einer interaktiven Lehr- und Lernplattform erweitert wurde. So wurde zunächst die Integration radiologischer Aufnahmen fokussiert, um diesen großen, medizinischen Arbeitsbereich der Bildgebung für Omentum zu erschließen.

Im Hinblick auf die praktische Verwendung zielt Omentum auf den fokussierten Bereich einer Umgebung für eTeaching und eLearning und nicht auf ein zertifiziertes Medizinprodukt zur qualitativen Befundung ab, weshalb die stren-

[8] omentum, lat.: Netz. In der Anatomie wird der Begriff verwendet um das große und kleine Netz in der Bauchhöhle zu beschreiben, das u.a. lymphatische Gefäße und Zellen zur Immunabwehr im Bauchraum enthält.

gen Richtlinien für eine Zertifizierung nach dem MPG[9] [33] nicht anzuwenden sind. In diesem Lehr- und Lernkontext bietet die Darstellung von Bilddaten auf einem Monitor und die Möglichkeit des Dialogs mit anderen Benutzern eine höhere Identifikationssicherheit für sichtbare Strukturen als die Betrachtung durch ein herkömmliches Lichtmikroskop, bei der man trotz möglicherweise anwesender Experten keine unmittelbare, individuelle Supervision erhält.

Ein Alleinstellungsmerkmal von Omentum ist die parallele Darstellung mehrerer Präparate, die lediglich durch die Größe des Bildschirms beschränkt ist und für die differentialdiagnostische Betrachtung von hohem edukativen Nutzen ist. Benutzer können so mehrere aufbereitete Datensätze an ihrem Gerät parallel, vergleichend betrachten, mit Annotationen beschriften und diskutieren.

Die erstellten Annotationen repräsentieren jeweils Objekte, die unabhängig von den Bilddaten, im System gespeichert werden. Dabei ist jede Annotation dezidiert einem Datensatz zugeordnet und verwendet Positionen im Koordinatenraum dieses Datensatzes um eindeutig eine Markierung zu platzieren. Zur Reduktion der Komplexität während der Verarbeitung der Annotationen werden die erstellten Objekte als vereinfachte Polygone, die durch Tupel von Koordinaten definiert sind, in Textform in einer Datenbank gespeichert. Die Freigabe der eigenen Annotationen für andere Benutzer ermöglicht es allen Nutzern mit diesen Beschriftungen zu arbeiten, voneinander zu lernen und einen sukzessiv wachsenden Datenbestand im System zu generieren.

Der intensive, fachliche Dialog wird sowohl zwischen mehreren Benutzern an einem Arbeitsplatz genutzt (z.B. in Seminaren), wie auch durch die Möglichkeit Fragen und Anmerkungen online zu hinterlassen, die von der Community diskutiert werden können. Die Besonderheit ist hier der sichtbare und unmittelbare Bezug zu einem abgebildeten Präparat, das nicht erst in einem Text beschrieben werden muss.

Neue Bilddaten können Benutzer in der Regel nicht in das System einspeisen. So sollen Benutzer daran gehindert werden ihre medizinischen Unterlagen für unqualifizierte Befunde oder Ratschläge mit anderen Benutzern zu teilen. Daher ist es nur bestimmten Entitäten im Netzwerk erlaubt neue Bilddaten einzuspeisen. Für die Konstruktion des in Kapitel 3 vorgestellten Overlays spielt diese Einschränkung eine wesentliche Rolle, da sie einen großen Einfluss auf die Änderungsfrequenz und den Aktualisierungsbedarf der Bilddaten hat, ohne dass sie

[9]MPG: Medizinproduktgesetz, entspricht der deutschen Umsetzung der europäischen Richtlinien für Medizinprodukte (93/42/EWG), Diagnosegeräte (98/79/EG) und aktive implantierbare Geräte (90/385/EWG).

die Arbeit mit bereits zur Verfügung gestellten Datensätzen einschränkt.

Die Analyse der Interaktionsmuster in Omentum ist, vor dem Hintergrund einer Untersuchung des Lernverhaltens, ebenfalls in den Blickpunkt gerückt. Aus diesem Grund wurden spezialisierte Algorithmen zur Speicherung der Benutzeraktionen entwickelt, die eine Grundlage für die im Nachgang entwickelte Synthese neuer Benutzerprofile anhand gespeicherter Aktionen bildeten. Ein Nebenprodukt dieser Algorithmen ist die individuelle Visualisierung der Arbeit mit dem Präparat für jeden Nutzer.

Zukünftig könnte über ein Ranking-System ähnlich zu vorhandenen Systemen, z.B. bei Stack Overflow[10], das Expertenlevel eines Benutzers und damit der Status in Omentum bestimmt werden. In Abhängigkeit des Rankings erhalten die Annotationen, Kommentare und Hinweise eines Nutzers entsprechende Relevanz.

1.3 Zielsetzung

Auch wenn viele verschiedenartige Architekturen für verteilte Systeme existieren, werden die Anforderungen des Anwendungskontextes von diesen nicht umfassend abgedeckt. Insbesondere sind hier die großen Datenvolumina von bis zu 50 GB pro virtuellem Präparat, die selbst mit einer adäquaten Sammlung von mehreren zehntausend Präparaten sehr schnell einige PetaByte erreichen, zu adressieren. Weiterhin stellt die äußerst heterogene, von allen Benutzern kooperativ erstellte Wissenssammlung besondere Anforderungen an eine Volltextsuche in verteilt gespeicherten Daten.

Im Rahmen dieser Arbeit wird mit Omentum das erste verteilte System entwickelt, das sich auf diese Anwendungsklasse fokussiert. Dabei soll über eine geeignete Datenaufbereitung und -partitionierung insbesondere ein schonender Umgang mit den verfügbaren Netzwerkressourcen bei effizienter Übertragung der angeforderten Bilddaten realisiert werden. Effektive Replikationsmechanismen sollen von der Datenpartitionierung ebenso profitieren und gleichzeitig die Abhärtung des Systems gegen Ausfälle vieler Knoten unterstützen. Das System ist dabei so zu organisieren, dass jederzeit der Ausfall einzelner Knoten oder Super-Peers kompensiert werden kann. Benutzer sollten eindeutig identifiziert werden können, da ihnen die Urheberschaft einzelner Annotationen zuzuordnen ist. Weiterhin ist die Kommunikation zwischen einzelnen Peers mit einem angemessenen Schutz vor unberechtigtem Zugriff zu versehen. Zur Bewertung von einzelnen Peers, Daten-

[10] Webseite: `http://www.stackoverflow.com`

partitionen oder Benutzern soll über eine Verbindung zentraler Knoten zu einem Super-Peer-Overlay eine effektive Datenaggregation ermöglicht werden.

1.4 Beiträge

Die wissenschaftlichen Beiträge dieser Arbeit wurden bidisziplinär als Forschungsbeiträge zu Konferenzen und Tagungen der Informatik und Medizin veröffentlicht.

1. Die Verwendung von Zufallsgraphen zur Konstruktion des Peer-to-Peer-Overlays für ein virtuelles Mikroskop, das eine effiziente, verkehrslose Datenlokalisation ermöglicht und optimierte Replikationsstrategien erlaubt, wurde erstmalig auf der DAIS 2013 publiziert [83].

2. Eine optimierte Replikationsstragie, die speziell die Anforderungen einer verteilten, interdisziplinären Lehr- und Lernplattform berücksichtigt, und ein effizientes zweistufiges Verfahren zur Pfadkompression in Zufallsgraphen wurde im Rahmen der IDCS 2015 vorgestellt [12].

3. Ein plattformunabhängiges User-Space Dateisystem (USpFS), das speziell die Anforderungen der Daten in einem verteilten virtuellen Mikroskop berücksichtigt und obligatorische Funktionen zur Sicherung der Datenintegrität bereitstellt, wurde im Rahmen der PDCAT 2015 präsentiert [13].

4. Die Analyse von Benutzerprofilen sowie die Verknüpfung von benutzergenerierten Inhalten und deren Verwendungsmöglichkeiten für die medizinische Aus-, Fort- und Weiterbildung wurden auf der Jahrestagung der Gesellschaft für Medizinische Ausbildung (GMA) 2015 vorgestellt [11].

1.5 Gliederung

Kapitel 2 beschreibt zunächst die Daten, die im Netzwerk gespeichert werden sollen und zeigt eine Strategie für eine effiziente Partitionierung, die den benötigten Key-Space im Peer-to-Peer-Overlay deutlich reduzieren wird.

In Kapitel 3 wird das Konstruktionsprinzip des Omentum Overlays mit seinen Basis-Operationen *store*, *get*, *search*, *join* und *leave* vorgestellt. Weiterhin wird eine Aggregationsstrategie für die lokale Leistungsbewertung im System eingeführt.

Das Replikationsmodell wird detailliert in Kapitel 4 dargestellt. Dabei werden insbesondere die lokale Autonomie sowie die Fehlerbehandlung betrachtet und

das Overlay um die Operation *crash* zur Erholung von Systemausfällen erweitert. Ferner werden Strategien für ein unaufgefordertes Laden von Daten anhand von heuristischen Verfahren und die Anbindung mobiler Clients vorgestellt.

Kapitel 5 befasst sich mit der Entwicklung eines speziell für Omentum entworfenen Dateisystems, das sehr viele kleine Dateien unterstützt. Der Schwerpunkt liegt hier auf der flexiblen und modularen Architektur sowie der Struktur der Metadaten.

Die Sicherheitsaspekte für ein praxistaugliches Peer-to-Peer-System werden in Kapitel 6 beschrieben, wobei die Authentifizierung und die Umsetzung von Zugriffsberechtigungen in einem verteilten System gesondert betrachtet werden.

Abschließend werden in Kapitel 7 die vorgestellten Strategien und Konzepte zusammengefasst und ein Ausblick auf weiterführende Arbeiten gegeben.

KAPITEL 2

Datentypen

Die Datensätze, die mit Omentum den Benutzern zur Verfügung gestellt werden sollen, stellen mit Größen von bis zu 50 GB eine besondere Herausforderung an die Aufbereitung der Daten. Daher erläutert dieses Kapitel das entwickelte Verfahren für eine pyramidenförmige Speicherung der Daten als JPEG-Kacheln und erweitert dieses Konzept zusätzlich auf Datensätze von 3D-Objekten, wie sie in der radiologischen Routine häufig verwendet werden. Vor allem für die spätere Verteilung im Peer-to-Peer-Netzwerk ist diese Aufbereitung und Unterteilung der Daten von Bedeutung, da so nur Teile der jeweiligen Datensätze übertragen oder repliziert werden müssen und nicht der vollständige Datensatz mit mehreren Gigabyte.

Gleichzeitig sind die von den Benutzern erzeugten Daten wie z.B. Beschriftungen (Annotationen), Benutzerprofile oder aufgezeichnete Interaktionsmuster hochdynamisch und müssen bei Änderungen an den Daten stets auf allen replizierenden Knoten aktualisiert werden.

2.1 Überblick

Neben der Zeit, die für die Datenspeicherung oder -lokalisation im entwickelten Peer-to-Peer-System aufgewendet werden muss, spielt bei der vorliegenden Datensatzgröße von mehreren Gigabyte vor allem die Zeit der Datenaufbereitung eine besondere Rolle. In der klinischen Diagnostik ist die Echtzeitverarbeitung von Daten äußerst wichtig. Zum einen sind diagnostische Befundungen häufig zeitkritisch, z.B. Schnellschnitte während einer größeren Operation zur Abklärung des weiteren Vorgehens, und zum anderen ändert die Konvertierung des Eingangsmaterials möglicherweise den Grad an verfügbaren Details, was gegen eine

Zertifizierung nach MPG[11] [33] spricht, die für klinische Systeme obligatorisch ist. So werden für das befundende Fachpersonal alle Ansichten aus dem Datensatz in Echtzeit berechnet. Dies ist insofern kein Nachteil, da in der klinischen Routine normalerweise nicht mehrere Experten zeitgleich den selben Datensatz befunden.

Für den in Kapitel 1.1 erwähnten Kontext der Ausbildungssituation hingegen wäre ein sehr hoher Rechenaufwand erforderlich um für viele Tausend Lernende alle Ansichten in Echtzeit aus den gespeicherten Daten zu berechnen. Aus diesem Grund ist eine Vorbereitung der Daten nicht nur sinnvoll, sondern zwingend notwendig. Der Bedarf für die Echtzeit-Verfügbarkeit ist in diesem Kontext ebenfalls nicht gegeben, da die Bilddaten ohnehin im Vorfeld gewonnen werden und nicht direkt für den Unterricht zur Verfügung stehen müssen.

Für Omentum wurde zu diesem Zweck ein Medienkonverter implementiert, der die Eingangsdaten unabhängig von ihrem Format verfügbar macht. Die Implementierung verwendet die *Bio-Formats* Bibliothek [166] des OME Projektes [111], die es ermöglicht 142 Formate[12] unterschiedlicher Hersteller zu verarbeiten.

Neben den überwiegend statischen Bilddaten, die im Netzwerk verteilt gespeichert werden sollen, gibt es in Omentum eine Gruppe sehr dynamischer, benutzergenerierter Daten. So ist es jedem Benutzer des Systems erlaubt Beschriftungen (Annotationen) an den Datensätzen vorzunehmen. Diese Annotationen werden unabhängig von den Datensätzen verteilt gespeichert und müssen bei Änderungen entsprechend aktualisiert werden.

2.2 Aufbereitung der statischen Bilddaten

Im folgenden wird ein Überblick über das entworfene Datenformat für die Speicherung im Peer-to-Peer-Netzwerk gegeben. Dabei wird insbesondere die Übertragungslast betrachtet, die von der gewählten Größe der einzelnen Teilbilder abhängt.

Die Eingangsdaten werden zunächst in der maximalen Vergrößerung aus den proprietären Herstellerformaten extrahiert, anschließend aufgeteilt und in Form mehrerer Kacheln gespeichert. Die Dimensionen der Kacheln haben dabei einen entscheidenden Einfluss auf die später im Netzwerk erzeugte Last. Um ein optimales Verhältnis zwischen der Anzahl der Anfragen und der Menge der zu über-

[11] MPG: Medizinproduktgesetz, entspricht der deutschen Umsetzung der europäischen Richtlinien für Medizinprodukte (93/42/EWG), Diagnosegeräte (98/79/EG) und aktive implantierbare Geräte (90/385/EWG).

[12] Stand: November 2015

tragenden Daten zu ermitteln, wurden unterschiedliche Kachelgrößen zwischen 64 und 2048 Pixel Kantenlänge für verschiedene Auflösungen betrachtet. Speziell wurden dazu die angeforderten Nutzdaten und die zugehören Metadaten, die während der Übertragung z.B. in Form von Headern o.ä. entstehen, für jede Kantenlänge und unterschiedliche Auflösungen betrachtet (vgl. Tab. 2.1 und Abb. 2.1).

Die für diesen Vergleich übertragenen Daten hatten eine durchschnittliche Nutzlast von ca. 0,55 Byte/Pixel. Aufgrund des geringeren Overheads an Metadaten und der deutlich geringeren Anzahl an Anfragen hat sich eine Kachelgröße von 256 Pixel Kantenlänge als optimales Format für die vorliegenden Daten erwiesen. Die Eingangsdatensätze werden daher in nicht-überlagernde Teilbilder mit einer Kantenlänge von 256 Pixel zerlegt. Um die spätere Berechnung für geringer aufgelöste Bildbereiche zu vermeiden, werden diese der Reihe nach für jede digitalisierte Fokusebene interpoliert (vgl. Algorithmus 2.1).

Kommerzielle Lösungen, die zur Darstellung von sehr großen Bildern fähig sind (z.B. WebScope von Aperio [8]), müssen zumeist neben der reinen Übertragung die Teilbilder erst aus den originalen Daten berechnen oder extrahieren und sind daher nicht für einen schnellen Wechsel der Präparate und intensive Interaktion ausgelegt [90]. *Jeong et al.* stellten 2010 ein System vor, das ausschließlich die benötigten Bilddaten stark komprimiert überträgt (*display aware*) und mit Hilfe einer leistungsfähigen GPU rendert. Omentum verwendet ein ähnliches Verfahren und überträgt lediglich ausreichend Bildinformationen, um das Fenster des Benutzers zu füllen. Eine leistungsstarke GPU wird für eine Anzeige der 2D-Daten nicht benötigt.

Die pyramidenartige Ablage von Teilbildern wird durch *Wang et al.* in Frage gestellt [167], da die Koordinaten einer Kachel, einem Quadtree entsprechend, an einer weniger stark aufgelösten Elternkachel orientiert sind. Zur Optimierung stellen die Autoren ein Verfahren vor, dass die Kacheln in n Gruppen der Größe 2^8 zusammenfasst und ermöglichen damit ein Auffinden der Kacheln in stark vergrößerten Bereichen in deutlich kürzerer Zeit.

In Omentum sind die einmal berechneten Kacheln für die Darstellung nicht voneinander abhängig. Jede erzeugte Kachel ist systemweit eindeutig anhand des Präparates, zu dem sie gehört, ihren x- und y-Koordinaten, der Vergrößerungsstufe z und der Fokusebene l identifizierbar. Daher lassen sich in der von Algorithmus 2.1 erzeugten pyramidenartigen Struktur (vgl. Abb. 2.2) die für eine Anzeige erforderlichen Dateien in der gewünschten Auflösung unmittelbar in $O(1)$ erreichen.

Tabelle 2.1: Tabellarische Übersicht der Verkehrsdaten für Kacheln zwischen 64 und 2048 Pixel Kantenlänge zur Bestimmung des optimalen Verhältnisses zwischen Nutz- und Metadaten.

	Window Size (Pixel)					
	1024 x 768			1280 x 1024		
Size	Req	Payload	Metadata	Req	Payload	Metadata
64	192	429312	10944	320	715520	18240
128	48	429216	2736	80	715360	4560
256	12	429216	684	20	715360	1140
512	4	572288	228	6	858432	342
1024	1	572288	57	2	1144576	114
2048	1	2289152	57	1	2289152	57

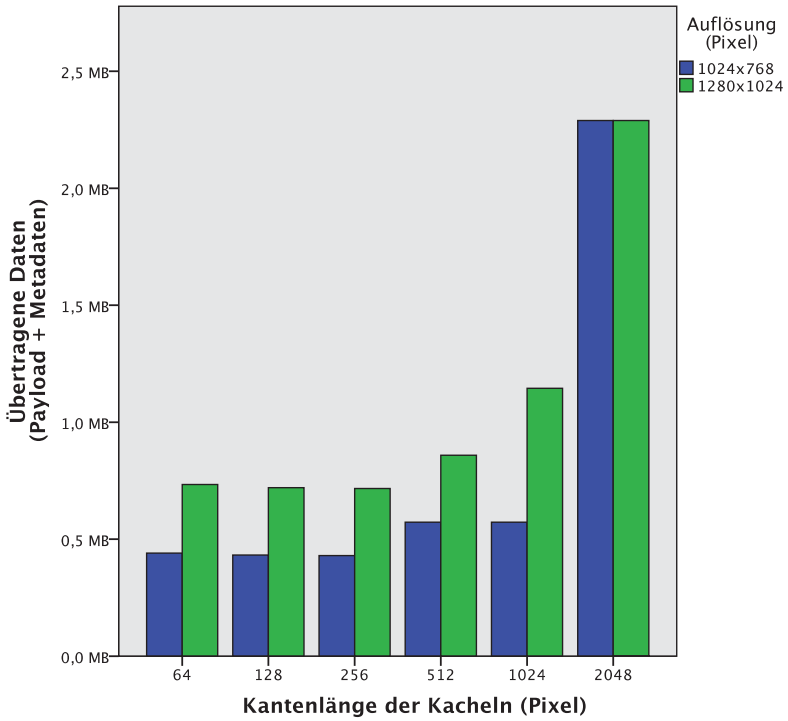


Abbildung 2.1: Darstellung der Verkehrsdaten für Kacheln zwischen 64 und 2048 Pixel Kantenlänge zur Bestimmung des optimalen Verhältnisses zwischen Nutz- und Metadaten.

Algorithmus 2.1 Restrukturierung der propriäteren Eingangsdaten in eine pyramidenartige Kachel-Struktur (vereinfachte Darstellung)

```
1:  z = 0;                                              ▷ aktuelle Vergrößerung
2:  l = 0;                                              ▷ aktuelle Fokusebene
3:  function INTERPOLATE(maxZoom, maxLayer)
4:      while z < maxZoom do
5:          z = z + 1;
6:          while l < maxLayers do
7:              path = CREATEDIRECTORY(z, l);
8:              while y < numTilesHeight do
9:                  while x < numTilesWidth do
10:                     s1 = SCALEIMAGETOHALF(2x, 2y, z − 1, l);
11:                     s2 = SCALEIMAGETOHALF(2x + 1, 2y, z − 1, l);
12:                     s3 = SCALEIMAGETOHALF(2x, 2y + 1, z − 1, l);
13:                     s4 = SCALEIMAGETOHALF(2x + 1, 2y + 1, z − 1, l);
14:                     i = COMBINESCALEDIMAGES(s1, s2, s3, s4);
15:                     SAVEIMAGE(i, path);
16:                     x = x + 1;
17:                 end while
18:                 y = y + 1;
19:             end while
20:             l = l + 1
21:         end while
22:     end while
23: end function
```

Möglich ist dies durch eindeutige Identifizierung einer Kachel im lokalen Dateisystem. Die dazu entwickelte Ordnung verwendet für jeden Datensatz einen Ordner entsprechend seiner ID, der wiederum Ordner für jede Zoomstufe enthält. Innerhalb der einzelnen Zoomstufen sind die Kacheln mit ihren x- und y-Koordinaten gespeichert. Gibt es mehr als eine Fokusebene, werden die Daten innerhalb einer Zoomstufe zusätzlich in Unterverzeichnissen organisiert (vgl. Abb. 2.3).

Die Dateigrößen innerhalb eines verarbeiteten Datensatzes liegen im Mittel bei 31.698 Byte (σ_n = 57,09) und verteilen sich wie in Abb. 2.4(a) exemplarisch dargestellt. Für die Analyse der Größenänderung eines Datensatzes durch die Aufbereitung wird der Datenbestand (n = 132) vor und nach der Umstrukturierung betrachtet (vgl. Abb. 2.5(a)). Jeder Datensatz wird durchschnittlich um den Faktor 1,87 (σ_n = 0,23) vergrößert. Die Eingangsgröße der Datensätze hat, wie das Diagramm in Abbildung 2.5(a) durch den annähernd linearen Verlauf

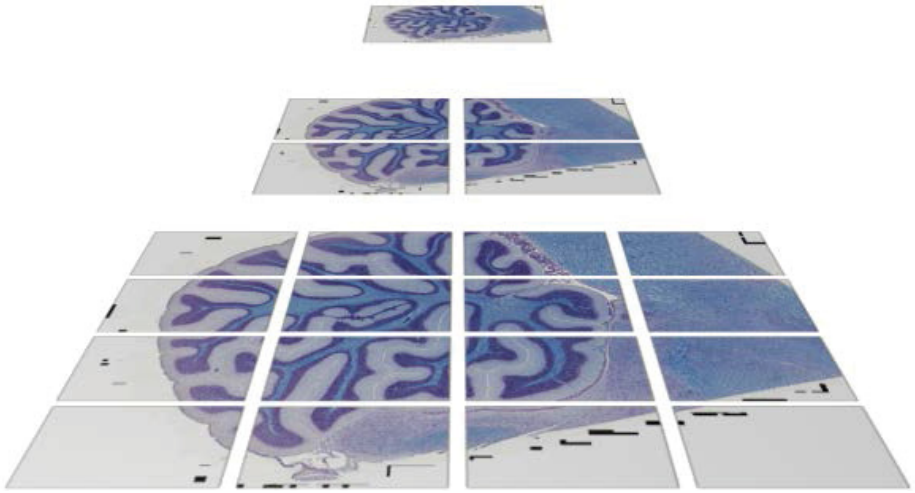


Abbildung 2.2: Darstellung der pyramidalen Struktur eines exemplarischen Datensatzes mit drei Zoomstufen und einer Fokusebene nach Verarbeitung durch den Medien-Konverter.

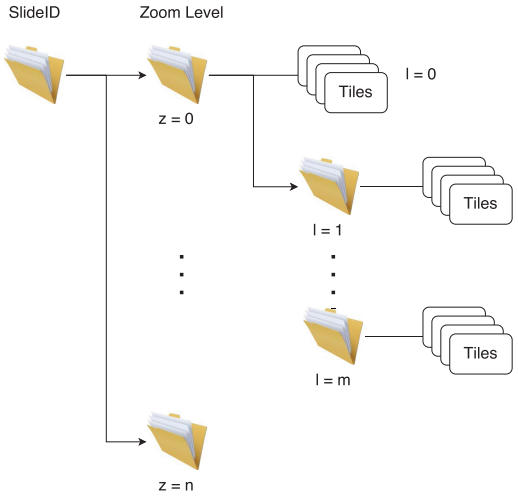


Abbildung 2.3: Ansicht des Dateisystems für die pyramidale Datenstruktur nach der Aufbereitung durch den Medienkonverter mit separaten Ordnern für die Zoomstufen (z) und die Fokusebenen (l).

zeigt, keinen Einfluss auf die Vergrößerung, was auch eine Clusterbildung in Abb. 2.5(b) sowie der Spearman-Rangkorrelationskoeffizient[13] ($r_s = -0{,}19$, $\alpha = 0{,}026$) statistisch bestätigt.

Da die in Kapitel 4.1 vorgestellte Datenübertragung von den durch die Aufbereitung erzeugten, kleineren Datenpaketen profitiert und die erzeugte Netzwerklast senkt, wird die Vergrößerung der Datensätze akzeptiert.

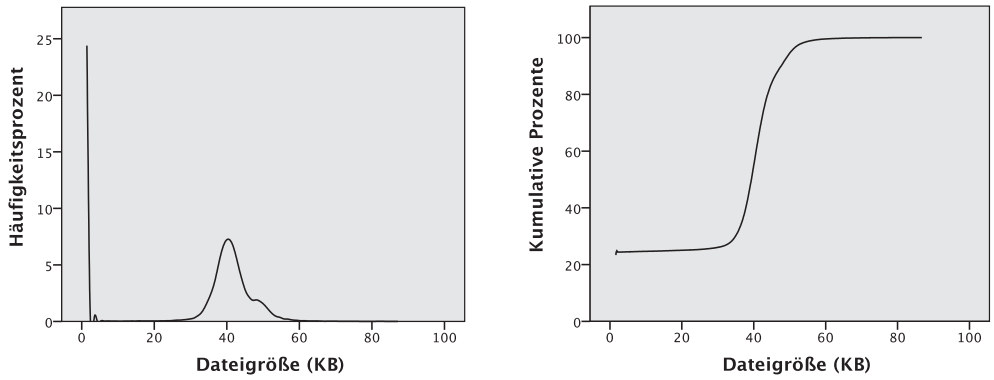
2.2.1 Verteilte Speicherung von 3D Objekten

Für dreidimensionale Objekte besteht durch die Abhängigkeit der Daten in drei Ebenen eine besondere Herausforderung für die Darstellung des kompletten Objektes ohne das bereits alle Daten geladen wurden, z.B. im Rahmen einer Vorschau. Ganzkörperscans aus Magnetresonanz- (MRT) oder Computertomographen (CT) bestehen, in Abhängigkeit von der gewählten Auflösung, aus vielen Gigabyte bis einigen Terabyte an Daten. Um auch diese Datensätze verfügbar zu machen, wurde ein Konzept zur verteilten Speicherung von 3D-Objekten entwickelt [126], das diese Herausforderungen adressiert.

Ziel der Aufbereitung von 3D-Datensätzen ist neben der effizienten verteilten Speicherung vor allem eine Geschwindigkeitsoptimierung beim Laden der Datensätze. Zu diesem Zweck können verschiedene Teile der Datensätze gleichzeitig aus mehreren Quellen bezogen werden (engl. *multi-source retrieval*). Weiterhin ist die Anzeige von lediglich teilweise geladenen Datensätzen von besonderem Interesse. So können vor allem große Datensätze in weniger performanten Umgebungen zumindest initial bereits dargestellt und betrachtet werden. Die Ordnerstruktur für die verteilte Speicherung ist durch die entwickelte Datenstruktur anpassungsfrei von den 2D-Daten aus Kapitel 2.2 zu übernehmen. Die Z-Schichten der 3D-Daten entsprechen dabei den Fokusebenen der zweidimensionalen Datensätzen.

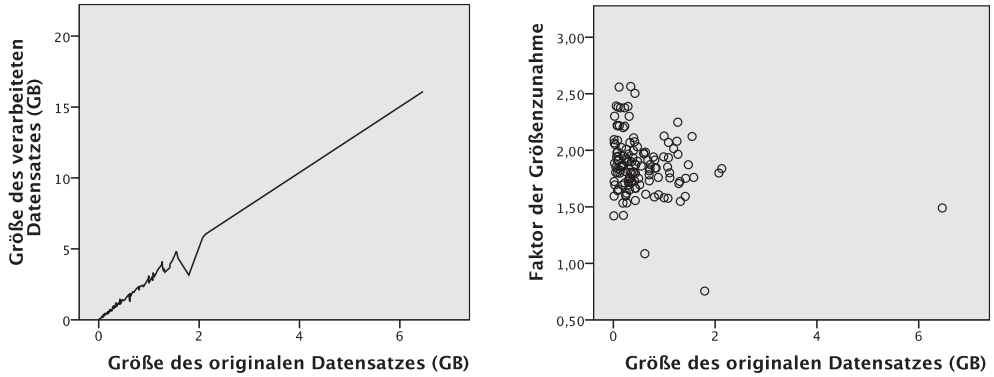
Zur Visualisierung der 3D-Daten wurde ein auf *Volume Rendering* basiertes Verfahren verwendet, dass im Besonderen für Aufnahmen aus MR oder CT geeignet ist [52]. Volume Rendering verwendet volumetrische Modelle für Darstellungen von komplexen Objekten, deren Oberfläche oft gar nicht oder nur sehr aufwändig durch polygonale Approximation dargestellt werden können, wie z.B. Nebel [61]. Um die Komplexität der Darstellung für nur teilweise geladene Datensätze zu reduzieren, wurde eine eigene Datenstruktur für den *Texture Cube*

[13] Spearman-Rangkorrelationskoeffizient: ein nicht-parametrisches Rangkorrelationsmaß zur Bestimmung des Zusammenhangs zweier Variablen über eine monotone Funktion, benannt nach Charles Spearman, (⋆10. September 1863, †17. September 1945), englischer Psychologe [56]



(a) Histogramm (b) kumulative Verteilung

Abbildung 2.4: Exemplarische Darstellung der Verteilung von Dateigrößen innerhalb eines Datensatzes.



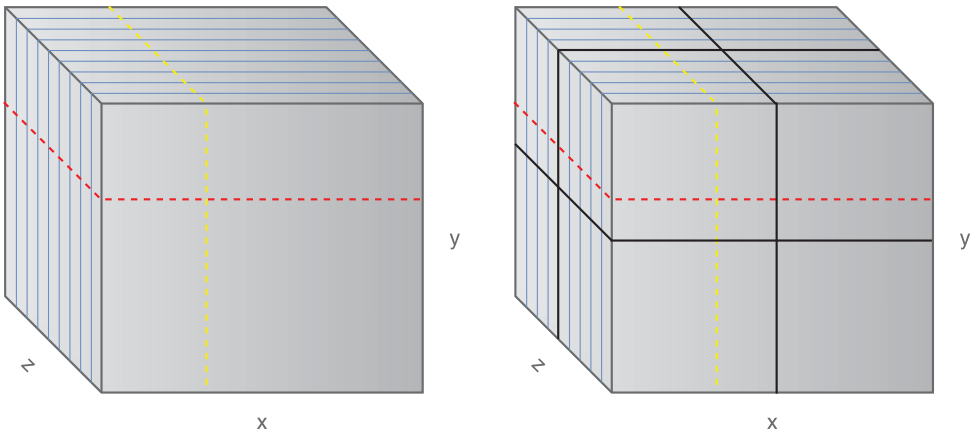
(a) Vergleich der Datensatzgrößen vor und nach der Aufbereitung

(b) Analyse der Größenänderung

Abbildung 2.5: Darstellung der Größenzunahme aller Datensätze durch die Vorverarbeitung in Abhängigkeit von der ursprünglichen Datensatzgröße.

eingeführt. Mit dem Texture Cube wird das Konzept der Bildzerlegung in die dritte Dimension der üblicherweise als *Z-Stapel* gespeicherten Aufnahmen der X-Y-Ebene transferiert. Er enthält alle Z-Schichten der jeweiligen Teilbereiche in der X-Y-Ebene.

Ohne weitere Unterteilung erfordern alle Ansichten außerhalb der X-Y-Ebene (vgl. Abb. 2.6(a)) die Übertragung von allen Daten in der Z-Achse (gelbe und rote Linie). Um dies zu verhindern, wird der Texture Cube in Subcubes unterteilt, die neben der X-Y-Ebene auch eine Untergliederung der Z-Achse ermöglichen



(a) Texture Cube mit Bildstapel in Z-Richtung, der alle Daten für Schnitte außerhalb der X-Y-Ebene benötigt.

(b) Texture Cube mit Unterteilung in Subcubes, der die Datenmenge in beliebigen Schnittebenen optimiert.

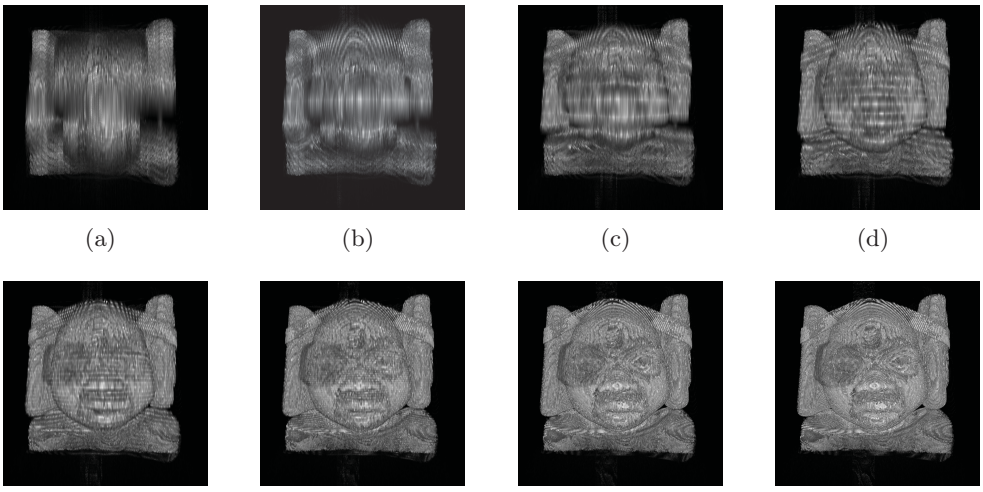
Abbildung 2.6: Darstellung von Texture Cubes zum Vergleich der Datenlast möglicher Schnittebenen, nicht-optimiert (a) und optimiert (b).

(vgl. Abb. 2.6(b)). Die Anzahl der Subcubes ermittelt sich dynamisch entsprechend der Datensatzgröße und das Konzept ihrer verteilten Speicherung ist dem pyramidalen Ansatz für 2D-Daten angepasst. Auf diese Weise können die bereits entwickelten Mechanismen zur Speicherung von 2D-Daten weiterhin verwendet werden und die Menge der zu übertragenden Daten bei Schnitten durch mehr als eine Z-Scheibe wird deutlich reduziert (2.6(a) und (b), rote Linie). Dieses Verfahren kann ohne weitere Anpassung auch für frei rotierbare Schnittebenen angewendet werden, die in der Realität den Großteil der Anwendungsfälle im klinischen Alltag ausmachen.

Eine weitere Möglichkeit Daten bei der Übertragung einzusparen, ist die Verringerung der benötigten Schnitte der Z-Achse (vgl. Abb. 2.7). Die Übertragung wird dazu in drei Phasen eingeteilt:

Phase 0: Eine Darstellung des Datensatzes ist noch nicht möglich, weil die Anzahl an dazu benötigten Daten (zwei Z-Scheiben) noch nicht übertragen wurde.

Phase 1: Die Darstellung ist möglich, aber mit deutlichen Qualitätseinbußen behaftet. Treffen im Hintergrund neue Daten aus weiteren Übertragungen ein, wird die Darstellungsqualität sukzessiv verbessert.



(a) (b) (c) (d)

(e) (f) (g) (h)

Abbildung 2.7: Veranschaulichung der unterschiedlichen Darstellungsqualität gerenderter 3D-Daten eines männlichen Kopfes [128] in den Phasen 1 (a-e) und 2 (f-h), die selbst in 3G Netzwerken bereits nach 1,33s eine Darstellung erlauben.

Phase 2: Es sind mindestens 80% des Datensatzes geladen und eine qualitativ gute Darstellung ist bereits möglich. Zusätzliche Übertragungen beeinflussen die Darstellungsqualität nur noch marginal.

Auf diese Weise lässt sich beim *Slice-based* Volume Rendering [156] die Qualität der Darstellung durch die Menge der mit Texturen gefüllten Ebenen einer *Proxy Geometrie* beeinflussen [109]. Dazu wird für jeden Bildpunkt ein Strahl durch die Proxy Geometrie berechnet und die einzelnen Farbwerte anhand bedarfsorientierter Transferfunktionen aggregiert, um möglichst realistische Darstellungen zu erreichen. Je mehr Ebenen bereits vorhanden sind, desto detaillierter wird die Darstellung, allerdings vergrößert sich der Rechenaufwand. Fehler durch eine approximierte Geometrie, wie sie beim *Surface-Rendering* z.B. durch *Marching Cubes* [112, 172] entstehen, werden verhindert. Gleichzeitig wird die Geschwindigkeit beim Rendern durch die Unterstützung moderner GPUs gesteigert.

Weiterhin ermöglicht die Untergliederung in Subcubes eine Übertragungsmaximierung bei Verwendung mehrerer Quellen für die Datenübertragung, da die jeweiligen Upload-Bandbreiten der einzelnen Quellen so besser ausgenutzt werden können. Unter der Voraussetzung, dass die Daten eines Subcubes für das

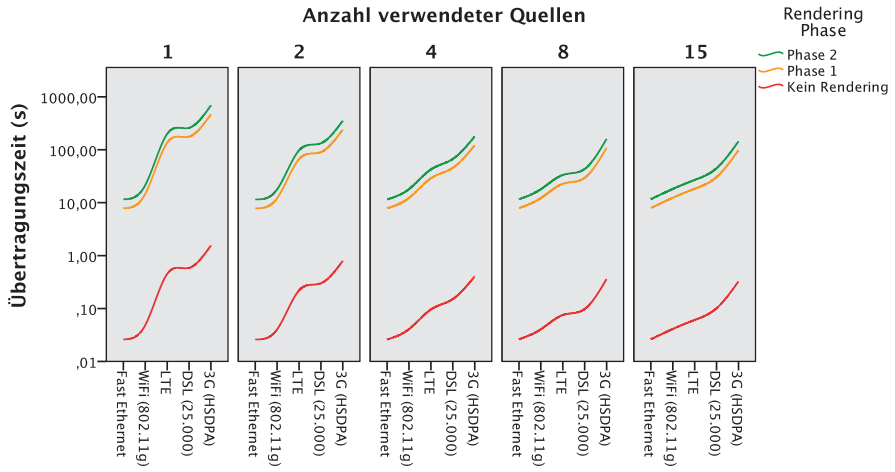


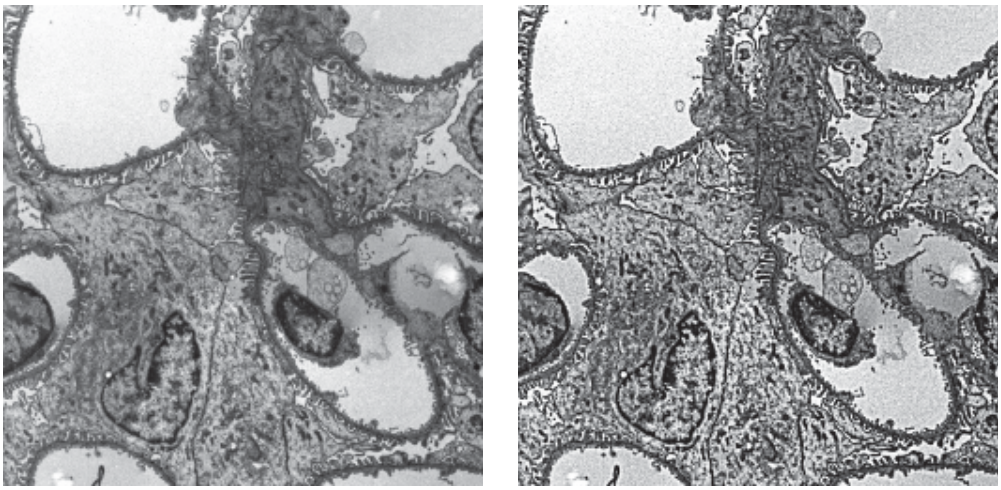
Abbildung 2.8: Vergleich des Einflusses mehrerer Quellen auf die Übertragungszeit und die Zeit bis zur initialen Darstellung eines Volumendatensatzes unter Berücksichtigung unterschiedlicher Netzwerkumgebungen.

Rendering eines Datensatzes lediglich teilweise vorliegen müssen, ergibt sich ein zusätzlicher Zeitgewinn, der, abhängig von der verwendeten Netzwerkumgebung, einen signifikanten Einfluss auf die Geschwindigkeit der initialen Darstellung hat (vgl. Abb. 2.8).

2.2.2 Qualitätsverbesserung der Bilddaten

Die Verwendung mehrerer Hersteller-Bibliotheken sowie der Bio-Formats-Bibliothek des OME erlaubt Omentum die Integration unterschiedlichster Bilddaten über die gleiche Vorverarbeitungskette und ihre Aufbereitung für eine verteilte Speicherung. Ein weiterer Vorteil der Datenaufbereitung ist es Einfluss auf die Bildqualität zu nehmen. Die von CCD Chips in Digitalkameras oder Scannern generierten Bilder enthalten gewöhnlich sehr starkes Rauschen [60]. Um das Signal-Rausch-Verhältnis (engl. SNR, signal-to-noise-ratio) in einem digitalen Bild zu verbessern, sind zahlreiche Algorithmen bekannt [31, 45, 104].

Für Omentum wurde ein gewichteter Medianfilter [30] verwendet, da über die Gewichtung ein guter Kantenerhalt bei gleichzeitiger Rauschunterdrückung erreicht werden kann. Dieser Filter ist als separates Modul in die Vorverarbeitungskette integriert und kann bei Bedarf sehr einfach gegen andere Algorithmen ausgetauscht werden. Es ist ratsam die Zeit für die Vorverarbeitung der Daten trotz des fehlenden Bedarfs an Echtzeitberechnungen möglichst klein zu halten.



(a) originales Bild f (b) gefiltertes Bild f'

Abbildung 2.9: Ein Ausschnitt einer größeren elektronenmikroskopischen Aufnahme[14], die anhand einiger Zellen einer renalen Filtrationseinheit die Ergebnisse der Rauschunterdrückung zeigt, die in der Vorverarbeitungskette eingesetzt wird.

Der in Gleichung 2.1 gezeigte *MEDIAN*-Operator o wird durch den Filter m aus Gleichung 2.2 auf ein Bild f angewandt, um das resultierende Bild $f' = f + m$ zu erhalten (vgl. Abb. 2.9). Der Median wird in einem Fenster der Breite $2w + 1$ und Höhe $2h + 1$ berechnet.

$$o(x,y) = \frac{\sum_{-w}^{w} \sum_{-h}^{h} f(x + w, y + h)}{w \cdot h} \tag{2.1}$$

$$m(x,y) = (w - 1) \cdot (f(x,y) - o(x,y)) \tag{2.2}$$

Um Stitching-Artefakte zu vermeiden, die an einer möglicherweise zu geringen Bildgröße nach der Anwendung des Filters liegen, werden im Vorfeld größere, passende Teilbilder aus dem aktuell bearbeiteten Datensatz extrahiert. Im Anschluss an die eigentliche Filter-Operation werden die Bilder passend beschnitten und die Aufbereitung entsprechend fortgesetzt.

[14] Die Aufnahme stammt aus dem Datenbestand der Arbeitsgruppe für Elektronenmikroskopie des Instituts für Anatomie I. © Dr. Klaus Zanger

2.2.3 Partitionierungsstrategie für 2D- und 3D-Daten

Wie bereits in Kapitel 2.2 gezeigt, sind die verwendeten Datensätze in der Regel mehrere Gigabyte groß und enthalten sehr viele kleine Dateien. Um nicht alle Dateien einzeln als Objekte zur Verteilung verwalten zu müssen, wird eine Partitionierung zur Untergliederung eines Datensatzes eingeführt. Als Partition wird in diesem Kontext eine Gruppe von Kacheln definiert, die einen lokalen Zusammenhang besitzen. Nicht notwendigerweise sind benachbarte Partitionen disjunkt zu einander. Die Partitionierung verringert nicht die Anzahl der Objekte, die zu einem Datensatz gehören, verkleinert jedoch den benötigten Adressraum, der von allen Objekten geteilt wird. Weiterhin soll durch die Partitionierung die Menge an Objekten, für die ein Knoten verantwortlich ist, im Vergleich zur Anzahl an Objekten im unpartitionierten Datensatz verkleinert werden, um eine bessere Lastverteilung zu ermöglichen.

Die Herausforderung an diesem Ansatz ist, dass alle am Netzwerk teilnehmenden Rechner (engl. Peers) in der Lage sein müssen zu berechnen, welche Partitionen zu einem Datensatz gehören und welche Kacheln in einer Partition enthalten sind. Nur so lässt sich die Basis für ein erfolgreiches Anfordern der Kacheln aus einem verteilten System legen, wie es in Kapitel 3 beschrieben wird.

Dazu lässt sich als T_0 die Anzahl an Kacheln in der maximal vergrößerten Schicht eines Datensatzes aus dessen Breite w und Höhe h sowie den Abmessungen einer Kachel (t_x,t_y) aus Gleichung 2.3 berechnen.

$$T_0 = \left\lceil \frac{w}{t_x} \right\rceil \cdot \left\lceil \frac{h}{t_y} \right\rceil \tag{2.3}$$

Die maximale Anzahl an Partitionen Π kann jetzt als Quotient aus T_0 und der maximalen Anzahl an Kacheln n_0 in der maximal vergrößerten Schicht berechnet werden (s. Gl. 2.4). Für eine einfachere Berechnung sollte n_0 eine Potenz von Zwei sein.

$$\Pi = \frac{T_0}{n_0} \tag{2.4}$$

Die Anzahl an Kacheln, die eine Partition P in der maximal vergrößerten Schicht enthält, ist durch ihre Dimension ($w_0 \cdot h_0$) beschränkt. Sie lässt sich anhand von Gleichung 2.5 ermitteln, wobei sich sowohl Breite als auch Höhe der Partition aus den Dimensionen des Datensatzes bestimmen lassen.

$$T_0(P) = w_0 \cdot h_0,\ w_0 = \left\lceil \frac{w}{t_x\sqrt{n_0}} \right\rceil \wedge h_0 = \left\lceil \frac{h}{t_y\sqrt{n_0}} \right\rceil \tag{2.5}$$

Die Gesamtanzahl der Kacheln einer Partition in all ihren Vergrößerungen und Fokusebenen wird anhand von Gleichung 2.6 berechnet, da die Anzahl an Vergrößerungsstufen z und Fokusebenen l aus der Vorverarbeitung (s. Kap. 2.2) bereits bekannt sind.

$$T(P) = \sum_{i=0}^{z} \left(\frac{l \cdot T_0(P)}{2^i} \right) \tag{2.6}$$

Ausgehend von dieser Information lässt sich die erste Kachel einer Partition P_k mit $k \, \epsilon \, \Pi$ aus Gleichung 2.7 bestimmen.

$$T_S(P_k) = ((k \cdot \sqrt{n_0}) \div w_0) \cdot h_0 + ((k \cdot \sqrt{n_0}) \bmod w_0) \tag{2.7}$$

Da somit jeder Peer in der Lage ist die Anzahl an Partitionen in einem Datensatz und deren jeweilige erste Kachel zu bestimmen, können die Partitionen zur Verteilung verwendet werden.

Zusätzlich werden durch die Rundung in Gleichung 2.5 die Kacheln der höher gelegenen Schichten, die unter anderem für Voransichten verwendet werden, häufiger repliziert, da sie in mehreren Partitionen enthalten sind und in der Regel auch sehr viel häufiger angefordert werden.

2.3 Dynamische Daten

Die in Omentum verfügbaren dynamischen Daten besitzen im Gegensatz zu den nicht von Aktualisierungen betroffenen, statischen Bilddaten nur ein sehr kleines Datenvolumen von wenigen Byte. Besonderes Augenmerk liegt hier auf der Aktualisierung und der Synchronisierung im Peer-to-Peer-Netzwerk. Im Folgenden werden die drei Gruppen dynamischer Daten grundlegend beschrieben und die spezifischen Anforderungen für die Aktualisierung behandelt.

2.3.1 Annotationen

Annotationen sind Markierungen, die Benutzer an beliebigen Positionen in einem Datensatz anfertigen können. Im vorliegenden Anwendungsfall können Sie als kurze Information, zur Vermittlung von komplexen Inhalten oder einer gezielten Fragestellung dienen. Zur Definition von Annotationen werden mehrere Parameter verwendet anhand derer auch die systemweit eindeutige ID jeder Annotation berechnet wird (vgl. Abb. 2.10). Besonders betrachtet wird hier die Form, die den Typ der Annotation und das den Umriss definierende Polygon enthält. Aus

Gründen der Effizienz für die Darstellung und Speicherung wird das Polygon nach der Erstellung einer Annotation vereinfacht. Aktuell werden vom Prototypen Kreise, Rechtecke und Freihandformen als Annotationen unterstützt. Über den einzugebenden Beschreibungstext können Annotation über die Suchfunktion wieder aufgefunden werden.

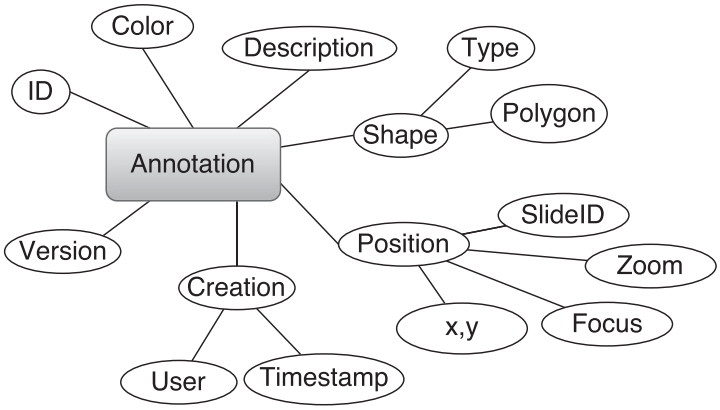


Abbildung 2.10: Schematische Darstellung der Parameter, über die eine Annotation eindeutig identifiziert werden kann.

Für die Eingabe der Beschreibungstexte wurden Schlagwortkataloge (Terminologia Anatomica[15]) und Freitexte betrachtet. Der Schlagwortkatalog bietet eine leichte Portierbarkeit in neue Sprachen, da intern lediglich mit Begriff-IDs gearbeitet wird, und beugt Schreibfehlern vor. Gleichzeitig ist die Eingrenzung auf einen vorgegebenen Katalog auch hinderlich, weil die Formulierung von Fragen in einem zumindest dem Benutzer teilweise unbekannten Themengebiet wesentlich schwieriger ist. Aus diesem Grund wurde zunächst die Freitext-Variante umgesetzt, die wiederum eine höhere Anforderung an die Suche stellt. Die Schlagwortkataloge sind aktuell lediglich als Unterstützung zur Kategorisierung vorgesehen.

Die *Versionsnummer* wird als Hilfsmittel zur Kollisionsauflösung verwendet. Auf diese Weise können sehr einfach doppelt eingehende Änderungen an der gleichen Version einer Annotation erkannt und an den modifizierenden Client zurückgegeben werden. Bei der Aktualisierung einer Annotation wird eine neue Kopie gespeichert und die vorherige Version wird zur Sicherung und Verlaufskontrolle beibehalten.

[15] Homepage der International Federation of Associations of Anatomists mit einem seit 2011 frei verfügbaren Auszug der Terminologia Anatomica: `http://www.unifr.ch/ifaa/`

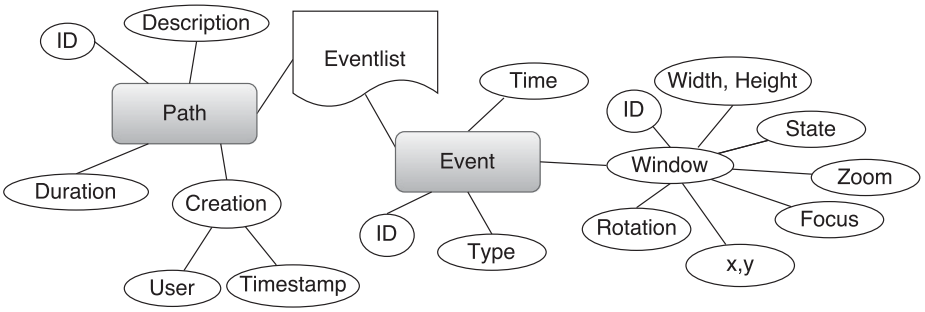


Abbildung 2.11: Schematische Darstellung der wichtigsten Parameter, über die ein aufgezeichnetes Interkationsmuster eindeutig identifiziert werden kann.

2.3.2 Aufgezeichnete Interaktionsmuster

Die Aufzeichnung von Benutzeraktionen ist sowohl für die Anwender als auch für die Analyse und Evaluation des Systems sehr hilfreich. Anwender können untereinander komplexe Erläuterungen zur Lokalisation bestimmter Strukturen und deren differentialdiagnostischer Betrachtung erstellen. Für die Evaluation des Systems lassen sich aufgezeichnete Interaktionen ideal als Eingabe verwenden, weil sie fehlerfrei und auf eine Millisekunde genau reproduziert werden können.

Jedes so erstellte Interaktionsmuster besteht aus einem Metadatensatz, der die abstrakten Beschreibungen enthält, und einer Liste der einzelnen Aktionen, die in der Summe das Interaktionsmuster erzeugen (vgl. Abb. 2.11). Analog zu den Annotationen sind auch hier die vergebenen IDs systemweit eindeutig, da die individuellen Parameter einer Aktion zur Berechnung der ID verwendet werden.

Das spezielle Format der Interaktionsmuster erlaubt eine effiziente Übertragung der Aktionen über das Netzwerk, da lediglich kleine Pakete anstatt eines Videos übertragen werden müssen. Auch ist eine nachträgliche Bearbeitung der Daten wesentlich effizienter möglich, da sehr leicht durch Veränderung der zeitlichen Offsets einzelner Aktionen die Reihenfolge verändert werden kann. Ebenso können neue Aktionen eingefügt oder bestehende entfernt werden. Auch hier wird analog zu den Annotationen eine Versionsnummer zur Kollisionsauflösung herangezogen, wobei es zu überprüfen gilt, wie häufig einmal erstellte Interaktionsmuster in der Praxis verändert werden.

Aufgezeichnete Aktionen lassen sich zusätzlich sehr gut in den Evaluationen verwenden, da ihre Ausführung objektiv reproduzierbar ist. Ihre Struktur bietet zusätzlich eine einfache Möglichkeit zur Diskretisierung um Ähnlichkeiten zwischen Aktionsmustern zu berechnen.

2.4 Zusammenfassung

In diesem Kapitel wurden die unterschiedlichen Datenformate beschrieben, die mit Omentum zur Verfügung gestellt werden sollen. Aufgrund der bis zu 50 GB großen Datensätze und der meist proprietären Formate der einzelnen Scannerhersteller, wurde ein Verfahren zur Datenaufbereitung entwickelt, dass die Bilddaten für 2D-Datensätze in eine JPEG-Pyramide überführt. Im Dateisystem werden die Kacheln der Pyramide eindeutig identifiziert, indem sie in Ordnern passend zur ID des jeweiligen Datensatzes, der Vergrößerungsstufe, der Fokusebene und ihren x- und y-Koordinaten im Koordinatenraum des Datensatzes gespeichert werden.

Zur Adaption auf die Anzeige von lediglich teilweise geladenen 3D-Objekten wurde das Konzept des Texture Cubes beim Slice-based Volume Rendering um Subcubes ergänzt, deren Z-Stapel sich auf die Verzeichnisstruktur der 2D-Daten abbilden lassen und dort die Fokusebene ersetzen. Es konnte gezeigt werden, dass dieses Verfahren eine effiziente Unterstützung für ein Multi-Source Retrieval bietet und durch sukzessive Qualitätsverbesserung der Darstellung bei weiteren eingehenden Daten bereits nach 1,33 Sekunden eine initiale Darstellung in 3G-Netzwerken erlaubt.

Da für die Konvertierung der Eingangsdaten ohnehin jedes Pixel betrachtet werden muss, wurde eine zusätzliche Qualitätsverbesserung zur Reduktion des starken Signal-Rausch-Verhältnisses der CCD-Chips in den Scannern integriert. Der eingesetzte, gewichtete Medianfilter verbessert die Darstellung deutlich, ohne die Sichtbarkeit filigraner Strukturen negativ zu beeinflussen.

Weiterhin wurde ein Verfahren vorgestellt, das es jedem Knoten im Netzwerk erlaubt die Partition zu berechnen, in der sich eine gesuchte Kachel befindet. Dies bildet die Grundlage für ein zu entwickelndes Routing-Verfahren, mit dem einzelne Daten aus dem Netzwerk angefragt werden können.

Die pyramidale Aufbereitung der Daten vergrößert die Datenmenge durchschnittlich um den Faktor 1,87. Gleichzeitig führt die Partitionierung zu übersichtlichen Kachelgruppen, die sich besser im nachfolgend vorgestellten Peer-to-Peer-Overlay speichern lassen, als komplette Datensätze mit einem Volumen von mehreren Gigabyte.

Die Aufbereitung und Partitionierung der vorgestellten Datensätze ermöglicht eine effiziente Ausnutzung der Datenübertragung aus mehreren Quellen zur spezifischen Darstellung relevanter Bildanteile bei gleichzeitiger Qualitätsoptimierung. Zusätzlich wird mit dieser Strategie das partielle Rendering von dreidimensiona-

len Daten unterstützt, deren Volumendaten anhand des neu entwickelten Konzeptes der Subcubes, analog zu zweidimensionalen Datensätzen, verteilt gespeichert werden können.

Speziell die hochveränderlichen, dynamischen Daten in Form von Annotationen und aufgezeichneten Interaktionen, die im System zu verteilen sind, werden in effizienten Datenstrukturen gespeichert. Eine Versionierung ermöglicht in Kombination mit systemweit eindeutigen IDs eine verlässliche Kollisionskontrolle.

KAPITEL 3

Omentum: ein auf Zufallsgraphen basierendes Overlay

Dieses Kapitel beschreibt die grundlegende Architektur des in dieser Arbeit neu entwickelten, unstrukturierten Omentum Overlays, das sich speziell auf den Anwendungskontext einer verteilten, interaktiven Lehr- und Lernplattform fokussiert. Das Overlay verknüpft die in Abschnitt 2.2.3 beschriebenen Partitionen miteinander in einem Zufallsgraphen und erlaubt so ein Routing zwischen den verschiedenen Partitionen, die im Zufallsgraphen *virtuelle Knoten* darstellen. Über die ID des Datensatzes, zu dem eine Partition gehört, kann die ID des zugehörigen virtuellen Knotens von jedem Peer errechnet werden. Die virtuellen Knoten werden in einem Bootstrap-Prozess an beitretende *Peers* übergeben, worüber die in Kapitel 4 beschriebene Replikation realisiert wird.

Um die Funktionsweise des entwickelten Overlays zu verdeutlichen, wird zunächst ein Überblick über den Funktionsumfang der Softwaremodule gegeben, die auf allen Geräten für eine Teilnahme am Overlay erforderlich sind. Jeder Peer im Netzwerk stellt prinzipiell die gleichen Funktionen zur Verfügung (vgl. Abb. 3.1). Dazu gehört eine graphische Oberfläche für angemeldete Benutzer. Weiterhin wird ein Modul verwendet, das wichtige Funktionen des Clients überwacht, z.B. die aktuelle Leistung, und ihn bei Bedarf vom Netzwerk trennt oder neu verbindet (vgl. Kap. 3.3 und 4). Unabhängig voneinander werden die gespeicherten Bilddaten in einem eigens entwickelten und für den Anwendungsfall optimierten Dateisystem (vgl. Kap. 5) gespeichert und die Anfragehäufigkeit der Daten für eine Optimierung der Lastverteilung analysiert. Ein eigener Nachrichtendienst (vgl. Kap. 6) sorgt für die Übertragung der benötigter Nachrichten zwischen Benutzern und auch Peers im Netzwerk. Die Nachrichtenübertragung ist nur gestattet sofern der neu entwickelte Authentifizierungsdienst die Korrektheit der benötigten

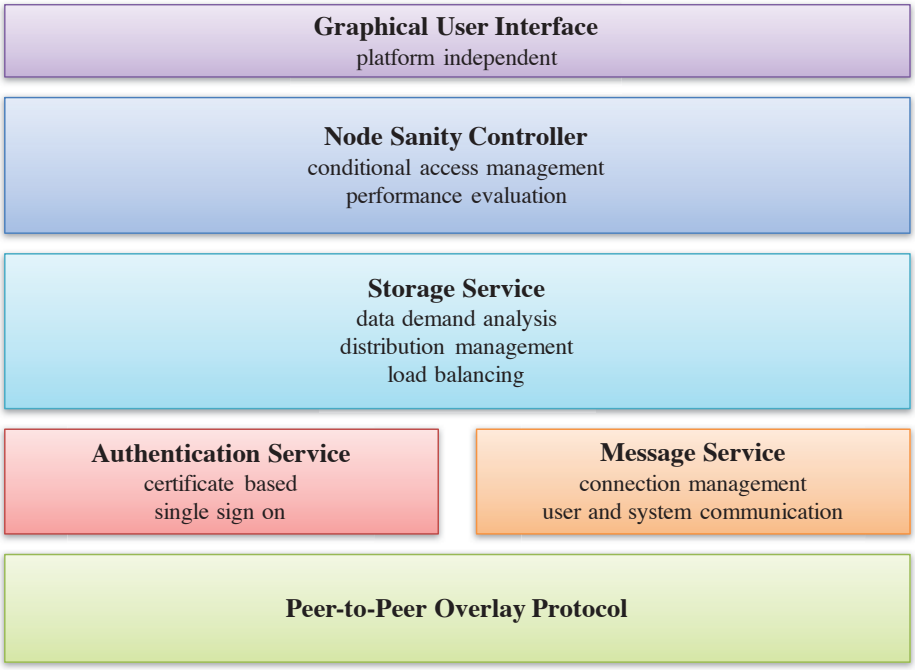


Abbildung 3.1: Schematische Darstellung der einzelnen Software-Module, die ein Peer im Omentum Overlay ausführt und ihre grundlegenden Funktionen.

Zertifikate bestätigt hat. Als elementare Funktionen stellt das Omentum Overlay sechs Operationen zur Verfügung:

get() lädt ein global eindeutig identifizierbares Objekt aus dem Netzwerk. Dabei kann es sich um erstellte Annotationen, eine Folge von Benutzeraktionen (Interaktionsmuster) oder eine darzustellende Kachel handeln. Diese Methode nutzt ggf. Routingmechanismen, falls der für das Objekt verantwortliche Peer nicht in unmittelbarer Nachbarschaft liegt und so keine direkte Verbindung zum Ziel existiert.

put() speichert das übergebene Objekt, z.B. eine Annotation, ein Interaktionsmuster oder einen neuen Datensatz im Netzwerk. Existiert dieses Objekt bereits im Netzwerk aktualisieren die verantwortlichen Peers autonom die verteilten Kopien. Vor allem aus Gründen des medizinischen Datenschutzes ist es nur bestimmten Entitäten im Netzwerk erlaubt neue Datensätze (Bilddaten) einzuspeisen, wohingegen alle Benutzer Annotationen oder Interaktionsmuster erstellen und speichern können.

search() bietet die Möglichkeit komplexer Suchen nach beliebigen Inhalten. So können Freitexte in den Annotationen oder Interaktionsmustern, die im Netzwerk gespeichert sind, gesucht werden. Es könnten z.B. alle Objekte zu dem Begriff „Glisson-Trias“[16] ermittelt werden. Diese Suche muss die Annotationen und Interaktionsmuster zu allen Datensätzen berücksichtigen sowie auch alle Metadaten zu den Bilddaten durchsuchen. Gleichzeitig werden Index-basierte Suchen nach gespeicherten Bilddaten effizient unterstützt.

join() erlaubt einem beliebigen Peer am Netzwerk teilzunehmen. Im Rahmen des in Kapitel 3.3 beschriebenen JOIN-Prozesses wird einem Peer mindestens eine Datenpartition zugeordnet. Eine Ausnahme bildet hier lediglich das in Kapitel 4.5 vorgestellte Router-Konzept zur Anbindung ressourcenarmer, mobiler Clients. Beim Beitritt wird vom Bootstrap-Knoten anhand einer Bewertung der bekannten virtuellen Knoten ein geeigneter Standort für den neuen Peer ermittelt.

leave() informiert die Nachbarschaft des Peers, der das System verlassen möchte, über sein Ausscheiden, schließt entsprechende Verbindungen und veranlasst ggf. erforderliche Aktualisierungen der Routing-Tabellen unmittelbarer Nachbarn.

recover() erlaubt es dem Netzwerk nach Ausfall von einem oder mehreren Peers wieder einen zunächst lokal und dann auch global konsistenten Zustand anzunehmen. Dazu werden ausgefallene Peers aus den entsprechenden Nachbarschaften entfernt und die jeweiligen Leistungskennzahlen für jeden Peer oder virtuellen Knoten neu berechnet.

Omentum wurde als verteiltes System entwickelt um Benutzern die Möglichkeit zu geben gemeinsam auf einem sehr großen Datenbestand zu arbeiten. Definitionen zu verteilten Systemen sind in der Literatur zahlreich zu finden. Diese sind jedoch mehrheitlich unzureichend, weshalb Andrew Tanenbaum und Maarten van Steen 2006 die Definition 3.1 einführten [161].

Definition 3.1 *Ein verteiltes System ist eine Ansammlung von unabhängig arbeitenden Computern, die für ihre Nutzer als ein kohärentes System erscheinen.*[17]

[16] Glisson Trias: anatomisches Eponym für die gemeinsam im periportalen Bindegewebe verlaufenden A. interlobularis, V. interlobularis und Ductus biliferi interlobularis

[17] Übersetzung des Autors, Original: „A distributed system is a collection of independent computers that appears to its users as a single coherent system.“ [161, S. 2]

Diese Definition vereint nach Meinung der Autoren zwei wesentliche Aspekte eines verteilten Systems: ein verteiltes System besteht aus (autonomen) Komponenten und wird von außen als zusammenhängendes Konstrukt wahrgenommen. Eine weitere Bedingung der Autoren dabei ist, dass die Komponenten miteinander interagieren müssen und der mögliche Ausfall einzelner Teile des Gesamtsystems – häufig von außen unbemerkt – kompensiert werden kann [161].

Da Protokolle von Peer-to-Peer-Systemen in der Regel auf heterogen miteinander vernetzten Rechnern arbeiten, werden sie als Overlays bezeichnet. Sie stellen ein logisches Netzwerk dar, z.B. auf einer physikalisch existierenden Infrastruktur oder einem anderen Overlay. Dies ermöglicht eine vielschichtige Struktur, wie sie z.B. für Quality of Service (QoS) [97] beim Streaming hoch aufgelöster Videodaten über das Internet zu finden ist [9].

Allen Overlays gemeinsam ist eine Routing-Strategie anhand von Metainformationen eines Knotens ohne vorherige Kenntnis seiner IP-Adresse. Die in der Literatur vorgestellten Overlaytopologien, können in strukturierte und unstrukturierte Overlays unterteilt werden [113].

In strukturierten Overlays, wie z.B. Chord [153], Tapestry [176] oder Pastry [141], wird allen Daten ein Schlüssel zugeordnet, der wiederum auf einen Knoten im Netzwerk abgebildet wird. Dieser Knoten speichert eine Kopie oder einen Verweis auf die Daten, denen der Schlüssel entspricht. Sollen Daten angefordert werden, ist der entsprechende Schlüssel zu berechnen und anschließend der zugeordnete Knoten anzufragen. Auf diese Weise wird eine effiziente Suche nach einem spezifischen Datensatz ermöglicht. Komplexe Suchanfragen sind in der Regel nicht vorgesehen oder nur durch zusätzliche Datenstrukturen umzusetzen, wie z.B. durch DAST [37] Plexus [4] oder SRing [155].

Bei unstrukturierten Overlays, wie z.B. BitTorrent [131], Gnutella [38] oder BubbleStorm [164], werden Knoten in einem Zufallsgraphen entweder flach oder hierarchisch (z.B. in Super-Peer Overlays wie Kazaa [69] oder eMule [53]) angeordnet. BitTorrent nimmt in dieser Gruppe eine Sonderstellung ein, da aktuelle Publikationen die Verwendung eines Trackers, der alle Knoten im Netzwerk kennt, als Verletzung des Prinzips unstrukturierter Overlays werten. Eine Suche wird vielfach z.B. durch *flooding* oder *random walks* realisiert. Dabei evaluiert in der Regel jeder Knoten eine erhaltene Anfrage in seinem lokalen Kontext, was auch komplexe Suchanfragen erlaubt. Ein evidenter Nachteil vieler Implementierungen ist, dass für die Lokalisation von selten replizierten Daten sehr viele Knoten möglicherweise ergebnislos angefragt werden müssen.

3.1 Omentum Overlay

Der Entwurf des Omentum Overlays basiert auf einer unstrukturierten Architektur, da dieses Konzept in sehr dynamischen Umgebungen deutlich robuster ist als strukturierte Overlays [161]. Weiterhin werden komplexe Suchen nach generierten Inhalten nativ unterstützt, was in strukturierten Systemen nur durch zusätzliche Erweiterungen realisierbar ist (vgl. Kap. 3.1.2). Das Anforderungsprofil für das Omentum Overlay umfasst in der Designphase fünf Attribute:

Skalierbarkeit Das Overlay muss zur Unterstützung von einigen Millionen Benutzern, die weltweit mehrere Zehntausend Datensätze gleichzeitig betrachten, offen skalieren und robust gegenüber Ausfällen und Fehlern sein.

Replikation Die Verfügbarkeit aller Datensätze muss durch effiziente Strategien sichergestellt sein und eine kostengünstig verifizierbare Anzahl von Replikaten erlauben.

Effiziente Suche Geeignete Strategien für komplexe Suchen sollen das Auffinden von benutzergenerierten Inhalten optimal unterstützen und dabei die Ressourcen des Netzwerks und der beteiligten Peers schonen.

Lastverteilung Die Verteilung der Anfragen soll die heterogene Leistungsfähigkeit aller virtuellen Knoten und Peers, vor allem in Hinblick auf die Ihnen zur Verfügung stehende Bandbreite, berücksichtigen.

Geringer Overhead Die Belastung des Peer-to-Peer-Netzwerks durch systemimmanente Nachrichten, z.B. Routing oder Replikation, soll möglichst gering gehalten werden.

Der Zufallsgraph, der unstrukturierten Peer-to-Peer-Architekturen in der Regel zugrunde liegt, ist im Fall von Omentum gerichtet und erweitert den in ähnlicher Form auch für das BubbleStorm Overlay [164] vorgestellten Ansatz und seine Erweiterung PathFinder [23]. Beide Overlays wurden im Hinblick auf Konstruktion und Replikation analysiert (vgl. Kap. 3.1.2). Sie bieten bereits eine solide Basis für kombinierte Index-basierte und komplexe Suchen. In Omentum wurde lediglich die Konstruktion des Zufallsgraphen aus den Overlays übernommen. Auf diese Weise konnten die spezifischen Anforderungen des Anwendungsfalls, im Hinblick auf Datenbereitstellung, Update-Frequenz für benutzergenerierte Inhalte und Analyse der Zugriffsmuster, adressiert werden.

Der Zufallsgraph wird anhand von zwei Pseudozufallszahlgeneratoren (engl. pseudo random number generator, kurz PRNG) gebildet, mit denen die Nachbarschaft für jeden virtuellen Knoten im Overlay berechnet werden kann. Mit dem ersten PRNG wird die Anzahl c der Nachbarn ermittelt, die ein Knoten v erhalten wird. Die maximale Größe einer Nachbarschaft ist von der aktuellen Größe des Netzwerks und damit der Anzahl virtueller Knoten abhängig sowie zusätzlich durch eine obere Schranke l limitiert. Diese Schranke verhindert ein unkontrolliertes Wachstum einer Nachbarschaft und bietet damit in sehr großen Netzwerken eine Unabhängigkeit von deren Größe N, denn der aus einer größeren Nachbarschaft resultierende erhöhte Verwaltungsaufwand für die einzelnen Knoten führt nicht zu signifikanten Pfadverkürzungen im Netzwerk.

Die Anzahl virtueller Knoten n im gesamten Netzwerk ist jedem Knoten bekannt, da sie als Summe der Partitionen (vgl. Kap. 2.2.3) aller gespeicherten Datensätze definiert ist. Mit dieser Information lässt sich anhand von Gleichung 3.1 eine geeignete Nachbarschaftsgröße c für den Knoten v ermitteln.

$$c(v) = \min_{l} \|x \cdot \log_b n\| \tag{3.1}$$

Signifikanten Einfluss auf das Ergebnis hat die Wahl der Basis b, da sie die Schrittlänge für eine Reorganisation des bestehenden Netzwerks festlegt. Ist sie zu groß, wird die Vorwärtsverkettung von Knoten mit kleiner ID zu Knoten mit großer ID zu schwach und damit die Belastung für die ersten Knoten im Rahmen des Routings sehr viel größer. Bei einer zu klein gewählten Basis wird das Netzwerk sehr häufig reorganisiert und damit sehr viele neue Kanten hinzugefügt, die eine zusätzliche Belastung durch eine größere Nachbarschaft für alle beteiligten Knoten bedeuten.

Bei sehr kleinen Netzwerken, die z.B. aus weniger als 50 Knoten bestehen, bietet der Faktor x eine gute Möglichkeit signifikanten Einfluss auf die Kantendichte im Netzwerk auszuüben, wobei er in großen Netzwerken durch die Schranke l keine zusätzliche Komplexität bedingt. Eine adäquate Nachbarschaftsgröße liegt dann vor, wenn bereits bei Netzwerken mit nur einigen Dutzend Knoten eine Partitionierung des Zufallsgraphen verhindert werden kann und bei großen Netzwerken, deren Knotenzahl in der Größenordnung $> 10^6$ liegt, der Verwaltungsaufwand für die Nachbarschaft nicht zu groß wird.

Der zweite PRNG ermittelt anschließend, auf Basis der zum aktuellen Zeitpunkt existierenden Knoten, die IDs, die zur Nachbarschaft hinzugefügt werden sollen.

Zur geeigneten Auswahl der Parameter aus Gleichung 3.1 wurden Simulationen mit unterschiedlich großen Netzwerken berechnet, die anhand der mittleren Pfadlänge zwischen allen Knoten Aufschluss über die Komplexität der erzeugten Graphen geben (vgl. Abb. 3.2). Die Reduktion der durchschnittlichen Nachbarschaftsgröße (vgl. Abb. 3.2(a)) beeinflusst bei großen Basen die durchschnittliche Pfadlänge im Zufallsgraphen sehr stark (vgl. Abb. 3.2(b)). Die durchschnittliche Pfadlänge bedingt die Anzahl notwendiger Routingschritte beim Nachrichtentransfer und erhöht so die Last auf den beteiligten Knoten. Die Auswahl einer geeigneten Basis sollte sowohl den Verwaltungsaufwand für zu große Nachbarschaften, als auch die Vermeidung von unnötigem Nachrichtenaufkommen im Netzwerk berücksichtigen.

Ein nach Gleichung 3.1 berechneter Zufallsgraph folgt einer Poisson-verteilten[18] Knotenanordnung (vgl. Abb. 3.3). Die Poisson-Verteilung ist eine diskrete Wahrscheinlichkeitsverteilung, die erwartete Ergebnisse einer Folge von Bernoulli-Tests[19] vorhersagt [28]. Für häufig wiederholte Tests mit geringer Erfolgswahrscheinlichkeit bietet die Poisson-Verteilung eine hinreichende Annäherung für die entsprechende Wahrscheinlichkeitsverteilung und wird häufig als die *Verteilung seltener Ereignisse* bezeichnet. Sie ermittelt die Wahrscheinlichkeit für eine Anzahl (k) unabhängiger Ereignisse einer zufälligen Folge innerhalb eines definierten Zeitabschnitts, wenn die durchschnittliche Anzahl an Ereignissen innerhalb einer bestimmten Zeit (λ) aus vorhergehenden Betrachtungen bekannt ist.

Die Wahrscheinlichkeitsverteilung P_λ wird durch den Parameter $\lambda \in \mathbb{R}_{>0}$ kontrolliert, der gleichzeitig Erwartungswert als auch Varianz ist. Wahrscheinlichkeiten werden der natürlichen Zahl k gemäß Gleichung 3.2 zugeordnet, wobei e der Eulerschen Zahl entspricht.

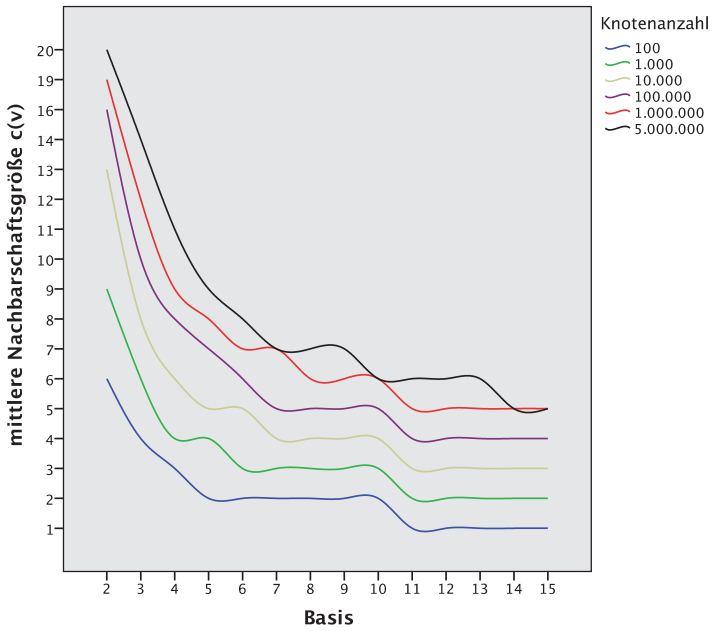
$$P_\lambda(k) = \frac{\lambda^k}{k!} e^{-\lambda} \tag{3.2}$$

Der Parameter λ kann als Ereignishäufigkeit in einem definierten Betrachtungszeitraum w mit konstanter Ereignisrate g betrachtet werden.

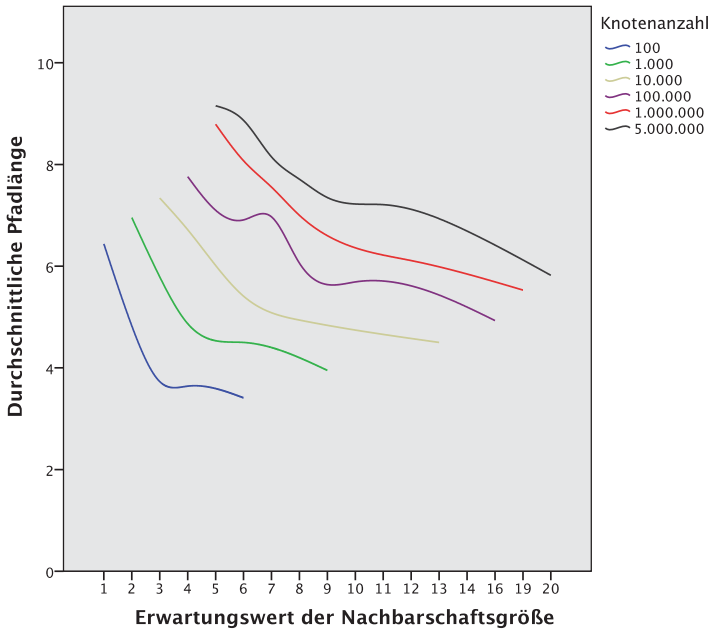
$$\lambda = g \cdot w \tag{3.3}$$

[18] benannt nach Siméon-Denis Poisson (⋆21. Juni 1781, †25. April 1840), französicher Mathematiker; erstmalig publiziert 1837 in „Recherches sur la probabilité des jugements en matières criminelles et en matière civile“ [59]

[19] benannt nach Jakob Bernoulli (⋆6. Januar 1655, †16. August 1705), schweizer Mathematiker [57]

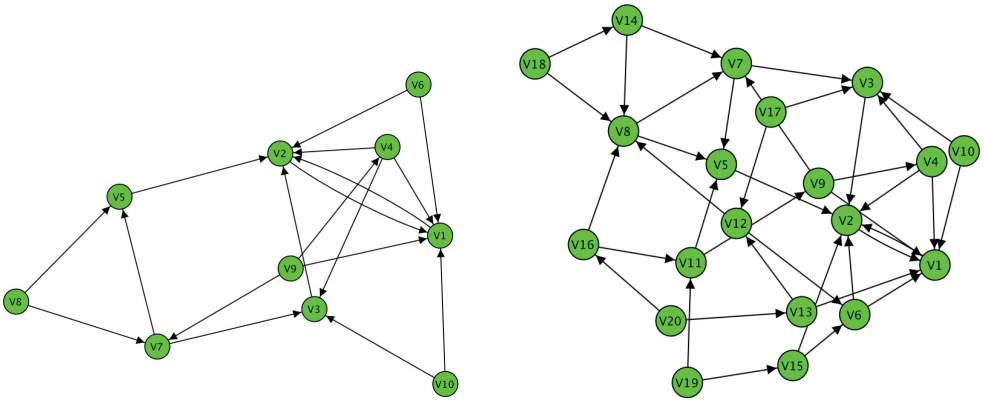


(a) Auswirkung der Basis-Manipulation auf die Nachbarschaftsgröße



(b) Einfluss der Nachbarschaftsgröße auf die Pfadlänge

Abbildung 3.2: Auswirkung einer Änderung der Basis b auf die Komplexität des erzeugten Zufallsgraphen.



(a) 10 Knoten (b) 20 Knoten

Abbildung 3.3: Darstellung des generierten Graphen mit unterschiedlicher Knotenanzahl.

Sind die Werte für λ und k sehr groß, wird durch die Berechnung von $P_\lambda(k)$ über die Stirling-Gleichung[20] [28] (s. Gl. 3.4) eine passable Annäherung $P'_\lambda(k)$ (s. Gl. 3.5) hergeleitet.

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n, \quad n \to \infty \tag{3.4}$$

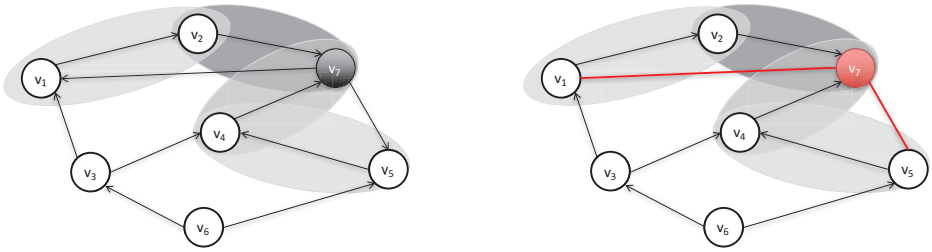
$$P'_\lambda(k) = \frac{e^{k(1+\ln(\lambda/k))-\lambda}}{\sqrt{2\pi(k+\frac{1}{6})}} \approx P_\lambda(k) \tag{3.5}$$

3.1.1 Lokale Routenberechnung und Pfadkompression

Durch die Möglichkeit, die Nachbarschaft jedes virtuellen Knotens im Netzwerk effizient lokal berechnen zu können, kann im Gegensatz zu strukturierten Overlays eine Routenermittlung zwischen zwei Knoten ohne Netzwerkverkehr erfolgen. Dazu sind die PRNG lediglich mit den IDs der Start- und Zielknoten zu initialisieren (vgl. Algorithmus 3.1).

Die Länge der generierten Route ist im Allgemeinen von der Größe des Netzwerks abhängig und entspricht, aufgrund der parallel von beiden Seiten ausgeführten Suche, dem kürzesten Pfad im gerichteten Graphen. Eine Routenberechnung

[20] benannt nach James Stirling (⋆Mai 1692, †5. Dezember 1770), schottischer Mathematiker [58]



(a) Pfadberechnung (b) Routing mit Pfadkompression

Abbildung 3.4: Vereinfachtes Beispiel zur Berechnung eines Pfades zwischen v_1 und v_5 sowie der möglichen Pfadkompression während der Routing-Phase im Overlay.

berücksichtigt lediglich ausgehende Kanten der betrachteten Knoten, da für eingehende Kanten zunächst der vollständige Graph berechnet werden müsste. Zur Laufzeit sind jedoch den virtuellen Knoten entlang der Route bereits alle eingehenden Kanten und auch alle anderen direkten Kommunikationspartner bekannt.

Während der Weiterleitung einer Nachricht entlang der berechneten Route sind alle erreichten Knoten in der Lage die berechnete Route auf Pfadverkürzungen zu untersuchen. Dabei wird die verbleibende Route in umgekehrter Reihenfolge nach bereits bekannten Knoten durchsucht und das Routing mit dem ersten Treffer fortgesetzt (vgl. Algorithmus 3.2).

Das in Abbildung 3.4(a) dargestellte Beispiel einer Routenberechnung ermittelt einen Pfad von v_1 zu v_5. Im zweiten Schritt von Algorithmus 3.1 wird der Knoten v_7 entlang der gerichteten Kanten als Schnittpunkt der beidseitig gestarteten Suchen identifiziert. Die berechnete Route enthält die Knoten v_2, v_7, v_4 und v_5 als Zielknoten.

Nach der Pfadberechnung wird die aktuelle Routingphase entsprechend Algorithmus 3.2 gestartet. Bei der Analyse der Wegpunkte in umgekehrter Reihenfolge erkennt v_1, dass der erste bekannte Knoten v_7 ist, von dem eine eingehende Kante bekannt ist. Daraufhin kann der Knoten v_2 übergangen werden und die Nachricht direkt an v_7 gesendet werden. Hier wird anhand des gleichen Verfahrens erkannt, dass es bereits eine Nachbarschaftsbeziehung zu v_5 gibt. Die ursprüngliche Route über vier Knoten kann so auf zwei Stationen reduziert werden (vgl. Abb. 3.4(b)).

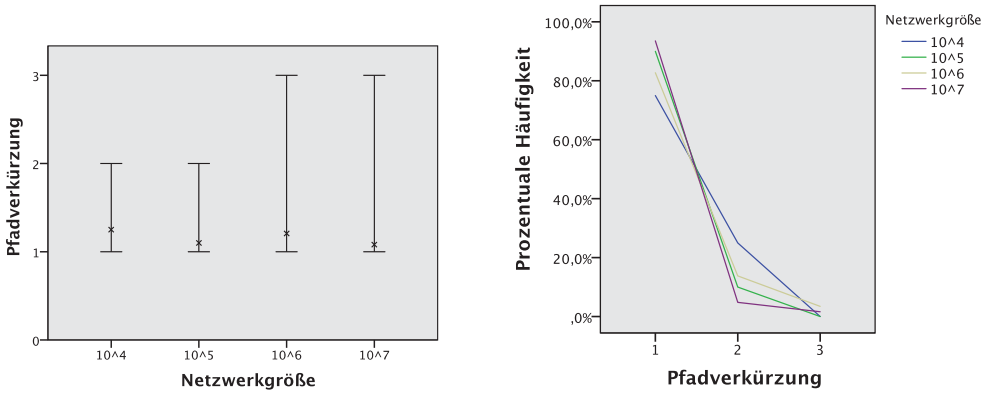
Die Möglichkeit der Pfadkompression reduziert die durchschnittliche Pfadlänge zwischen zwei kommunizierenden Knoten in Abhängigkeit von der gesamten Größe des Graphen. Anhand unterschiedlich großer, simulierter Netzwerke wird

Algorithmus 3.1 Rekursive Routenberechnung im Omentum Overlay

```
1:  s = sID;                                          ▷ ID des Startknotens
2:  d = dID;                                          ▷ ID des Zielknotens
3:  route = {};                                       ▷ Liste der Wegpunkte
4:  searchDirection = right;                          ▷ aktuelle Richtung der Breitensuche
5:  function CalculatePath(s,d,route,searchDirection)
6:      if searchDirection = right then
7:          searchDirection = left;
8:          route.add(s);
9:          if Neighbors(s) ∩ (Neighbors(d) ∪ {d}) = ∅ then
10:             for all n ∈ Neighbors(s) : ∄n ∈ way do
11:                 return CalculatePath(n,d,route,searchDirection);
12:             end for
13:         else
14:             return route;
15:         end if
16:     else
17:             ▷ Ausführung einer äquivalenten Suche in der anderen Richtung
18:     end if
19: end function
```

Algorithmus 3.2 Pfadkompression während der Nachrichtenübermittlung in Omentum

```
1:  route = {N_i, ..., N_n};                          ▷ Liste der Wegpunkte
2:  cNodes = {...};                                   ▷ Liste der bereits verbundenen Peers
3:  compressed = {};                                  ▷ komprimierte Route
4:  n = nil;                                          ▷ aktuell bearbeiteter Knoten
5:  function CompressPath(route)
6:      remaining = route.size();
7:      while remaining > 0 do
8:          n = route[remaining − 1];
9:          compressed.prepend(n);
10:         if n ∈ cNodes then
11:             return compressed;
12:         end if
13:         remaining = remaining − 1;
14:     end while
15: end function
```



(a) Einfluss der Pfadkompression

(b) Verteilung der Pfadkompression

Abbildung 3.5: Darstellung der Simulationsergebnisse zur Messung des Einflusses und der Verteilung der Pfadkompression in Netzwerken unterschiedlicher Größe.

der Einfluss der Pfadkompression näher untersucht. Dazu werden in den vollständig berechneten, gerichteten Graphen die Pfade von zufällig ausgewählten Startknoten (15% der im Overlay vorhandenen Knoten) zu ebenso zufällig ausgewählten Zielknoten berechnet und auf eine mögliche Pfadverkürzung überprüft.

Die Analyse (vgl. Abb. 3.5) zeigt, dass eine Pfadkompression in den getesteten Netzwerkgrößen mit sehr großer Wahrscheinlichkeit auftritt. Lediglich ihr Einfluss ist abhängig von der Netzwerkgröße und ermöglicht unterschiedlich starke Verkürzungen.

3.1.2 Verwandte Arbeiten

Die grundlegende Idee hinter der Verwendung von zwei PRNG zur Erzeugung des Zufallsgraphen stammt vom PathFinder Overlay [23], das die in BubbleStorm [163] verwendete, unstrukturierte Architektur um einen effizienten Index-basierten Lookup erweitert. Als Besonderheit setzt BubbleStorm voraus, dass die Daten eines Knotens ausschließlich während seiner Lebensdauer im Netzwerk existieren dürfen, was allerdings für den vorliegenden Anwendungsfall nicht zielführend ist.

Das Omentum Overlay erlaubt, im Gegensatz zu PathFinder, kein Löschen oder zusätzliches Speichern von neuen Datensätzen durch normale Benutzer. So entstehen nur selten neue virtuelle Knoten im Overlay, wohl aber treten regelmä-

ßig Peers dem Netzwerk bei oder verlassen es. Aus diesem Grund wird der in BubbleStorm und PathFinder verwendete Ansatz der *Random Walks* zur Integration neuer Datenobjekte verworfen und stattdessen der Ansatz der Nachbarschaftszuweisung anhand von zwei PRNG verfolgt. Für die grundlegenden Funktionen des Overlays wurden neue Algorithmen und ebenfalls ein neues Replikationsmodell entwickelt (vgl. Kap. 4), da die ursprünglichen Protokolle, Algorithmen und Replikationsstrategien von BubbleStorm und PathFinder aufgrund der modifizierten Datenintegration nicht mehr verwendet werden können.

Eine Vielzahl von Architekturen für Overlays verteilter Systeme sind bereits in der Literatur beschrieben worden. Vor allem strukturierte Overlays, wie z.B. CAN [134], Chord [153] oder Pastry [141], bieten effiziente Verfahren um Datensätze mit bereits bekannten IDs (oder Namen, die durch eine Hash-Funktion umgewandelt werden können) direkt zu lokalisieren. Abhängig von der umgesetzten Topologie des Overlays ist eine Kenntnis der genauen Adresse eines Knotens nicht erforderlich, da diese anhand der implementierten Routingstrategie, z.B. der binären Suche in Chord über Fingertabellen, ermittelt wird. Die überwiegend verwendeten, verteilten Hashtabellen (engl. *distributed hash table*, kurz DHT) bieten Skalierbarkeit, Lastverteilung und Robustheit bei Knotenausfällen, wie auch effiziente Lookup-, Insert- und Remove-Operationen für Daten (Keys). Dazu wird jedem DHT Knoten eine eindeutige ID zugeordnet, die in der Regel vom Hash-Wert des öffentlichen Schlüssel eines Knotens abhängt und z.B. bei Pastry 128 Bit lang ist [141]. Soll ein Datensatz eingefügt werden, so wird der Knoten, der die neuen Daten aufnehmen soll, anhand des geringsten Abstands seines Hash-Wertes zum Hash-Wert des neuen Datensatzes ermittelt. Komplexe Suchen, die mehrere Datensätze als Ergebnis liefern können, lassen sich hingegen nur mit Hilfe von Erweiterungen realisieren, z.B. zusätzlichen Indexstrukturen [136], weiteren Overlays [173] oder *Arbitrary Segment Trees* [37]. Unterschiede in den Implementierungen der vorgestellten DHTs gibt es vor allem im Hinblick auf die durchschnittliche Pfadlänge bei Lookups, die Fähigkeiten für Lastverteilung und die Widerstandsfähigkeit bei Ausfällen vieler Knoten (engl.: *churn*). Nachfolgend werden die durchschnittlichen Pfadlängen in unterschiedlichen DHT-Implementierungen analytisch verglichen.

Analog zu PathFinder beträgt die durchschnittliche Pfadlänge im Omentum Overlay

$$L_{avg}(N) = \frac{\log N}{\log c}, \tag{3.6}$$

wobei N die Anzahl der Knoten im Netzwerk und c seine Kapazität, die durchschnittliche Anzahl an Nachbarn eines Knotens, bezeichnet. Die Verwendung der vorgestellten, zweistufigen Pfadkompression während der Routing-Phase erlaubt eine zusätzliche Reduktion der Pfadlänge um durchschnittlich einen Hop.

Die Länge des Lookup-Pfades in Chord ist asymptotisch als

$$L_{avg_C}(N) = \frac{1}{(1+d)\cdot\log(1+d) - d\cdot\log d}\cdot\log N \tag{3.7}$$

definiert [177], wobei d die Fingerdichte in Chord bezeichnet. Wird die in Chord übliche Fingerdichte von 1 verwendet, beträgt $L_{avg_C} = \frac{\log N}{2}$. Bereits für kleine Knotenkapazitäten erlaubt der in Omentum verwendete Zufallsgraph deutlich kürzere Pfade als Chord.

In einem d-dimensionalen CAN, partitioniert in n gleiche Zonen, beträgt die durchschnittliche Pfadlänge bei festem d

$$L_{avg_A}(n) = \frac{1}{4}dn^{\frac{1}{d}}, \tag{3.8}$$

wobei jeder Knoten $2d$ Nachbarn unterhält [134]. Für große d folgt die durchschnittliche Pfadlänge einer Gauss-Verteilung[21] [24] und verhält sich damit ähnlich zu Chord.

Unter Pastry kann die Pfadlänge unter Einfluss des konfigurierbaren Parameters b, der nach Empfehlung von *Rowstron und Druschel* mit $b = 4$ verwendet werden sollte, als

$$L_{avg_P}(N) = \lceil \log_{2^b} N \rceil \tag{3.9}$$

abgeschätzt werden [141]. Die Routing-Tabelle in Pastry enthält $\lceil \log_{2^b} N \rceil$ Zeilen mit je $2^b - 1$ Einträgen. Für ein Netzwerk mit 50 Millionen Knoten, ergeben sich damit 96 Nachbarn für jeden Knoten. In Omentum lassen sich mit dem verwendeten Zufallsgraphen vergleichbare Ergebnisse bereits mit einer durchschnittlichen Kapazität von $c = 20$ erreichen.

Der Vergleich der durchschnittlichen Pfadlängen zeigt, dass die Routing-Komplexität im erzeugten Graphen, verglichen mit strukturierten Ansätzen, durch kürzere Pfade geringeren Overhead erzeugt und effizienter umgesetzt werden kann. Für strukturierte Overlays gibt es Erweiterungen, die der Ausführung von komplexen Suchen dienen, z.B. Mercury [17] oder Arbitrary Segment Trees [37].

[21] benannt nach Johann Carl Friedrich Gauss (⋆30. April 1777, †23. Februar 1855), deutscher Mathematiker [55]

Bei *Mercury*, einem skalierbaren Routing-Protokoll, werden ausgewählte Knoten entsprechend eines zugeordneten Attributs zu Routing-Verteilern. Die Unterstützung mehrerer Attribute in einer Anfrage führt zu einer Evaluation der Anfrage in unterschiedlichen Routing-Stationen, abhängig von den verwendeten Attributen. So wird sichergestellt, dass eine Suche, die ein bestimmtes Attribut betrifft, immer alle vorhandenen Daten findet. Für komplexe Suchen werden die Routing-Stationen in einem kreisförmigen Overlay angeordnet und die Daten angrenzend auf dem entstandenen Ring gespeichert. Diese Anordnung verhindert die Verwendung von Hash-Funktionen zur Speicherung der Daten, da so die kontinuierliche Anordnung der Daten nicht gewährleistet werden kann. Um die inhomogene Verteilung der Daten zu vermeiden ist eine explizite Umsetzung einer Lastverteilung unumgänglich.

Unstrukturierte Overlays, die auf Zufallsgraphen basieren, sind ebenfalls sehr stabil [72] und unterstützen auch komplexe Suchen, z.B. anhand von *Flooding*, wobei diese Suchstrategien häufig keine optimale Laufzeit haben und sehr viel Netzwerkverkehr erfordern. Der in BubbleStorm vorgeschlagene *BubbleCast*-Algorithmus [163] reduziert die komplexe Suche in Graphen auf ein Rendezvous-Problem[22] [162] und löst dieses durch die Replikation von Daten und Suchanfragen, um eine garantierte Erfolgswahrscheinlichkeit für einen Treffer zu gewährleisten. Für Omentum wurde die Anzahl der Knoten, die für eine komplexe Suche angefragt werden müssen, durch eine berechenbare Positionierung der benutzergenerierten Inhalte verringert, was die Komplexität des Rendezvous-Problems senkt (vgl. Kap. 4). Auf diese Weise können komplexen Suchanfragen nach Annotationen, Diskussionsbeiträgen und Fragen effizienter beantwortet werden.

Grundsätzlich werden in unstrukturierten Overlays unterschiedliche Suchverfahren differenziert. Für *Flooding* und *Random Walk* lassen sich nach Meinung von *Lin und Wang* mit der Anfrageneffizienz (engl. Query Efficiency, kurz QE) und dem Antwortverhalten der Suchen (engl. Search Responsiveness, kurz SR) zwei objektivierbare Parameter ableiten, die zum Vergleich unterschiedlicher Verfahren verwendet werden können [110]. Die Analyse der Autoren zeigt, dass Suchalgorithmen in den unstrukturierten Topologien entweder die gesamte Bandbreite im Netzwerk überlasten um eine möglichst optimale SR zu produzieren (Flooding) oder SR zugunsten einer besseren QE zu reduzieren (Random Walk).

[22] Rendevouz-Problem: 1976 erstmalig beschrieben durch Steve Alpern, 1995 von ihm formalisiert [7].

3.2 Dynamische Knotenbewertung

Um die aufkommenden Anfragen von Benutzern im Peer-to-Peer-Netzwerk adäquat verteilen zu können, ist eine objektive Bewertung der einzelnen Knoten unabdingbar. Die Leistungsparameter, die zu einer quantitativen Messung auf den Knoten herangezogen werden, sind im Einzelnen die Prozessor- und Speicherauslastung sowie die einem Knoten zur Verfügung stehende Netzwerk-Bandbreite.

Bandbreite Die Bandbreite zwischen zwei Knoten muss anhand von Nutzdaten bestimmt werden, weshalb das Einschleusen zusätzlicher Messpakete in den Datenstrom nicht möglich ist. Ausschlaggebend dafür ist die ohnehin hohe Belastung des Netzwerks, so dass vorhandene Nachrichten ausreichen müssen, um eine verlässliche Aussage mit tolerabler Varianz zu formulieren. Ist eine Nachricht aufgrund ihrer Größe für die Bandbreitenbestimmung geeignet, erfolgt die Messung automatisch.

RAM Die unter der verwendeten Java Version[23] möglichen Informationen zur Auslastung des Arbeitsspeichers werden herangezogen, um die aktuelle Situation des Speichers im Hinblick auf Page Faults, Speicherdruck und Swapping zu untersuchen.

CPU Die Belastung der CPU kann lediglich als Durchschnittswert ermittelt werden. Dieser ist jedoch höchstens ein Indikator für eine intensive Nutzung des Prozessors und muss zusätzlich auf die dem System zur Verfügung stehenden Prozessoren und/oder Kerne umgerechnet werden.

3.2.1 Grundlagen der Bandbreitenmessung

Speziell der Bandbreite kommt eine besondere Bedeutung zu, da sie einerseits quantitativ ohne zusätzliche Belastung des Netzwerks als auch kontinuierlich zur Erkennung von gravierenden Änderungen ermittelt werden muss. Zu diesem Zweck werden verschiedene Verfahren der Bandbreitenmessungen analysiert und auf ihre Anwendbarkeit hin untersucht [169].

Die üblicherweise mit einem Netzwerkpfad P assoziierten Metriken zur Bestimmung des Durchsatzes sind die End-zu-End Kapazität C und die verfügbare Bandbreite A. Als gemeinsame Einheit der Kapazität und Bandbreite wird *bit/s* verwendet. Die Kapazität beschreibt das maximale Datenvolumen, das über einen

[23] Oracle Java Runtime Environment (JRE) v. 1.8.0_60

Pfad P, bestehend aus h Stationen, versendet werden kann, wenn keine anderen Daten über P übertragen werden [86] und ist definiert als

$$C \equiv \min_{i=1...h} C_i. \tag{3.10}$$

Sofern über einen Link i in einem Zeitintervall $[t_0,t_{0+\tau}]$ bereits Daten übertragen werden, sinkt die Kapazität C_i um die Auslastung $C_i u_i(t_0,t_{0+\tau})$. Durch $u_i(t_0,t_{0+\tau})$, oder kurz $u_i^\tau(t_0)$, wird die Benutzung im Zeitintervall $[t_0,t_{0+\tau}]$ beschrieben, mit $0 \leq u_i^\tau \leq 1$. Die verfügbare Bandbreite $A_i^\tau(t_0)$ von Link i entspricht damit dem Anteil der Kapazität C_i, der im Zeitintervall noch nicht verbraucht wurde.

$$A_i^\tau(t_0) \equiv C_i\,[1 - u_i^\tau(t_0)] \tag{3.11}$$

Die verfügbare Bandbreite A^τ eines Pfades P in einem Zeitintervall lässt sich demnach über die minimale Restkapazität der verwendeten Links berechnen.

$$A^\tau(t_0) \equiv \min_{i=0...h} \{C_i\,[1 - u_i^\tau(t_0)]\} \tag{3.12}$$

Es ist zu beachten, dass es sich um Momentaufnahmen handelt, die einer starken Fluktuation durch diverse Parameter unterworfen sein können. So kann eine Applikation ausschließlich eine reduzierte Sendeleistung zur Verfügung stellen oder die Überlastkontrolle (engl.: Congestion Control, kurz CC) von TCP [81] eine vollständige Nutzung der Bandbreite verhindern. TCP steigert die Senderate zu Beginn einer Übertragung nur langsam, um sich der verfügbaren Bandbreite anzunähern. Diese Annäherung erfolgt über eine exponentielle Steigerung der Senderate bis zum ersten Paketverlust. Danach erhöht sich die Senderate linear [6]. Ist die Anzahl an Paketverlusten zu hoch, kann die Senderate der tatsächlichen Bandbreite nicht schnell genug angenähert werden und die Anwendung somit die zur Verfügung stehende Bandbreite nicht vollständig ausnutzen.

Zur Messung der verfügbaren Bandbreite stehen grundsätzlich zwei unterschiedliche Verfahren zur Verfügung. Beim *active probing* werden zusätzliche Netzwerkpakete, die ausschließlich der Messung dienen, in den Datenverkehr eingeschleust. Im Gegensatz dazu versucht das *passive probing* lediglich Datenpakete zur Messung zu verwenden, die ohnehin übertragen werden sollen. Beide Verfahren haben Vor- und Nachteile; diese müssen für jeden Anwendungsfall individuell beachtet werden.

Durch die separaten Pakete beim active probing lässt sich z.B. unter Verwendung von UDP [130] die Senderate und der Abstand der einzelnen Messpakete aktiv festlegen oder auch die Messintervalle zur Erholung der Paketwarteschlangen vergrößern. Nachteilig ist hier die Erzeugung von zusätzlichem Datenverkehr, der, vor allem in Systemen mit vielen untereinander verbundenen Knoten, eine signifikante Zunahme des globalen Netzwerkverkehrs bedeuten kann.

Die ausschließliche Verwendung von systemimmanenten Datenpaketen beim passive probing schränkt die Messung durch das gewählte Übertragungsprotokoll und die spezifischen Berechtigungen der Anwendung ein, z.B. werden Abstand und Senderate der Messpakete unter TCP dynamisch adaptiert. Häufig steht nur ein limitierter Zugriff auf die Informationen der Transportschicht des Systems zur Verfügung, was wiederum die mögliche Auswahl an Messverfahren einschränkt, da z.B. akkurate Zeitstempel für die Sende- und Empfangszeit eines Pakets auf der Anwendungsschicht nicht zur Verfügung stehen.

Eine Erzeugung von zusätzlichem Messverkehr ist bei der bereits im System herrschenden Datenlast keine akzeptable Lösung, weshalb passive probing als einzig sinnvolles Messverfahren verwendet wird.

In der Literatur sind seit 1980 verschiedene Messtechniken für unterschiedliche Metriken der Bandbreite zu finden. Die meisten Verfahren verwenden dabei entweder *packet pairs* bzw. *packet trains* [82] oder *self-induced congestion* [88]. Beide Techniken werden nachfolgend in Kürze erläutert, um die Grundlagen und Hindernisse für die Entwicklung eines geeigneten Verfahrens unter Verwendung von TCP zu verdeutlichen.

Packet Pair (PP)

Der TCP slow-start Algorithmus wurde 1988 durch *Jacobson und Karels* entwickelt [82]. Grundlegender Gedanke ist, dass aufeinander folgende Pakete auf einem Pfad durch Links mit geringerer Kapazität gestreckt werden. Da neue Pakete erst nach Erhalt eines ACK-Pakets vom Empfänger versendet werden, kann diese Streckung über den Abstand der Pakete bei ihrer Ankunft am Zielpunkt ermittelt werden.

Keshav stellte 1991 einen Ansatz für *Packet Pair Probing* vor, der es erlaubte einen Link anhand des Abstands der eintreffenden Pakete beim Empfänger zu klassifizieren [92]. 1993 wurde von *Bolot* die Roundtrip-Zeit von UDP-Paketen zur validen Bestimmung des Internetverkehrs eingesetzt und damit ebenfalls bestehende Verbindungen charakterisiert [20]. Der Versand von zwei Paketen über

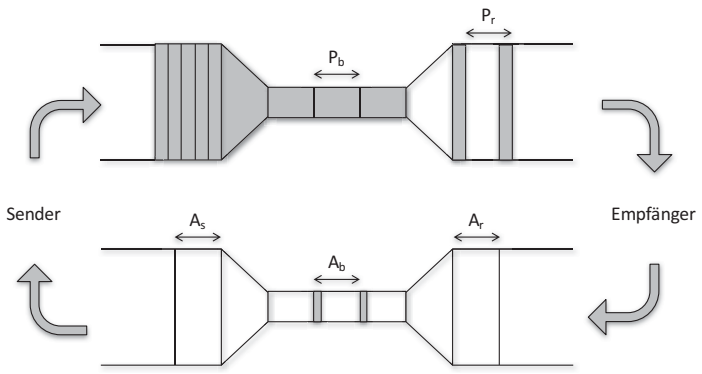


Abbildung 3.6: Schematische Darstellung der zeitlichen Streckung von TCP-Paketen durch einen *bottle neck* Link nach Jacobson [82]. P bezeichnet die Paketabstände auf dem bottle neck Link b und beim Empfänger r. Mit A sind analog die Abstände der Antwortpakete bezeichnet.

einen Pfad ermöglicht anhand ihres Abstands die Bestimmung der Kapazität des schwächsten Links (engl. bottle neck), da kein anderer Link die einzelnen Pakete noch weiter strecken wird (vgl. Abb. 3.6) [82]. Sofern der Sender anschließend neue Pakete ausschließlich als Antwort auf eine Sendebestätigung verschickt, entspricht der Sendeabstand exakt der Paketzeit im bottle neck.

Eine besondere Betrachtung erfordert *Cross Traffic*, also Datenverkehr, der nichts mit der eigentlichen Datenübertragung zu tun hat. Dessen Pakete können sich in den Paketwarteschlangen zwischen die Messpakete drängen [100], wodurch der Abstand der Pakete beim Empfänger ebenfalls beeinflusst werden kann. Eine Lösungsmöglichkeit liegt in der Kenntnis der Kapazität des bottle neck, um aus ihr die Menge an Cross Traffic auf dem entsprechenden Link zu interpolieren.

Eine zusätzliche Verfälschung kann sich durch die Abarbeitung der Paketwarteschlangen auf der Empfängerseite ergeben, falls die Messpakete in unterschiedliche Paketwarteschlangen eingereiht werden, was den Abstand der Pakete auch ohne Cross Traffic erhöht. Die Verkleinerung des initialen Paketabstands reduziert nach Meinung von *Hu und Steenkiste* die Wahrscheinlichkeit für die Verarbeitung von zwei Paketen in unterschiedlichen Warteschlangen [79].

Selbst in kleinen Zeitintervallen schwankt der Datenverkehr im Internet ungleichmäßig, weshalb in der praktischen Anwendung häufig *Packet Trains*, also Messreihen, die aus mehr als zwei Paketen bestehen, zur Anwendung kommen. So lassen sich die protokoll- oder verbindungsspezifischen Schwankungen in einzelnen Messwerten mathematisch ausgleichen [91, 106].

Self-Induced Congestion (SIC)

Jain [88] nutzt für SIC die Pfadeigenschaft eines zunächst steigenden Durchsatzes bei steigender Sendefrequenz bis die maximale Kapazität erreicht ist, woraufhin der Durchsatz aufgrund gefüllter Paketwarteschlangen abfällt, was u.a. von TCP [6] und TCP Vegas [25] z.B. für die Überlastkontrolle eingesetzt wird.

Zur Überlasterkennung werden die zeitlichen Abstände Δt_i der eintreffenden Pakete i und $i-1$ in der Paketwarteschlange gemessen und anhand der verfügbaren Bandbreite A des Pfades sowie der Senderate R_i für das Paket i verglichen. Sofern $R_i > A$ gilt, ist $\Delta t_i \geq \Delta t_{i-1}$ und bei $R_i < A$ gilt entsprechend $\Delta t_i \leq \Delta t_{i-1}$. Eine wiederholte Messung nähert mit diesem Verfahren die Senderate schließlich der verfügbaren Bandbreite an, wobei auch hier Cross Traffic, z.B. durch kontinuierliche Änderungen an Δt_i, einen gravierenden Einfluss auf die Messung haben kann [137]. Zur effektiven Minimierung eines möglichen Einflusses von intermittierendem Cross Traffic sollten mit diesem Verfahren die Messzyklen mehrerer Packet Trains betrachtet werden.

3.2.2 Verwandte Arbeiten

Die meisten Verfahren zur Messung der Bandbreite, die auf der Packet Pair Technik (z.B. Initial Gap Increasing [79], ABwE [122], Spruce [154]) oder self-induced congestion (z.B. Packet Transmission Rate [79], Pathload [87], pathChirp [137]) basieren, leiten die verfügbare Bandbreite aus dem im Netzwerk gemessenen Cross Traffic ab. Dazu muss jedoch die Kapazität des bottle neck bekannt sein, die mittels active probing über Verfahren wie z.B. bProbe [35], pathrate [50] oder Nettimer [101] ermittelt werden kann. Für passive probing ist dies jedoch nicht umsetzbar.

Auf SIC aufbauende Verfahren sind in der Regel besser für passive probing einsetzbar, denn die meist erforderliche Regulierung der Senderate lässt sich zumindest in Teilen über die von TCP selbstständig geregelte Überlastkonrolle erreichen. Das für diese Arbeit entwickelte Verfahren soll lediglich den Paketabstand auf Empfängerseite betrachten und die üblicherweise verwendeten Sendezeitstempel der Pakete verwerfen, um eine zusätzliche Systembelastung zu vermeiden. Zur Verdeutlichung der Unterschiede und zum besseren Verständnis werden nachfolgend die beiden Verfahren detaillierter beschrieben, die einen maßgeblichen Einfluss auf das Design hatten.

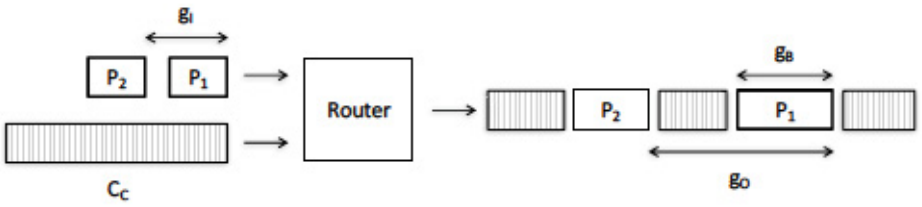


Abbildung 3.7: Schematische Darstellung der zeitlichen Streckung von TCP-Paketen durch Cross Traffic auf einem *bottle neck* nach *Hu und Steenkiste* [79]. g_I entspricht dem initialen Abstand, g_B ist die Paketlänge des ersten Messpaketes auf dem bottle neck und g_O ist der Paketabstand mit dem Cross Traffic C_C.

Initial Gap Increasing (IGI)

IGI [79] verwendet die Packet Pair Technik und adressiert das Problem, dass das zweite Paket eine leere Paketwarteschlange erreicht.

Hu und Steenkiste definieren dazu zunächst die Paketabstände beim Eintritt in den Router als g_I und beim Verlassen des Routers unter Einwirkung von Cross Traffic C_C als g_O. Die Paketlänge auf dem bottle neck Link ist g_B (vgl. Abb. 3.7). In dieser Situation gilt es für P_2 zwei unterschiedliche Szenarien zu betrachten.

P_2 erreicht eine leere Warteschlange, in der P_1 bereits verarbeitet wurde, was als *Disjoint-Queuing-Region* (DQR) bezeichnet wird. Betrachtet man die Bedingungen, die dazu vom Router erfüllt werden müssen, kann man erkennen, wann P_2 in der DQR ist. Die Warteschlage Q muss in Q/C_B abgearbeitet werden. P_1 muss in g_B und der angefallene Cross Traffic zwischen den Messpaketen in $C_C \cdot g_I/C_B$ verarbeitet werden, wofür maximal die Zeitspanne g_I zur Verfügung steht.

$$g_I > \frac{Q}{C_B} \cdot \frac{C_C \cdot g_I}{C_O} + g_B \tag{3.13}$$

Es gilt zu beachten, dass g_I möglichst klein sein sollte ohne dabei das Netzwerk zu stark zu belasten. Für einen Link mit der Kapazität C_i und eine Paketwarteschlange Q gilt

$$g_O = g_I - \frac{Q}{C_i}. \tag{3.14}$$

Der Quotient gibt die Wartezeit des ersten Pakets auf die Bearbeitung der Paketwarteschlange wieder.

Alternativ erreicht P_2 eine nicht-leere Warteschlange in der P_1 noch nicht verarbeitet wurde, was als *Joint-Queuing-Region* (JQR) bezeichnet wird. Die Wartezeit des ersten Pakets erhöht sich in diesem Fall um die Verarbeitung des

möglicherweise zwischen den Paketen liegenden Cross Traffic C_C.

$$g_O = g_B - \frac{C_C \cdot g_I}{C_i} \tag{3.15}$$

Zur Berechnung der Menge an Cross Traffic dürfen nach Meinung der Autoren nur die Pakete aus der JQR verwendet werden, da man mit Paketen aus der DQR die Ermittlung verfälschen würde.

Weiterhin geht IGI von der Annahme aus, dass konstanter Cross Traffic C_C auf einem Single Hop vorliegt, was jedoch die Realität nicht hinreichend abbildet. *Hu und Steenkiste* adressieren diese Unstimmigkeit durch die Verwendung von gemittelten Messwerten über *Packet Trains*, bei denen zwischen K unveränderten, M vergrößerten und N verringerten Paketabständen unterschieden wird. Intuitiv sind vor allem die vergrößerten Abstände von Interesse, da die anderen beiden Gruppen nur wenig bis keinen Cross Traffic erkannt haben.

Durch Anwendung von Gleichung 3.15 auf den Kontext der vergrößerten Abstände lässt sich der Cross Traffic abschätzen.

$$C_C(t) = \frac{C_O \cdot \sum_{i=1}^{M} \left(g_i^+ - g_B\right)}{\sum_{i=1}^{M} g_i^+ + \sum_{i=1}^{K} g_i^= + \sum_{i=1}^{N} g_i^-} \tag{3.16}$$

Experimentelle Analysen der Autoren zeigen, dass eine Annäherung an ein optimales g_I dann erreicht ist, wenn das durchschnittliche g_I dem durchschnittlichen g_O entspricht. Der dazu entwickelte Algorithmus beginnt bei $g_I = g_B/2$ und nähert sich in jeder Iteration um $g_B/8$ bis die Annäherung erreicht ist. Anschließend kann nach Subtraktion des Cross Traffic aus Gleichung 3.16 von der Kapazität des bottle neck C_B die verfügbare Bandbreite ermittelt werden.

Packet Transmission Rate (PTR)

PTR [79] basiert, wie auch IGI, auf dem Konzept der JQR und DQR, verwendet aber SIC als Messtechnik. Es wird daher versucht, sich über die Modifkation der Senderate der verfügbaren Bandbreite anzunähern. Die Nutzung des Algorithmus von IGI führt zu dem bereits bekannten Problem des fehlenden Einflusses auf die Senderate. *Hu und Steenkiste* verwenden daher die Empfangsrate der Pakete und nicht den Cross Traffic zur Messung. Ausgehend vom SIC-Prinzip entspricht die

Empfangsrate der verfügbaren Bandbreite A. Daher gilt

$$A = \frac{(M + K + N) \cdot L}{\sum_{i=1}^{M} g_i^{+} + \sum_{i=1}^{K} g_i^{=} + \sum_{i=1}^{N} g_i^{-}}, \tag{3.17}$$

wobei L der Länge der Messpakete entspricht und M, K sowie N die bereits bekannten Messwerte mit erhöhten, gleichbleibenden und verringerten Paketabständen bezeichnen.

Die Verwendung von mehreren Packet Pairs ermöglicht auf der Empfängerseite eine für Cross Traffic sensitive Messung der Paketabstände ohne genaue Kenntnis über die Kapazität des bottle neck.

3.2.3 Bandbreitenbewertung durch PTR mit TCP-CC

Ursprünglich wurden bei PTR, das auf der Idee von IGI basiert und SIC verwendet, Datenpakete per UDP übertragen, um die Senderate der Bandbreite anzunähern. Da UDP im Anwendungskontext nicht zur Verfügung steht, wird zur Anpassung der Senderate die Fluss- oder Überlastkontrolle von TCP ausgenutzt [169]. Dabei wird die Senderate zunächst exponentiell gesteigert (TCP-Slowstart). Nach einem ersten Paketverlust wird die Senderate lediglich linear weiter gesteigert (TCP-Congestion-Control) und die verfügbare Bandbreite fair unter allen TCP-Verbindungen aufgeteilt [6]. Nach Gleichung 3.17 ist die verfügbare Bandbreite genau dann erreicht, wenn der durchschnittliche Abstand der Messpakete auf der Sender- und der Empfängerseite gleich ist. Eine Messung wäre auch auf der Senderseite möglich, allerdings würde dort die Reaktion auf eine Messung erst mit Erhalt der Antwortpakete für die nächste Übertragung erfolgen können. Daher werden auf der Empfängerseite die Übertragungszeiten der einzelnen Pakete und ihr Abstand zueinander gemessen. Wäre nur die Kapazität der Verbindung von Interesse, würde die Messung der Übertragungszeit ausreichen. Für die Bestimmung des Durchsatzes ist jedoch vor allem der Cross Traffic von Bedeutung.

Die Verwendung von Nutzdaten für die Messung bedingt eine detaillierte Analyse der im Netzwerk verwendeten Paketgrößen, die mittels *Wireshark* [41] durchgeführt wurde. Die Tatsache, dass die Auslastung eines Netzwerkpfades hauptsächlich von großen Paketen erreicht wird [122], wurde anhand der erhobenen Daten bestätigt. So überwiegen auf den betrachteten Gigabit-Ethernet-Links hauptsächlich Pakete in der Größenordnung von 1.500 Bytes, die der konfigurier-

ten Maximum Transmission Unit (MTU) entspricht. Von der MTU müssen der IPv4-Header (20 Byte exklusive *Options*) [80], der TCP-Header (20 Byte exklusive *Options*) [81] und der auf UNIX-basierten Systemen übliche Zeitstempel (12 Byte) abgezogen werden, da nur die Nutzdaten eines Pakets betrachtet werden sollen, die somit 1.448 Byte je Paket ausmachen. Als Messpakete werden daher lediglich Nutzdaten ausreichender Größe (> 2.896 Bytes) verwendet.

Da nativ unter *Java* kein direkter Zugriff auf die Transportschicht erfolgen kann, müssen alle Messungen auf der Anwendungsschicht durchgeführt werden.

Die Verfälschung der Messung wird minimiert, indem nicht alle Pakete einer Verbindung zur Abschätzung der Bandbreite verwendet werden. So wird das erste Paket aufgrund des Zeitverlustes für den Verbindungsaufbau (SYN, SYN-ACK, ACK) ignoriert. Die Übertragung des letzten Pakets, das möglicherweise wesentlich kleiner als seine Vorgänger ist, findet ebenso keinen Eingang in die Berechnung.

Paketverluste beeinflussen die gemessene Bandbreite aufgrund der von TCP garantierten Übertragungsreihenfolge signifikant. Unter der Annahme, dass ein übertragenes Paket P_i während der Übertragung verloren geht, werden die möglicherweise bereits übertragenen Pakete $P_{i+1}, \ldots, P_{i+n}$ bis zum Eintreffen der neuen Übertragung von P_i auf der Transportschicht zwischengespeichert. Die Übertragungszeit für P_i könnte zwar exakt bestimmt werden, allerdings wäre der Abstand Δt_i zum Vorgänger P_{i-1} durch die doppelte Übertragung wesentlich größer und würde daher in der Berechnung einer zu niedrigen Bandbreite resultieren. Andererseits sind die nachfolgenden Pakete bereits gepuffert, so dass ihre Auslieferung an die Anwendungsschicht einen sehr viel kürzeren Abstand und damit eine zu hohe Bandbreite ergäbe. Daher wird eine Filterung durchgeführt, die alle Pakete von der Messung ausschließt, die einen konfigurierbaren Bereich um den Median der Bandbreite aller übertragenen Pakete oder eine maximale Bandbreite überschreiten.

Ähnliche Effekte werden auch durch einen Kontextwechsel einer Applikation verursacht, der z.B. während einer laufenden Messung die CPU entzogen wird, da die Messung nicht kontinuierlich erfolgt, sondern lediglich bei Verarbeitung eines eingetroffenen Pakets.

Die Berücksichtigung aller übertragenen Pakete zur Berechnung der verfügbaren Bandbreite kann zu einer geringeren Flexibilität bei der Erkennung von akuten Einbrüchen der Bandbreite führen. Aus diesem Grund werden ältere Messungen mit sinkender Gewichtung δ mit $0 \leq \delta \leq 1$ aggregiert. Für ein übertra-

genes Paket P_1 und ein Folgepaket P_2 lässt sich anhand von Gleichung 3.17 die Bandbreite A bestimmen als

$$A_1 = \frac{P_1 \cdot L}{\Delta t_1} \quad \text{und} \quad A_2 = \frac{(\delta \cdot P_1 + P_2) \cdot L}{\delta \cdot \Delta t_1 + \Delta t_2}. \tag{3.18}$$

Äquivalent ergibt sich für das Paket P_n eine Bandbreite mit gewichtetem Einfluss der Vorgängerpakete als

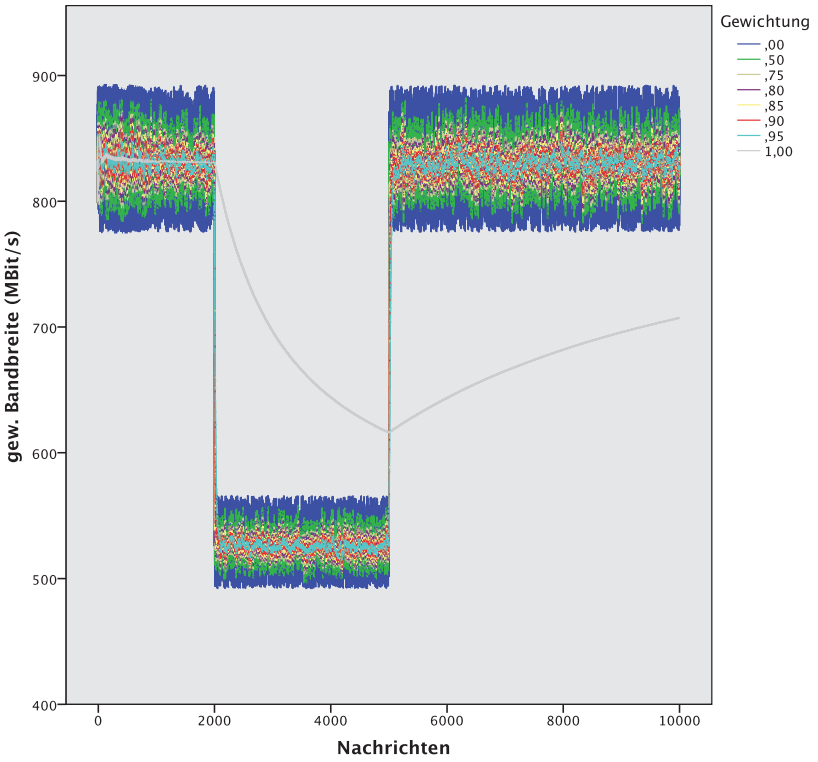
$$A_n = \frac{\left(\sum_{i=1}^{n} \left(\delta^{n-i} \cdot P_i \right) \right) \cdot L}{\sum_{i=1}^{n} \left(\delta^{n-i} \cdot \Delta t_i \right)}. \tag{3.19}$$

3.2.4 Evaluation

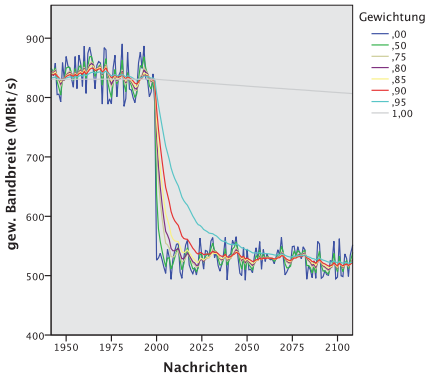
Für unterschiedliche Gewichte δ wurde die Reaktionsgeschwindigkeit des Systems auf eine Änderung der Bandbreite hin untersucht. Dazu wurde die Bewertung der Bandbreite auf einer dedizierten Verbindung unter Einwirkung von Cross Traffic gemessen. Die Gewichtung ist vor allem bei Verbindungen mit niedrigem Transfervolumen relevant, da es hier besonders wichtig ist, sich möglichst schnell der realen Bandbreite anzunähern. Bei sehr vielen übertragenen Nachrichten innerhalb einer kurzen Zeitspanne relativiert sich die Bedeutung der Gewichtung. Durch die leistungsgerecht verteilte Anforderung der Daten im Peer-to-Peer-Netzwerk ist die Gewichtung für die meisten Verbindungen jedoch von großer Bedeutung.

Getestet wurde in einem Gigabit-Netzwerk, da aufgrund der Übertragungsgeschwindigkeit im Vergleich zur Verarbeitungszeit der Nachrichten, die Schwankungen in den Messungen mit durchschnittlich rund 13% maximal waren. Der Cross Traffic im Volumen von 320 MBit/s wurde durch separate Clients auf der gleichen Verbindung mittels iPerf [103] eingespeist, um die ursprüngliche Messung der Bandbreite nicht zu verfälschen. Es wurden acht Messreihen zu jeweils 10.000 Nachrichten erhoben, wobei der Cross Traffic zwischen Nachricht 2.000 und 5.000 aktiv war (vgl. Abb. 3.8). Die mit iPerf im Vorfeld ermittelte Bandbreite der Verbindung lag bei 982 MBit/s. Durch die Einspeisung von Cross Traffic wurde die zur Verfügung stehende Bandbreite auf 557 MBit/s gesenkt.

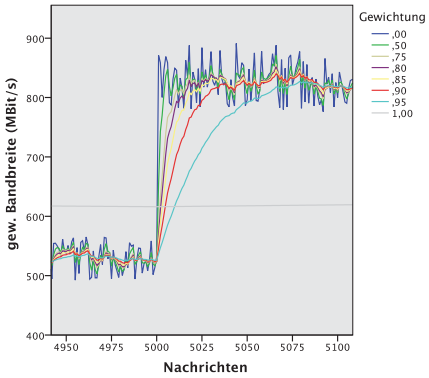
Je niedriger die Gewichtung ist, desto sensibler reagiert das System auf die Messtoleranzen von bis zu 13% im Gigabit-Netzwerk. Im Vergleich dazu liegt die Messtoleranz in langsameren Umgebungen sehr viel niedriger, z.B. knapp unter 3% im 3G Netz. Eine maximale Gewichtung ($w = 1$) bedeutet, dass alle vor-



(a) Übersicht der Messung



(b) Start des Cross Traffic bei Nachricht 2.000.



(c) Ende des Cross Traffic bei Nachricht 5.000.

Abbildung 3.8: Übersicht der Ergebnisse (a) sowie detaillierte Betrachtung des Flankenwechsels zu Beginn (b) und Ende (c) einer Bandbreitenmessung anhand von 10.000 Nachrichten unter Einfluss von Cross Traffic zum Vergleich unterschiedlicher Gewichtungen δ für die Optimierung der Reaktionsgeschwindigkeit bei möglichst hoher Stabilität im Messungsverlauf.

hergehenden Messungen den gleichen Einfluss auf die Bewertung der Bandbreite haben, wie die aktuelle Messung. Vor allem im Bereich der Bandbreitenwechsel durch den eingebrachten Cross Traffic ist eine schnelle Reaktionsfähigkeit erforderlich (vgl. Abb. 3.8(b) und 3.8(c)). Die ungewichtete Messung ($w = 0$) fluktuiert erwartungsgemäß und ist für die qualitative Betrachtung der Verbindungsgeschwindigkeit damit nicht geeignet. Eine Gewichtung von 0,8 wurde als reaktive und gleichzeitig stabile Lösung aus den Messungen extrahiert. Gravierende Bandbreitenänderungen werden so bereits nach 14 Messungen korrekt dargestellt und Messtoleranzen führen lediglich zu Schwankungen in der Größenordnung von durchschnittlich weniger als 5%.

3.2.5 Unscharfe Aggregation

Die Bandbreite oder die Auslastung von CPU und Arbeitsspeicher bieten für sich genommen keine aussagekräftigen Anhaltspunkte zur Leistungsbewertung eines Gesamtsystems. Daher wurde eine Aggregations- und Diskretisierungsstrategie für alle Parameter anhand einer konfigurierbaren Gewichtung mittels *Fuzzy Logic* entwickelt.

Anhand individueller Trapez-Funktionen [174] wird jedem Parameter ein Zugehörigkeitswert zu einer Leistungskategorie zugeordnet, wobei die jeweils möglichen Maximalwerte dynamisch angepasst werden. Zur finalen Aggregation wird die elipsoide Fuzzy-Funktion aus Gleichung 3.20 verwendet, in der die Bandbreite $a \in [0,A]$, die CPU-Last $l \in [0,L]$ und die Speicherauslastung $m \in [0,M]$ mit spezifischen Gewichten α, λ und μ versehen werden.

$$\frac{a^2}{(\alpha \cdot A)^2} + \frac{l^2}{(\lambda \cdot L)^2} + \frac{m^2}{(\mu \cdot M)^2} = 1 \tag{3.20}$$

Anhand des gewichteten euklidischen Abstands d eines Punktes $P(a,l,m)$ zum Mittelpunkt des Ellipsoiden

$$d = \sqrt{(\alpha \cdot a)^2 + (\lambda \cdot l)^2 + (\mu \cdot m)^2} \tag{3.21}$$

wird ihm über eine Trapez-Funktion ein diskreter Wert zugewiesen, wobei zentrumsnahe Punkte eine bessere Bewertung bedeuten. Daher wird über Gleichung 3.22 einem Punkt Q, der außerhalb des Ellipsoiden liegt, der gleiche Leistungsindex $I \in [0,I_{\max}]$ zugeordnet, wie dem Punkt Q', der in der Schnittgeraden auf dem Rand des Ellipsoiden liegt (vgl. Abb. 3.9).

$$I = \begin{cases} I_{\max}, & \text{falls} \quad \dfrac{a^2}{(\alpha \cdot A)^2} + \dfrac{l^2}{(\lambda \cdot L)^2} + \dfrac{m^2}{(\mu \cdot M)^2} \geq 1 \\ \dfrac{d}{I_{\max}}, & \text{sonst} \end{cases} \tag{3.22}$$

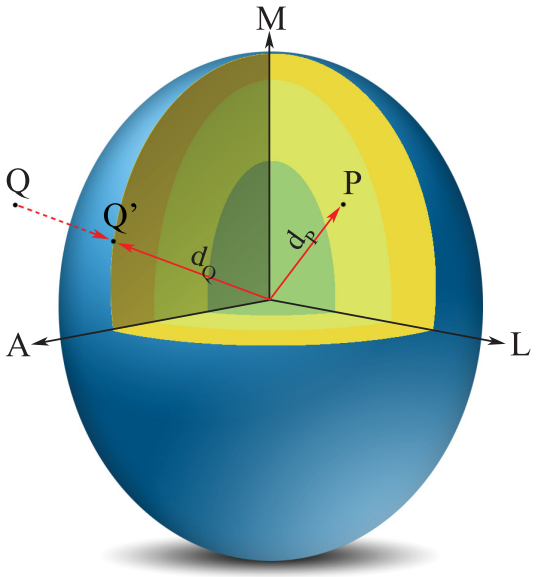


Abbildung 3.9: Schematische Darstellung der unscharfen Aggregationsfunktion zur Diskretisierung der drei Leistungsparameter auf einen Gesamtwert.

Unter Verwendung der in Kapitel 4.1 beschriebenen Schnittstelle wird die berechnete Bandbreite zwischen zwei kommunizierenden Knoten X und Y übermittelt (vgl. Abb. 3.10). Für beide Knoten ist es wichtig sowohl die Geschwindigkeit ihres Uplinks als auch ihres Downlinks zu kennen, denn nur aus beiden Werten lassen sich in asynchronen Netzwerken valide Aussagen zur Verbindungsgeschwindigkeit treffen.

Zunächst sendet X seine Anfrage an Y, auf dessen Seite damit die Bandbreite für den Upload $A_u(X)$ von X bestimmt werden kann. Diese Information wird X im Rahmen der nächsten Nachricht mitgeteilt, woraufhin X über $A_u(Y)$ den Upload von Y berechnen kann. Zusätzlich kann X durch die Kenntnis der eigenen Upload-Bandbreite seinen Leistungsindex I_X berechnen, den er Y in der nächsten Nachricht gemeinsam mit $A_u(Y)$ mitteilt. Mit dieser Information kann Y schließlich seinen Leistungsindex I_Y berechnen und ihn gemeinsam mit einer

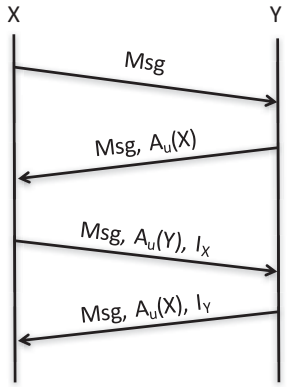


Abbildung 3.10: Schematische Darstellung des Kommunikationsprotokolls zur Übermittlung einer Nachricht, der gegenseitig gemessenen Bandbreite A und des Leistungsindex I zwischen zwei kommunizierenden Knoten.

aktualisierten Upload-Bandbreite $A_u(X)$ an X übermitteln. Die Übermittlung des Leistungsindex eines Knotens erfolgt automatisch in allen Nachrichten und vergrößert jede Nachricht um 1 Byte. Wird zusätzlich eine aktualisierte Bandbreite übermittelt, werden weitere 8 Byte benötigt.

3.3 Knotenbeitritt (join)

Neben der Verwaltung der Replikate in autonomen Nachbarschaften ist insbesondere das Management der einzelnen Peers für die Organisation des Peer-to-Peer-Netzwerks von zentraler Bedeutung. Für jeden Peer werden drei grundlegende Operationen unterstützt: *join*, *leave* und *crash*. Bei den ersten beiden ist der Peer selbst Initiator der Operation, wohingegen seine Nachbarschaft für die letzte Operation verantwortlich ist, damit ein fehlerfreier Status des Gesamtsystems wiederhergestellt werden kann.

Zur Teilnahme am Netzwerk muss ein Peer p Kontakt zu einem Bootstrapknoten b aufnehmen (vgl. Abb. 3.11). Die möglichen Bootstrapknoten werden initial mit dem Installationspaket ausgeliefert und können dynamisch um Super-Peers und Knoten mit hoher Verfügbarkeit aus den bereits bekannten Nachbarschaften ergänzt werden. Am Bootstrapknoten wird für p anschließend anhand des in Kapitel 4.2.2 vorgestellten Verfahrens ein geeigneter virtuellen Knoten v für die Replikation ermittelt. Das jüngste Replikat r_n von v wird nun mit der Koordinierung des Beitritts von p beauftragt.

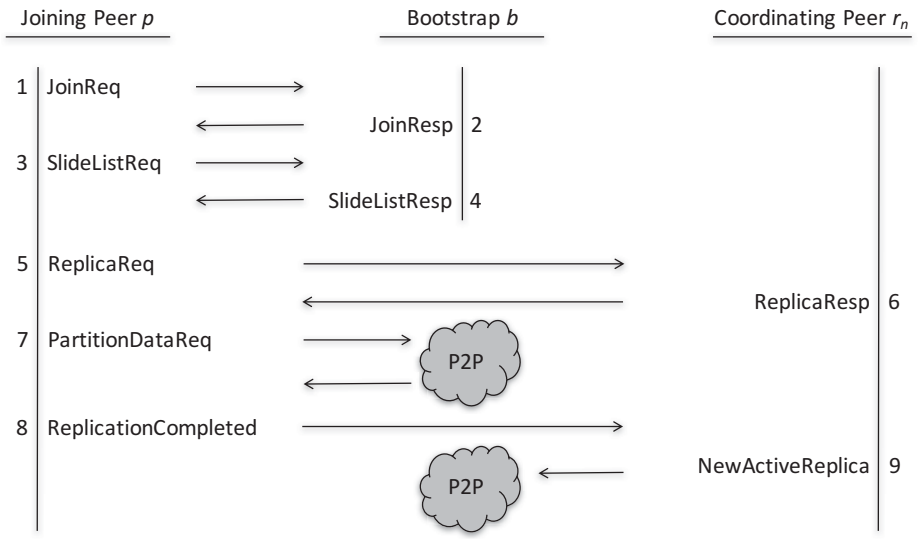


Abbildung 3.11: Schematische Darstellung des Kommunikationsprotokolls zur Erzeugung eines Replikats (*join*) über einen Bootstrap Knoten.

Zur Berechnung des Pfades zu v fordert p die aktuelle Liste der Datensätze von b an, womit die Kommunikation zwischen diesen beiden Knoten beendet ist. Den Pfad zu v kann p anschließend selbstständig über den in Kapitel 3.1 vorgestellten Routing-Algorithmus (vgl. Alg. 3.1) berechnen.

Über das jüngste Replikat r_n der Replikatsnachbarschaft $R(v)$ erhält p die Liste aller Replikate $r_i \in R(v)$ und kann von dort parallel die einzelnen Bestandteile der Partition anfordern. Sind alle Daten übertragen, wird von dem koordinierenden Replikat r_n die Information über ein neues Replikat auf p an die Nachbarschaft verteilt und p in allen Nachbarschaftlisten als letztes Element eingetragen. Sollte ein weiterer Knoten hier als neues Replikat aufgenommen werden, würde dieser Prozess nun von p koordiniert. Der Aufbau der für diese Operation benötigten Nachrichten ist detailliert in Anhang B.2 beschrieben.

Unabhängig von einem möglicherweise erforderlichen Transport der Partitionsdaten sind für den Beitritt eines Peers zum Netzwerk laut Protokoll (vgl. Abb. 3.11) sieben Systemnachrichten erforderlich. Für die Synchronisierung der eigenen Replikate werden weitere $|R(v)|$ Nachrichten benötigt. Die c virtuellen Nachbarn des aktualisierten Knotens und ihre Replikate müssen ebenfalls über das neue Replikat informiert werden, was zusätzliche $R(v_i)$ Nachrichten für jeden Nachbarn erfordert. Aus Gleichung 3.23 lässt sich die Anzahl der erforderlichen

Nachrichten M zur Integration eines neuen Peers p als Replikat des Knotens v und der Synchronisierung aller Replikatslisten ermitteln.

$$M = |R(v)| + \sum_{i=1}^{c} |R(v_i)| + 7 \tag{3.23}$$

Die Nachrichten zur Replikat-Synchronisation werden innerhalb der jeweiligen Nachbarschaften verteilt und müssen nicht von dem aktualisierten Knoten organisiert werden, was jedoch keinen Einfluss auf die Anzahl der benötigten Nachrichten hat, sondern lediglich der Lastverteilung dient.

Abhängig von der Menge der Daten, die der virtuelle Knoten enthält, werden entsprechend viele Request- und Response-Nachrichten benötigt, deren Anzahl sich aus Gleichung 2.6 (s. S. 28) berechnen lässt. Die Requests enthalten neben dem allgemeinen Header (vgl. Tab. 4.1 auf S. 72) zusätzlich die in Anhang B.2 dargestellten Informationen und kommen damit auf eine Gesamtlänge von 42 Byte. Die Antworten enthalten zusätzlich die Bilddaten.

Sollte der beitretende Peer die zugewiesene Partition noch nicht während eines vorherigen Besuchs erhalten haben, müssen die entsprechenden Kacheln übertragen werden. Für eine vollständig gefüllte Partition mit einer Kantenlänge von 64 Kacheln ergibt sich rechnerisch unter Verwendung der in Kapitel 2.2 ermittelten durchschnittlichen Kachelgröße von 31.698 Byte, in einem Präparat mit zehn Vergrößerungsstufen und einer Fokusebene, eine Übertragungslast von 8.184 Requests zu je 42 Byte und eben so vielen Antworten zu durchschnittlich je 31.744 Byte im Gesamtvolumen von 260.136.624 Byte.

*3.4 Knotenaustritt (*leave*)*

Bei einem regulären, nicht fehlerbedingten Ausscheiden eines Peers p aus der Replikatsnachbarschaft $R(v)$ eines Knotens v informiert p alle Replikate $r_i \in R(v)$ über sein Ausscheiden (vgl. Abb. 3.12). Daraufhin ist jedes Replikat r_i eigenständig dafür verantwortlich p aus seiner eigenen Replikatsliste zu entfernen und die Verbindung zu p zu schließen.

Nachdem alle Verbindungen geschlossen sind, informiert p das jüngste Replikat r_n, dass das Verlassen der Nachbarschaft abgeschlossen ist. Daraufhin informiert r_n zur Vermeidung von Abstimmungsfehlern oder eventuellen Timeouts alle $r_i \in R(v)$ über die aktuelle Nachbarschaftsliste. Sollte p das jüngste Replikat r_n in $R(v)$ sein, wird stattdessen r_{n-1} informiert.

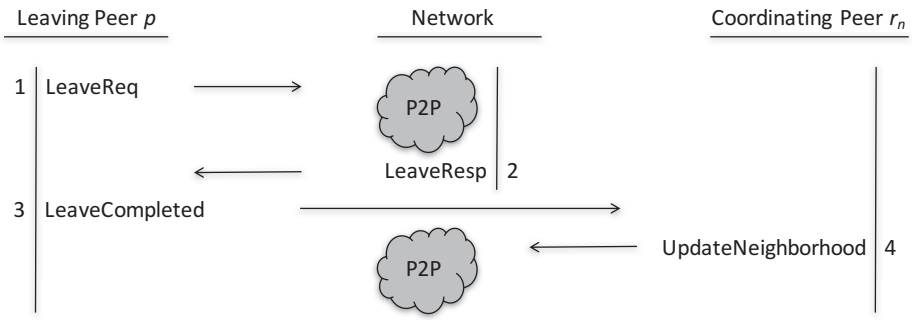


Abbildung 3.12: Schematische Darstellung des Kommunikationsprotokolls zur Auflösung eines Replikats (*leave*).

Der Abgleich der Replikatsliste ist vor allem für die Rückkehr zu einem konsistenten Gesamtsystem erforderlich, falls die Kommunikation mit einem Replikat während der Operation, z.B. aufgrund eines Timeouts, fehlgeschlagen ist.

Für das Verlassen einer Nachbarschaft werden laut Protokoll (vgl. Abb. 3.12) $r+1$ Systemnachrichten benötigt, wobei r die Anzahl der Replikate für den betroffenen Knoten bezeichnet. Zusätzlich sind die c virtuellen Nachbarn des Knotens über eine Änderung der Replikate zu informieren, was weitere r_c Nachrichten für jeden Nachbarn erfordert, die jedoch über die Replikat-Synchronisation der jeweiligen Nachbarschaft verteilt werden. Die Gesamtanzahl M benötigter Nachrichten für das ordnungsgemäße Verlassen einer Nachbarschaft kann anhand von Gleichung 3.24 berechnet werden.

$$M = 2 \cdot |R(v)| + \sum_{i=1}^{c} |R(v_i)| + 1 \tag{3.24}$$

Vorhandene Daten werden nicht direkt gelöscht, wenn ein Peer eine Nachbarschaft verlässt. Sollte eine spätere Wiedereingliederung in die gleiche Nachbarschaft erforderlich sein, müssen nicht alle Partitionsdaten erneut übertragen werden. Ist der ausgeschiedene Knoten jedoch z.B. für temporäre Replikate verantwortlich, werden diese automatisch gelöscht.

3.5 Zusammenfassung

Dieses Kapitel stellte die zentralen Bausteine des neu entwickelten, unstrukturierten Omentum Overlays vor. Neben der robusten Architektur des eingeführten Overlays wurde eine effiziente Routingstrategie mit einer zweistufigen Pfad-

kompression vorgestellt. Die größere Abhärtung des Omentum Overlays gegenüber vielen gleichzeitigen Knotenausfällen wird mit einem grundsätzlich höheren Overhead zur Verwaltung des Zufallsgraphen bezahlt. Vergleichend wurden unterschiedliche Implementierungen strukturierter Overlays analysiert und gezeigt, dass strukturierte Overlays für Index-basierte Suchen in dynamischen Umgebungen in der Regel keine kürzeren Pfade ermöglichen, sondern mitunter zu komplexeren Nachbarschaften führen, womit sie schlechter skalieren als die vorgestellte, unstrukturierte Architektur für Omentum.

Die Einführung einer effizienten Routingstrategie mit Pfadkompression reduziert den durch die unstrukturierte Architektur eingeführten Overhead. Dazu werden die von allen Peers verwalteten Knotenlisten verwendet, die, im Gegensatz zum Algorithmus für die Routenberechnung, auch eingehende Kanten berücksichtigen. Die verbesserte Routingstrategie erlaubt Omentum eine ähnliche Leistung für Index-basierte Lookups, wie eine vergleichbare DHT, mit einer durchschnittlichen Pfadlänge von $\frac{\log N}{\log c}$, wobei die Anzahl der verwendeten Nachbarn c klein und unabhängig von der Größe des Netzwerks N ist. Aufgrund der Verteilung einzelner Datensätze auf mehrere Replikate (vgl. Kap. 4) wird die Pfadkompression um eine weitere Stufe ergänzt, die zur Laufzeit zusätzliche Verkürzungen durch die gemeinsame Replikatsnachbarschaft gestattet. In Omentum konnte die für ein effektives Routing erforderliche Nachbarschaftsgröße im Vergleich zu strukturierten Overlays bei nahezu gleichen Ergebnissen um bis zu 80% reduziert werden. Gleichzeitig wurde die in BubbleStorm realisierte Suche über die Lösung eines Rendezvous-Problem durch eine effiziente Zuordnung von Annotationen in der Komplexität signifikant reduziert, da nicht mehr alle Knoten des Netzwerks betrachtet werden müssen, sondern lediglich eine Partition jedes Datensatzes.

Weiterhin wurde anhand von etablierten Verfahren zur Bandbreitenmessung die Entwicklung eines neuen Ansatzes zur Abschätzung der Bandbreite durch passive probing vorgestellt, der mittels Paket Transmission Rate unter TCP mit Congestion Control arbeitet. Die erzielte Genauigkeit schwankt in schnellen Netzwerken durch eine Pufferung der Pakete auf der Transportschicht stärker, erreicht aber im Mittel die verfügbare Bandbreite mit einer akzeptablen Toleranz von unter 5%. Das Ergebnis der Bandbreitenmessung wird unscharf mit der CPU- und Speicherauslastung zu einer einzelnen, diskreten Knotenbewertung aggregiert, die einen guten Anhaltspunkt zur Leistungsbewertung eines Knotens liefert. Die Bearbeitung von Nachrichten berücksichtigt die so ermittelten Leistungseigenschaften der beteiligten Knoten und unterstützt damit eine adäquate Lastverteilung.

KAPITEL 4

Adaptive Replikation

Virtuelle Knoten im Overlay repräsentieren die Datenpartitionen der in Kapitel 2.2.3 erzeugten Datensätze. Die im Folgenden betrachteten Replikate r stellen virtuelle Nachbarschaften R_{P_i} einer einzelnen Datenpartition P_i dar, die durch eine Kopie auf den beteiligten Peers gebildet wird (vgl. Abb. 4.1). Innerhalb der Nachbarschaft sind alle Replikate einer Partition untereinander verbunden. Die so realisierte Abstimmung der einzelnen Replikate macht zusätzliche Lebenszeichen überflüssig, da ohnehin auf regelmäßiger Basis Informationen ausgetauscht werden. Weiterhin können die erhaltenen Leistungsparameter der übrigen Replikate von jedem Peer lokal aggregiert und daraus ein Leistungsindex für die gesamte virtuelle Nachbarschaft berechnet werden. Ein bedeutender Nebeneffekt ist die zusätzliche Abhärtung, die das Overlay aufgrund der Replikatsvernetzung erfährt, denn so wird die Möglichkeit des vollständigen Versagens einer Route im Zufallsgraphen signifikant reduziert.

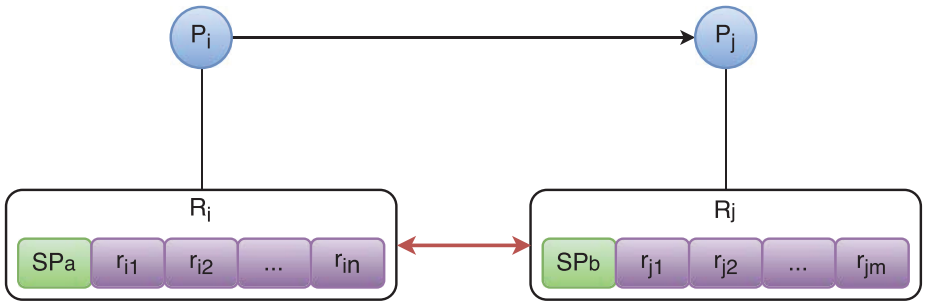


Abbildung 4.1: Schematische Darstellung einer Beziehung der Replikatsnachbarschaften von P_i und P_j. Die Replikate sind absteigend nach Alter sortiert und die Liste wird vom Super-Peer angeführt, der den Datensatz initial zur Verfügung gestellt hat.

Jeder Peer verwaltet die Replikatsnachbarschaft intern als sortierte Liste, in der die Replikate mit abnehmenden Alter gespeichert sind, und ist eigenständig für ihre Aktualisierung anhand von erhaltenen Informationen verantwortlich. Die Sortierung ist vor allem bei der Erzeugung oder Auflösung von Replikaten relevant (vgl. Kap. 4.2), da so eine zusätzliche Abstimmung der Peers untereinander, z.B. während der Fehlerbehandlung, nicht mehr erforderlich ist.

Ein weiterer, starker Vorteil der virtuellen Replikatsnachbarschaft ist, dass regelmäßig Nachrichten ausgetauscht werden. Verbindungsabbrüche oder Ausfälle einzelner Peers können so zeitnah erkannt werden. Möchte ein Peer nach einem Ausfall wieder am Netzwerk teilnehmen, so sieht das entwickelte Konzept explizit vor, die bereits zuvor bereitgestellten Daten erneut zur Verfügung zu stellen, sofern anderweitig kein dringender Bedarf besteht und der Peer entsprechende Leistungsreserven besitzt. Daher versuchen neu gestartete Peers, sofern die Zeit ihrer Abwesenheit im Rahmen einer festgelegten Toleranz liegt, zunächst Kontakt zu den Peers ihrer ehemaligen Nachbarschaft aufzunehmen, bevor sie einen regulären Beitrittsversuch zum System starten.

Die Integration der Super-Peers in allen Nachbarschaften, die ihre Daten enthalten, hat für die Bereitsteller den zusätzlichen Nutzen, dass eine Verbreitung und Anfragehäufigkeit ihrer Datensätze sehr leicht ermittelt und aggregiert werden kann.

4.1 Kommunikation mit lokaler Fehlerkorrektur

Kommunikation ist in Peer-to-Peer-Netzwerken von zentraler Bedeutung. Einerseits ist das Netzwerk von starken Fluktuation der Peers betroffen und andererseits sollte der Overhead für systemimmanente Kommunikation möglichst gering gehalten werden. Daher wurde für einen stabilen Peer-to-Peer-Betrieb im vorliegenden Anwendungskontext eine effiziente und fehlertolerante Kommunikationsschnittstelle entwickelt [115]. Nach einer kurzen Einführung in die Grundlagen der Fehlererkennung und Behandlung wird das entwickelte Kommunikationsprinzip detailliert beschrieben.

Ein Fehler ist ein Teil eines Systemzustands, der zu einem Funktionsausfall führen kann, womit eine unzulässige oder aussetzende Funktion einer beteiligten Komponente bezeichnet wird [161]. Grundlegend werden Fehler in drei Bereichen unterschieden: Design, Produktion und Betrieb [70]. Vorsätzliche Fehler werden in dieser Einteilung nicht separat aufgeführt, können aber vor dem Hin-

tergrund einer gezielten und eventuell betrügerischen Manipulation in der Regel nicht von gewöhnlichen Fehlertoleranzmechanismen entdeckt und behandelt werden. Hauptsächlich werden im Folgenden die Betriebsfehler betrachtet, die z.B. durch Hardwarefehler, Bedienerfehler oder unsachgemäße Wartung verursacht werden. Zwei unterschiedliche Szenarien müssen dabei Beachtung finden. So kann ein Fehler direkt zu einem Ausfall einer Komponente (Fail-Stop Failures) oder zu unvorhersehbaren Ergebnissen (Byzantine Failures) führen, die im weiteren Verlauf wiederum andere Komponenten beeinflussen können [147]. Mit einzubeziehen sind des Weiteren der Ort an dem ein Fehler entstanden ist und seine Dauer, wobei hier zwischen intermittierenden und permanenten Fehlern differenziert wird.

Definitionsgemäß wird einem System die Eigenschaft der Fehlertoleranz zugesprochen, wenn es trotz eines fehlerhaften Zustands eines Teils seiner Komponenten noch in der Lage ist korrekt zu arbeiten [149]. Fehlertoleranzverfahren müssen unterscheiden können, ob das Gesamtsystem oder lediglich eine einzelne Komponente von einem Fehler betroffen ist. Weiterhin sollte die Beseitigung eines Fehlers vom System erkannt und mit der erneuten Bereitstellung der ursprünglichen Funktion darauf reagiert werden. Dazu kann ein fehlertolerantes System auf mehrere Verfahren zurückgreifen, die der Fehlerdiagnose und Fehlerbehandlung dienen.

Die Fehlerdiagnose zielt darauf ab, einen Fehler und seinen Ursprung zu erkennen, um folglich eine adäquate Fehlerbehandlung zu ermöglichen, die eine Fehlerausgrenzung, -behebung oder -kompensation bedeutet [77]. Speziell vor diesem Hintergrund muss beachtet werden, dass Fehlertoleranz bei Funktionen, die auf externe Hilfsmittel angewiesen sind, wie beispielsweise auf ein Netzwerk für die Kommunikation, nicht zwangsläufig bedeutet, dass die Komponente auch bei Wegfall des Hilfsmittels weiterhin korrekt funktioniert. Vielmehr bedeutet es, dass sie in der Lage ist, den Fehler zu erkennen, zu tolerieren und mit angemessenen Maßnahmen darauf zu reagieren ohne die Funktion des Gesamtsystems zu beeinträchtigen [77].

In vielen Fällen lassen sich Fehler durch Redundanz in Informationen oder Hardware behandeln sowie durch Maskierung vor anderen Komponenten oder durch spezifische Akzeptanz-Validierung berechneter Ergebnisse. Sollte ein Akzeptanztest fehlschlagen, besteht die Möglichkeit über einen Recoveryblock die Systemkomponente in einen definierten und fehlerfreien Ausgangszustand zu versetzen und die fehlererzeugende Operation bis zum Erreichen einer im Vorfeld

festgelegten Versuchsanzahl oder der Akzeptanz zu wiederholen [77]. Ebenfalls ist ein Anlegen von Checkpoints oder Recoverypoints möglich, die eine Momentaufnahme einzelner Prozesse darstellen, z.B. in Bezug auf Zugriffe, Zustand oder Nachrichtenübertragung. Sollte ein Fehlerfall eintreten, kann auf den aktuellsten, fehlerfreien Checkpoint gewechselt werden und der Betrieb regulär wieder aufgenommen werden. Ein Problem hierbei ergibt sich aus der globalen Konsistenz, die, bei einer lokalen Rückkehr zu einem älteren Zustand (backward recovery) oder einer Fehlerbeseitigung anhand eines definierten Maßnahmenkatalogs (forward recovery), nicht mehr gewährleistet sein muss. Für koordinierte Checkpoints sind bei einer Wiederherstellung oder Rücksetzung alle Knoten eines Systems betroffen. Im Gegensatz dazu ist bei einem individuellen oder nicht-koordinierten Checkpoint nur eine einzelne Systemkomponente betroffen [161].

Die gesamte Kommunikation muss im Fehlerfall adäquat reproduzierbar sein und darf nicht zu einer massiven oder unkalkulierbaren Belastung der beteiligten Knoten führen. Aus diesem Grund behandeln die beiden, an einer Kommunikation beteiligten Parteien die wiederholten Nachrichten jeweils mit eigenen Methoden zur Fehlertoleranz (vgl. Abb. 4.2). Dabei wird bei jeder Nachricht anhand einer Kommunikationsidentifikation unterschieden, ob es sich um eine erwartete Nachricht handelt oder nicht.

Der Sender erstellt eine Nachricht für einen Empfänger und markiert sie als noch nicht gesendet. Vor dem Senden wird ein entsprechender Checkpoint, der bereits die ungesendete Nachricht enthält, angelegt und die Nachricht nach erfolgreichem Versand in eine Ausgangsliste gelegt, sofern eine Antwort erwartet wird. Andernfalls wird die Nachricht direkt in die empfängerspezifische Kommunikationsliste verschoben.

Auf Empfängerseite wird direkt nach dem Empfang ebenfalls ein Checkpoint mit der erhaltenen Nachricht angelegt und diese in der Eingangsliste erfasst. Sofern eine Antwort erforderlich ist, wird diese gesendet und bei einem Übermittlungserfolg aus der Eingangsliste in die Kommunikationsliste des zugehörigen Senders verschoben.

Ein elementares Kriterium für das Design der Kommunikationsschnittstelle ist die Reduktion der Größe aller Nachrichten. Für wenige Knoten erscheint dies nicht sonderlich relevant, jedoch summieren sich diese Beträge bei zunehmender Anzahl an Knoten zu sehr großen Datenmengen im gesamten Netz. Aus diesem Grund wurde während der Implementierungsphase die Java-eigene Serialisierung von Objekten für die Nachrichtenübermittlung durch eine eigene, effizientere Me-

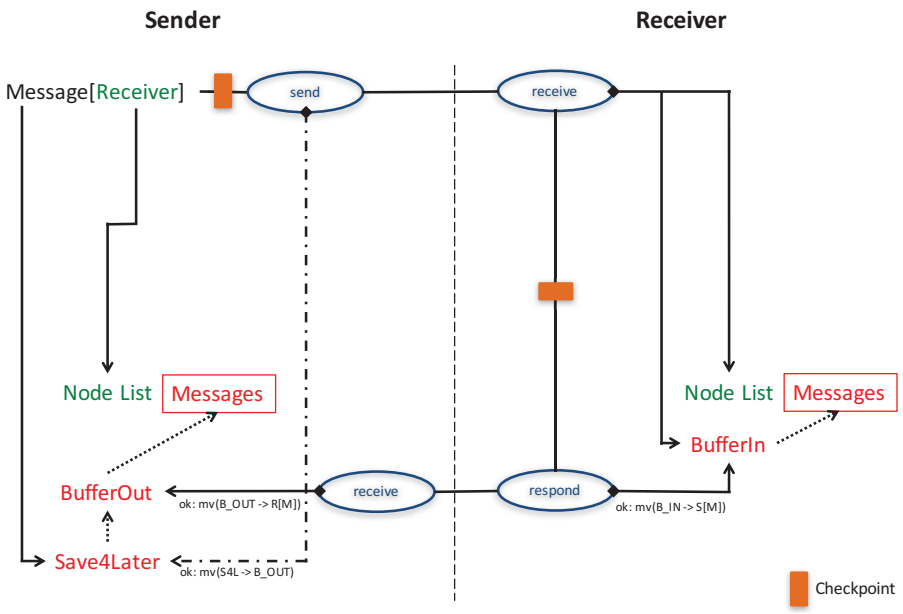


Abbildung 4.2: Schematische Darstellung des Kommunikationikationsprotokolls unter Verwendung von Checkpoints.

thode ersetzt. Diese erreicht eine durchschnittliche Reduktion der Nachrichtengröße um 96%.

Zur optimierten Verwaltung werden die letzten drei Nachrichten, die zwischen zwei Knoten ausgetauscht werden, in einer knotenspezifischen Liste vorgehalten. Im Vergleich zu einer globalen Nachrichtenliste mit einer begrenzten Anzahl an Einträgen, ist eine einfache Wiederherstellung für alle beteiligten Kommunikationspartner möglich und verringert gleichzeitig die Größe der zu speichernden Listen.

Im Falle eines Fehlers, der zum kurzfristigen Ausfall der Kommunikationsinfrastruktur oder einem vollständigen Anhalten eines Knotens führt, wird im Rahmen des Neustarts ein existierender Wiederherstellungspunkt von der Fehlererkennung verarbeitet. Sind die enthaltenen Daten veraltet, werden die entsprechenden Wiederherstellungspunkte gelöscht.

Grundsätzlich enthalten alle Nachrichten einen Header, bestehend aus Informationen zu Sender und Empfänger sowie zur Nachricht selbst (vgl. Tab. 4.1). Zu letzteren zählen z.B. eine Typisierung der Nachricht wie auch eine eindeutige Zuordnung zu einer bestimmten Kommunikation zwischen zwei beteiligten Parteien.

Tabelle 4.1: Aufbau des gemeinsamen Headers aller Nachrichten (Länge: 34 Byte).

Type	**MessID**	**CorrID**	**SendID**	**SendIP**	**RecID**	**RecIP**	**CP**
1 Byte	8 Byte	8 Byte	4 Byte	4 Byte	4 Byte	4 Byte	1 Byte

Type Über diesen Parameter werden Nachrichtentypen differenziert.

MessID Die MessageID identifiziert eine Nachricht für jeden Sender eindeutig und findet hauptsächlich in der Nachrichtenverwaltung und dem Checkpointing Verwendung.

CorrID Über die CorrespondenceID wird eine Nachricht zwischen beteiligten Kommunikationspartnern eindeutig einer spezifischen Unterhaltung zugeordnet.

SendID/RecID Diese IDs werden aus der Hardware der beteiligten Geräte generiert. Sie sind systemweit eindeutige Identifikationsnummern für den Sender und Empfänger.

SendIP/RecIP Diese Parameter enthalten die IP-Adressen des Senders und des Empfängers.

CP Mit diesem Schalter wird das System darüber informiert, ob die Nachricht in den Checkpointing-Prozess zu integrieren ist.

Das Mitführen von Sender- und Empfängerinformationen mag redundant erscheinen, erweist sich jedoch bei Nachrichten, die im Netzwerk über mehrere Knoten geroutet werden müssen oder im Hinblick auf Network Address Translation (NAT) [151] als förderlich (vgl. Kap. 6.4). Ein weiterer Vorteil ist, dass der Empfänger einer Nachricht direkt Antworten versenden kann, ohne zusätzliche Kenntnis über die bisherige Kommunikation zu haben. Zur Optimierung der Checkpoints werden ausschließlich systemrelevante Nachrichten[24] in die Sicherung integriert, womit eine Wiederherstellung von zwischenzeitlich obsoleten Anfragen verhindert wird (vgl. Tab. 4.2).

So sind z.B. Nachrichten, die der Übertragung von Bilddaten dienen, ausgeschlossen und werden in einem Fehlerfall von der entsprechenden Komponente

[24] Eine Übersicht aller Nachrichten und deren Beschreibung findet sich in Anhang B.

Tabelle 4.2: Übersicht über die Berücksichtigung ausgewählter Nachrichten im Rahmen von Sicherheits- (Checkpoints) und Leistungsaspekten (Bewertung).

Typ	Größe (Bytes)	Checkpoints	Bewertung
TileRequest	58	nein	nein
TileResponse	58 + Bild	nein	ja
SlideListRequest	38	nein	nein
SlideListResponse	34 + Liste	nein	ja
JoinRequest	34	ja	nein
JoinAccept	41 + Nachbarn	ja	ja
NewReplica	41 + Replikate	nein	nein

erneut angefordert. Der Vorteil dieses Vorgehens liegt in der Verkleinerung der Checkpoints und der Nachrichtenliste, die für jeden Knoten gespeichert wird, wodurch auch die Zeit für die Suche nach bestimmten Ereignissen reduziert wird.

4.2 Replikationsmodell

Grundsätzlich werden die Daten im Omentum Overlay, motiviert durch *Leng* [107], in unterschiedliche Kategorien eingeteilt, die im Hinblick auf eine mögliche Replikation differenziert behandelt werden müssen (vgl. Abb. 4.3). Im Gegensatz zu Leng werden jedoch nicht verschiedene Replikationsstrategien differenziert, sondern Objektgruppen definiert, deren Eigenschaften großen Einfluss auf die Replikation und den Unterhalt der einzelnen Objekte haben.

Dynamische Objekte beschreiben hochveränderliche Daten, wie z.B. jegliche Art von Benutzer-generiertem Inhalt. Die hohe Änderungsfrequenz erfordert eine Versionierung und ein zeitnahes Update der Daten auf betroffenen Knoten. Zur besseren Einteilung dieser heterogenen Gruppe werden die dynamischen Objekte in vier Klassen eingeteilt.

Einmalige Objekte bezeichnen alle Objekte, die eine sehr kurze Lebenszeit haben. Sie werden weder repliziert noch können sie Ziel von Aktualisierungen sein. Typische Beispiele dieser Gruppe sind Anfragen oder Verwaltungsnachrichten zwischen einzelnen Knoten oder Knotengruppen.

Temporäre Objekte fassen alle Objekte zusammen, die eine etwas längere Lebenszeit haben, aber nicht persistent sind. Aus diesem Grund werden ihre Replikate nicht gespeichert und nach Ablauf der Lebenszeit (engl. time to live, kurz TTL) gelöscht. Vertreter dieser Gruppe sind z.B. *PriorityAssistanceRequests* (PAR), die u.a. von Knoten gesendet werden können, wenn ihre Leistungsfähigkeit bestimmte Werte unterschreitet, während ihr Arbeitsaufkommen hoch ist. Des Weiteren zählen Kontextaktualisierungen von anderen Benutzern, die den gleichen Datensatz bearbeiten, dazu.

Verwaltete Objekte sind sehr stark an den Knoten gebunden, der sie erzeugt hat. Sie existieren nur während dieser Knoten am Overlay beteiligt ist. In diese Gruppe fällt unter anderem die Organisation der Nachbarschaften eines Peers.

Persistente Objekte müssen auch dann verfügbar bleiben, wenn der erzeugende Knoten das Netzwerk verlässt. Aus diesem Grund muss eine Mindestanzahl von Replikaten im Netzwerk vorgehalten werden. Annotationen und Kommentare von Benutzern machen die häufigsten Objekte in dieser Gruppe aus.

Statische Objekte repräsentieren Bilddaten, Kacheln, Partitionen oder ganze Datensätze. Sie sind in der Regel keinen Änderungen unterworfen und müssen vor allem zur Lastverteilung repliziert werden. Lediglich eine Löschung muss hier von den Replikaten verwaltet werden. Diese Gruppe stellt das mehrheitliche Datenvolumen in Omentum dar.

Neben der Replikation einzelner Objekte sind vor allem Eigenschaften zur Aktualisierung oder zur Auflösung von Konflikten relevant, die für jeden Objekt-Typ individuell betrachtet werden müssen (vgl. Tab. 4.3).

Annotationen werden in einer separaten Partition der Datensätze gespeichert. Ihre Replikation verläuft damit analog zu den Bilddaten. Nichtsdestotrotz führt ihr dynamischer Charakter zu einer anderen und, vor allem bedingt durch Updates, teureren Behandlung in Omentum, denn jegliche Änderung muss umgehend allen Replikaten mitgeteilt werden. Zur Lösung von auftretenden Konflikten werden Zeitstempel und Versionsnummern verwendet.

[25] Konflikte können hier nicht entstehen, da Änderungen an diesem Objekt nur aus einer Quelle erfolgen können.

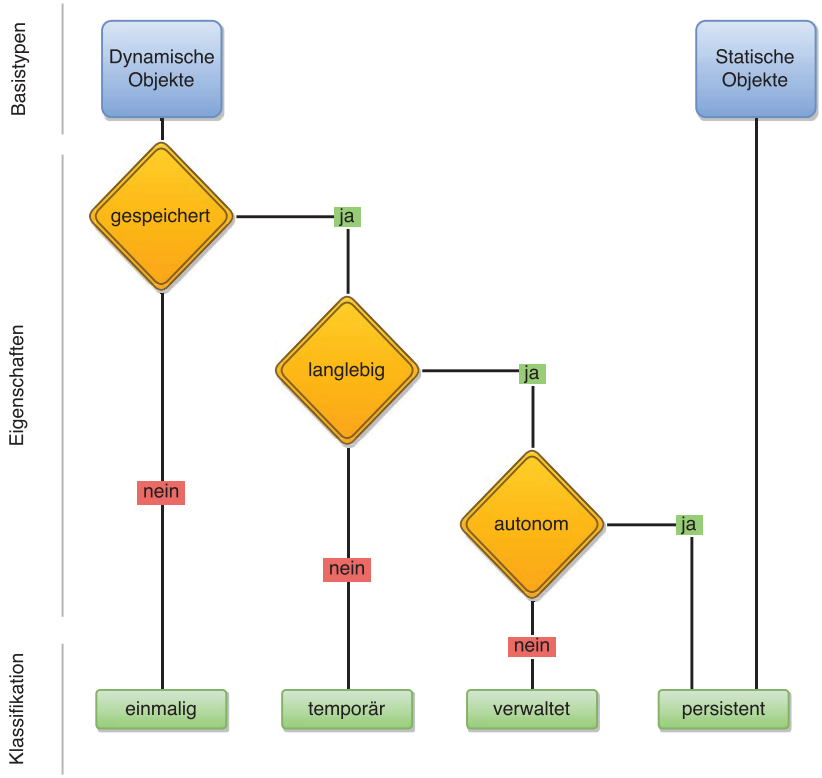


Abbildung 4.3: Schematische Darstellung der Einteilung unterschiedlicher Objektklassen im Omentum Overlay.

Obschon Annotationen dynamisch sind und damit mehr Nachrichten zu ihrer verteilten Aktualisierung erforderlich sind, ist das Datenaufkommen, verglichen mit den Anforderungen an die Übertragung der Bilddaten, sehr gering. Der Vorteil liegt hier in der internen Darstellung der Annotationen, die als Polygone und nicht als graphische Objekte definiert sind.

4.2.1 Selbstorganisation

Die Verwaltung der Replikate in einer eigenen Nachbarschaft ermöglicht die Weiterleitung von Anfragen auf weniger stark belastete Knoten, die über den in Kapitel 3.2.5 vorgestellten Leistungsindex identifiziert werden können. Eine Übertragung der Verantwortung für das Erstellen oder Auflösen eines Replikats auf einen einzelnen Knoten verringert den Aufwand für die zeitliche Koordination entsprechender Requests in der gesamten Nachbarschaft. Sollte der für das Re-

plikat verantwortliche Peer, z.B. aufgrund hoher Belastung oder eines Ausfalls, nicht mehr in der Lage sein, den Task abzuschließen, wird diesem automatisch der Task entzogen und der nächste Peer in der Liste mit der Aufgabe betraut.

Die Knoten, die neue Daten im Netzwerk bereitstellen dürfen, sind ebenfalls in einer separaten Nachbarschaft untereinander verbunden. Sie fungieren als Super-Peers, denn sie sind in alle Nachbarschaften eingebunden, an denen ihre virtuellen Knoten beteiligt sind. Das Overlay dient ihrer Unterstützung im Hinblick auf die Erreichbarkeit und Lastreduktion.

Die Super-Peer-Nachbarschaft ermöglicht zusätzlich die leichte Aggregation von Leistungsdaten und Verfügbarkeitszeiten aller Peers in den jeweiligen Nachbarschaften, solange die betreffenden Super-Peers online sind.

Die Replikation wird innerhalb der Replikatsnachbarschaft vom jeweils jüngsten Replikat zentral organisiert. Auf diese Weise wird zusätzlicher Overhead zur Abstimmung zwischen den einzelnen Replikaten vermieden. Weiterhin werden die Koordinierungsprozesse auf diese Weise zwischen mehreren Replikaten verteilt und müssen nicht ausschließlich vom Super-Peer verantwortet werden. Scheiden koordinierende Replikate während eines laufenden Vorgangs aus, kann ihre Kommunikation nahtlos vom dann verantwortlichen Vorgänger fortgesetzt werden.

4.2.2 Erzeugung neuer Replikate

Neue Replikate werden anhand festgelegter Kriterien in absteigender Wichtigkeit vom System erzeugt. Von zentraler Bedeutung ist hierbei die Auswahl einer geeigneten Partition für die Replikation (vgl. Alg. 4.1). Die Kriterien, die zur Unterscheidung herangezogen werden, sind eine mögliche Untersättigung, eventuell bestehende Unterstützungsanforderungen (PAR) und der Leistungsindex (vgl. Tab. 4.4). Sie werden mit einer entsprechenden Gewichtung versehen, um

Tabelle 4.3: Übersicht der identifizierten Objekt-Typen und ihrer Eignung für die Replikation im Omentum Overlay (modifiziert nach [107]).

Objekt	**Replikation**	**Unterhalt**	**Persistenz**	**Aktualisierung**	**Konfliktauflösung**
einmalig	–	–	–	–	–
temporär	+	+	–	–	–
verwaltet	+	+	–	+	X[25]
persistent	+	+	+	+	+

Algorithmus 4.1 Auswahl eines neuen Replikatstandorts (vereinfachte Darstellung)

1: $r = \text{nil}$; ▷ Peer, der das neue Replikat werden soll
2: $S(P) = \{\ldots\}$; ▷ Liste der Replikatsnachbarschaftsgrößen aller Partitionen
3: $PAR_v = \{\ldots\}$; ▷ Liste der PAR für v
4:
5: **function** SELECTNODEFORREPLICATION(r)
6: Wähle $v_S \in S(P)$, so dass ▷ min. Replikate
7: $\forall v : v \leq v_S \Rightarrow v_S = v$;
8: $I(v_S) = 10 \cdot \left(1 - \frac{S(P_{v_S})-1}{S_{opt}}\right)$
9: Wähle $v_P \in PAR$, so dass ▷ max. PAR
10: $\forall v : \sum_{i=1}^{n} \frac{TTL(PAR_v(i))}{TTL_{max}} \geq \sum_{i=1}^{n} \frac{TTL(PAR_{v_P}(i))}{TTL_{max}} \Rightarrow v_P = v$
11: $I(v_P) = \sum_{i=1}^{n} \frac{TTL(PAR_{v_P}(i))}{TTL_{max}}$
12: Wähle $v_{PI} \in PI$, so dass ▷ max. Leistungsindex
13: $\forall v : PI(v) \geq PI(v_{PI}) \Rightarrow v_{PI} = v$
14: $I(v_{PI}) = \frac{PI_{min}-PI_{opt}}{PI_{min}-PI(v_{PI})}$
15: Wähle $v \in \{v_S, v_P, v_{PI}\}$, so dass
16: $\forall v : I(v) \geq I(v_x) \Rightarrow v = v_x$
17: ADDREPLICA(v,r)
18: **end function**
19:

die Relevanz für das Gesamtsystem zu verdeutlichen. Die Auswahl der Gewichtung wurde experimentell durchgeführt und gewährleistet die Priorisierung einer minimalen Replikatanzahl zur Sicherung der Datenverfügbarkeit (vgl. Abb. 4.4).

Die Koordination neuer Replikate, z.B. im Hinblick auf den Datentransfer oder die Information der übrigen Nachbarschaften, übernimmt das jüngste Replikat der jeweiligen Nachbarschaft, wobei eine einmal gestartete Synchronisierung von diesem Peer auch dann beendet wird, wenn inzwischen ein weiteres Replikat der Nachbarschaft hinzugefügt wurde. In diesem Fall informiert der koordinierende Peer das jüngste Replikat über den neuen Knoten, das daraufhin die übrige Nachbarschaft informiert.

Ist die optimale Anzahl an erforderlichen Replikaten S_{opt} für eine Partition P_i noch nicht erreicht, wird zunächst versucht alle untersättigten Nachbarschaften gleichverteilt aufzufüllen. Mit steigender Sättigung wird der Einfluss dieses Kriteriums kontinuierlich schwächer.

Tabelle 4.4: Übersicht über die Bewertung unterschiedlicher Motivation zur Erzeugung neuer Replikate.

Motivation	Beschreibung	Faktor	Gewicht
Untersättigung	$\|S(P_i)\| < S_{opt}$	$1 - \frac{S(P_i) - 1}{S_{opt}}$	10
Unterstützungs-anforderung	$\exists r \in PAR_{P_i} \wedge n = \|PAR_{P_i}\|$	$\sum_{i=1}^{n} \frac{TTL(PAR_{P_i})}{TTL_{max}}$	1
Leistungsindex	$PI_{min} > PI(v_i) > PI_{opt}$	$\frac{PI_{min} - PI_{opt}}{PI_{min} - PI_{v_i}}$	1

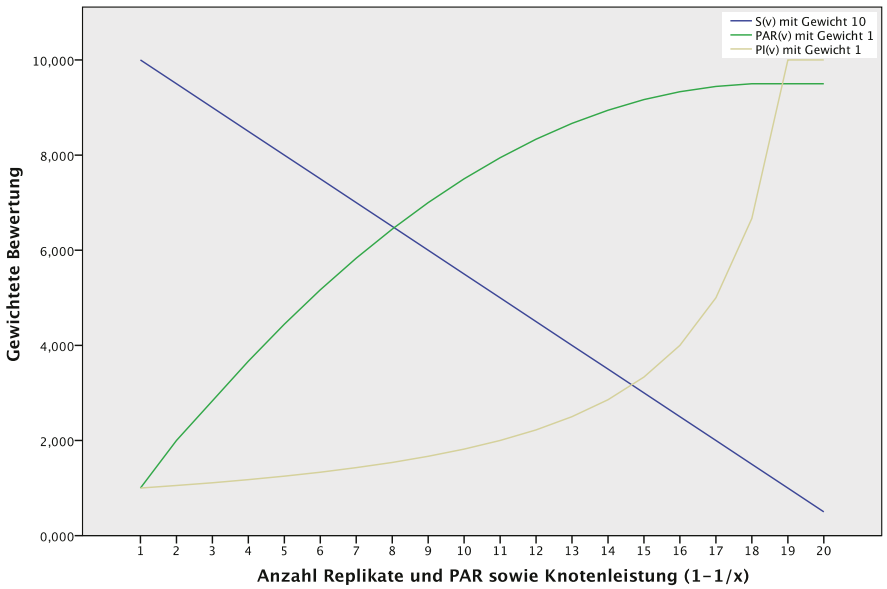


Abbildung 4.4: Vergleich der vergebenen Gewichtungen für die Sättigung (S), die Behandlung von Unterstützungsanforderungen (PAR) und die Berücksichtigung des Leistungsindex (PI) eines Knotens in der Nachbarschaft zur Prioritätsverwaltung während der Replikaterzeugung.

Im nächsten Schritt werden von verbundenen Peers versandte PAR berücksichtigt, da hier Handlungsbedarf aufgrund der schlechten Leistung einer Nachbarschaft besteht. Um einem lokalen Extremum vorzubeugen, muss eine eingeschränkte Leistung über eine gewisse Zeitspanne bestehen, bevor ein PAR ausgelöst wird. Mit sinkender TTL wird der statische, nicht aktualisierbare PAR kontinuierlich schlechter bewertet. Erhält der initiierende Peer in einem festgelegten Zeitintervall keine Antwort auf einen PAR, ohne das der Bedarf gesunken ist, werden in definierten Zeitabständen weitere PAR erzeugt, die sich gegenseitig verstärken und so einen größeren Einfluss auf die Bewertung haben. Antworten mehrere Peers auf einen PAR, wird derjenige mit dem höchsten Leistungsindex als neues Replikat ausgewählt. Hier sind auch alternative Strategien denkbar, die neben der reinen Leistungsfähigkeit z.B. die Uptime des Peers, eine geographische Verteilung der Knoten oder auch bereits erhaltene Datenpakete der betroffenen Partition berücksichtigen.

Weiterhin spielt die aktuelle Leistungsfähigkeit der einzelnen virtuellen Knoten und ihrer Nachbarschaften eine entscheidende Rolle bei der Erzeugung von neuen Replikaten. In einer Partition sind alle Replikate untereinander vernetzt und, aufgrund der jeweils lokal strikten Verwaltung ihrer Datenstruktur, in der Lage die Gesamtleistung eines virtuellen Knotens zu berechnen.

Ist das Netzwerk gesättigt, werden neue Peers den Nachbarschaften mit der schlechtesten Leistung zugeordnet, um die Last dort verteilen zu können. Bei unvollständiger Sättigung im Netzwerk werden die Anforderungen numerisch verglichen und der Umstand mit höherer Dringlichkeit vorrangig behoben. Dabei sinkt die Anforderung für die Sättigung entsprechend der Anzahl der Peers, die bereits Replikate verwalten. So werden z.B. Knoten, die nur von einem oder zwei Peers repliziert werden, vorrangiger mit neuen Peers unterstützt, als Knoten mit bereits vielen bestehenden Replikaten. Der Algorithmus zur Wahl eines geeigneten Peers als neues Replikat verwendet einen reinen Leistungsvergleich. Aufgrund der modularen Integration in das Gesamtsystem kann die verwendete Strategie leicht gegen beliebige andere ausgetauscht werden. Zum einen sind stärker zufallsbasierte Wahlmöglichkeiten realisierbar, bei denen, ähnlich zu BubbleStorm, Random Walks zur Suche geeigneter Peers verwendet werden. Andererseits können auch ortsbasierte Strategien umgesetzt werden, die versuchen Replikate an einem Ort zu konzentrieren oder sie über das gesamte Netzwerk zu verteilen. Die vorgeschlagenen Strategien bedingen jeweils eine individuelle Beachtung der PAR, da hierüber das Antwortverhalten der Peers gesteuert werden kann.

4.2.3 Auflösen von Replikaten

Die Auflösung eines Replikats (vgl. Alg. 4.2) kann nur vom jüngsten Peer der Nachbarschaft gestartet werden und ist an drei Kriterien gebunden (vgl. Tab. 4.5). Betrachtet werden eine mögliche Übersättigung, die individuelle Leistungsfähigkeit der Nachbarschaft sowie die Fluktuation der Replikate. So darf zum einen die minimale Anzahl an Replikaten für den betroffenen virtuellen Knoten nicht unterschritten werden und zum anderen der Leistungsindex des virtuellen Knotens nicht oberhalb von PI_{min} liegen. Weiterhin müssen alle Peers p,

Tabelle 4.5: Übersicht über die Bewertung unterschiedlicher Bedingungen zur protokollkonformen Auflösung von Replikaten.

Motivation	**Beschreibung**	**Faktor**
Übersättigung	$\lvert S(P_i)\rvert \geq S_{opt}$	$\frac{S(P_i)}{S_{opt}}$
Leistungsindex	$PI(v_i) \leq PI_{min}$	$\frac{PI_{min} - PI_{v_i}}{PI_{min} - PI_{opt}}$
Existenzzeitraum	$t_{P_i}(p) \geq t_{min}$	$\frac{t_{P_i}(p)}{t_{min}}$

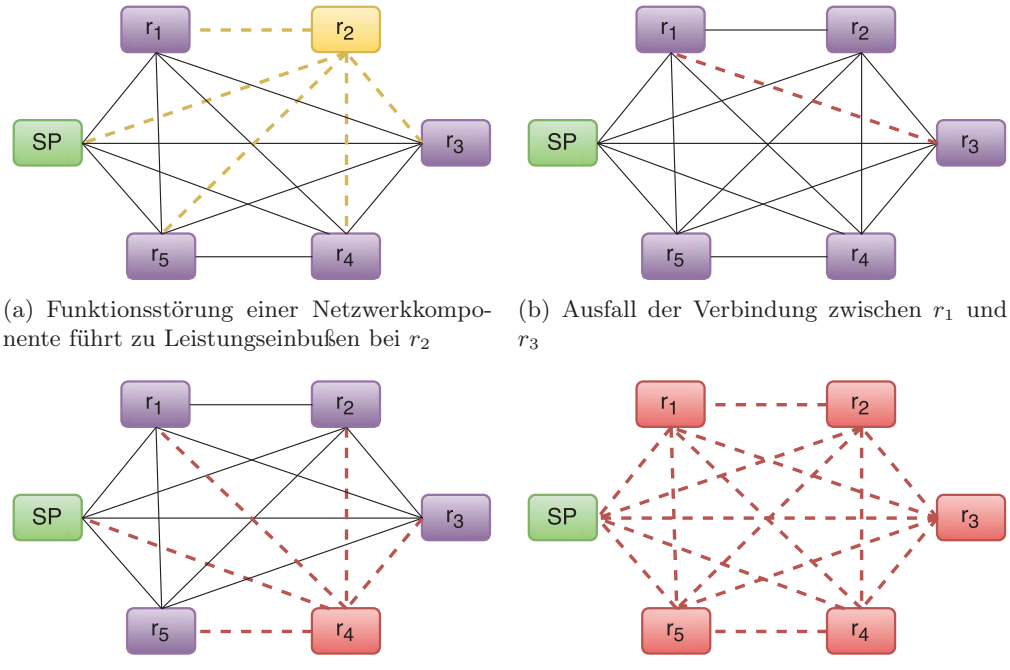
Algorithmus 4.2 Auflösung eines Replikats r des virtuellen Knotens v auf Peer p (vereinfachte Darstellung)

```
1:  v = i;          ▷ Knoten, dessen Replikat von diesem Peer entfernt werden soll.
2:  R_v = {...};                                  ▷ Liste der Replikate des Knotens v
3:  PAR_v = {...};                                ▷ Liste der PAR für v
4:
5:  function canResolveReplica(p)
6:      I(v_S) = S(P_v) / S_opt                   ▷ Übersättigung
7:      I(v_PI) = (PI_min − PI(v)) / (PI_min − PI_opt)   ▷ Leistungsindex
8:      I(v_t) = t_p(v) / t_min                   ▷ Fluktuation
9:      if min(v_S,v_PI,v_t) ≥ 1 ∧ PAR_v = ∅ then
10:         return true;
11:     end if
12:     return false;
13: end function
```

zur Vermeidung häufiger Schwankungen in der Replikatsnachbarschaft, für eine definierte Zeitspanne t_{min} in der Nachbarschaft verbleiben. Dies soll auch der Bildung lokaler Extrema im Graphen vorbeugen, die aufgrund in kürzester Zeit stark schwankender Laständerungen entstehen könnten. Dazu prüft der entsprechende Peer bei jeder Aktualisierung des Leistungsindex auf die Erfüllung der vorgenannten Kriterien. Liegen für eine Partition noch PAR vor, kann kein Replikat der Partition korrekt aufgelöst werden.

4.3 Fehlerbehandlung (crash)

Die Verbindung der Replikate untereinander in einer separaten Nachbarschaft ermöglicht einen schnellen Austausch von Leistungsinformationen und damit auch über Fehlfunktionen einzelner Peers. Durch die geöffneten Kommunikationswege zwischen den Peers können Verbindungsabbrüche effizient erkannt werden. Grundlegend sind vier Fehlerfälle zu unterscheiden (vgl. Abb. 4.5):



(a) Funktionsstörung einer Netzwerkkomponente führt zu Leistungseinbußen bei r_2

(b) Ausfall der Verbindung zwischen r_1 und r_3

(c) Ausfall des Replikats r_4

(d) Ausfall aller Replikate von v_i

Abbildung 4.5: Schematische Darstellung einer Replikatsnachbarschaft mit einem beteiligten Super-Peer und fünf Replikaten zur Visualisierung verschiedener Fehlerszenarien.

(a) Funktionsstörungen von Netzwerkkomponenten, die möglicherweise Einfluss auf die Übertragungsleistung in Form von Durchsatzeinbußen, Erhöhung der Latenz oder Paketverlust eines Peers haben, schlagen sich in seiner allgemeinen Leistungsbewertung nieder. Auf diese Weise wird ein Replikat automatisch mit weniger Anfragen belastet und im Falle einer ohnehin stark belasteten Nachbarschaft beginnt der verantwortliche Peer mit der Suche nach weiteren Replikatstandorten.

(b) Die Verbindung zwischen zwei Replikaten eines Knotens kann beendet werden, z.B. durch einen Ausfall von Netzwerkkomponenten, während die übrigen Replikate die Kommunikationskanäle zu allen anderen Peers aufrecht erhalten können.
Sei r ein Replikat der Nachbarschaft $R(v)$ dessen Kommunikation zu einem anderen Replikat r_x des gleichen Knotens v unterbrochen worden ist. Damit wird r_x aus der Replikatsnachbarschaft von r entfernt und keine weiteren Nachrichten von r an r_x weitergeleitet. Das Replikat r wartet nun eine definierte Zeitspanne, die u.a. von der Leistungsbewertung der beteiligten Peers abhängt, auf eine neue Replikatliste $R(v)$ des koordinierenden Replikats r_n. Erhält r diese Liste nicht, ohne das dabei die Verbindung zu r_n beeinträchtigt ist, wird r_n von r über den Ausfall von r_x informiert, da r_n offenbar keine Informationen über den Ausfall hat. Es ist nun an r_n zu entscheiden, wie mit r_x zu verfahren ist. Überschreitet die Anzahl an Ausfallmeldungen des Replikats r_x die vorgegebene Toleranz von 50% der existierenden Replikate, wird r_x aus der Nachbarschaft entfernt und die aktualisierte Nachbarschaftsliste unter den übrigen Replikaten verteilt. Anderfalls speichert r_n die Anzahl der Ausfallmeldungen zu r_x und kann sie bei Bedarf an seinen Nachfolger weitergeben.

(c) Ein Replikat oder die Verbindung zu ihm kann, z.B. durch einen Stromausfall oder eine Fehlfunktion, vollständig ausfallen. Damit werden die bestehenden Verbindungen zu den anderen Replikaten unterbrochen. Jedes Replikat entfernt daraufhin den ausgeschiedenen Peer aus seiner Replikatsnachbarschaft. Zuletzt übernimmt das jüngste Replikat der Nachbarschaft die Synchronisierung der Replikatsliste und sendet diese an die verbliebenen Replikate.

(d) Für den Fall, dass alle Replikate eines Knotens v ausfallen, können die Daten weiterhin vom Super-Peer angefordert werden. Ist auch dieser nicht länger

erreichbar, stehen die Daten von v nicht mehr im Netzwerk zur Verfügung. Kehrt ein Peer p ins Netzwerk zurück und besitzt noch Daten von v, so wird die Replikation der Daten, wie in Abschnitt 4.2.2 beschrieben, gestartet. Sind einzelne Daten eines Peers unvollständig oder ungültig, werden von anderen vorhandenen Replikaten nur diese Daten übertragen. Sind kein anderes Replikat und der entsprechende Super-Peer verfügbar, steht der Datensatz nicht vollständig zur Verfügung.

Die vorgenannten Fehler (a)–(c) lassen sich bereits durch systemimmanente Verfahren beheben. Einzig der Ausfall aller Replikate eines Knotens inkl. des Super-Peers ist vom System erst dann kompensierbar, wenn ein Replikat mit einer Kopie des fehlenden Knotens ins Netzwerk zurückkehrt.

4.4 Heuristisches Laden von Daten

Heuristische[26] Algorithmen erlauben das Lösen komplexer Fragestellungen mit angenäherten Ergebnissen und finden Anwendung, falls konventionelle Lösungsansätze keine oder nur sehr langsam eine exakte Lösung ergeben [95].

Die Anzahl und Vielfalt existierender Algorithmen zur Lösung unterschiedlicher Probleme ist nahezu unvorstellbar. Grundsätzlich lassen sich die Verfahren zur Lösungsfindung nach *Kokash* in unterschiedliche Gruppen einteilen:

- Exhaustive Search
- Local Search
- Divide and Conquer
- Branch and Bound
- Dynamic Programming

Die zu lösenden Probleme lassen sich in Klassen unterschiedlicher Komplexität einteilen [95]:

P Die Probleme dieser Klasse lassen sich in polynomieller Zeit von einer deterministischen Turing-Maschine[27] lösen.

NP Für Probleme dieser Klasse lassen sich Lösungen in polynomieller Zeit ausschließlich auf nicht-deterministischen Turing-Maschinen berechnen.

[26] aus dem altgriechischen für „Finden“ und „Entdecken“

[27] benannt nach Alan Turing (⋆23. Juni 1912, †7. Juni 1954), britischer Mathematiker und Logiker, Pionier und Gründervater künstlicher Intelligenz [54]

NP-vollständig Ein Problem ist NP vollständig, wenn sich für seine Lösung polynomielle Algorithmen auf nicht-deterministischen Turing-Maschinen entwickeln lassen, die gleichzeitig alle anderen Probleme in NP lösen.

NP-hart Diese Klasse enthält alle Probleme, die NP-vollständig oder härter sind. Dabei ist es möglich, dass keine Algorithmen zur Lösung dieser Probleme existieren und sie damit formal nicht zur Klassen NP gehören.

Heuristische Verfahren werden daher hauptsächlich zur Lösung von Problemen eingesetzt, die den Klassen NP-vollständig oder NP-hart zugeordnet sind. Mögliche Techniken für heuristische Verfahren sind z.B. Simulated Annealing [93], Reactive Search [15] und Evolutionäre Algorithmen [63].

Für den vorliegenden Anwendungsfall erlaubt die detaillierte Auswertung der von allen Peers angeforderten Daten aus einer Partition die Entwicklung lokaler Anforderungsmuster. Aus diesen Mustern lassen sich die Wahrscheinlichkeiten für kommende Übertragungsanforderungen ermitteln. Dazu kann aus der akkumulierten Häufigkeit der aufgezeichneten Anforderungen, gewichtet nach ihrer Reihenfolge, ein Wert für die Anforderungswahrscheinlichkeit einzelner Kacheln ermittelt werden. Mit Hilfe dieser Werte ist es möglich Daten, die noch nicht angefordert wurden, zusätzlich in den Cache zu laden, um die Beantwortung von eventuell eingehenden Anfragen zu beschleunigen.

Die Steigerung dieser Möglichkeit ist die Übertragung von Bilddaten anhand heuristischer Kennzahlen ohne das sie von einem Peer angefordert worden sind. Im Allgemeinen ist für den Einsatz dieser Techniken keine quantifizierbare Güte zu ermitteln. Die Last, die durch Übertragung nicht angeforderter Daten entsteht, muss sehr detailliert mit dem erwarteten Leistungszuwachs korreliert werden.

4.5 Anbindung mobiler Clients

Die Konstruktion des Overlays erfordert die Verwendung von zwei Pseudozufallszahlengeneratoren (vgl. Kap. 3.1). Durch die Verwendung von Java sollten diese PRNG plattformübergreifend nutzbar sein. Für mobile Endgeräte, die im Low-Cost-Bereich lediglich über eine ressourcenschwache Hardware verfügen, ist die Auslagerung ressourcenintensiver Prozesse sinnvoll.

Dieses Kapitel führt *Router* im Netzwerk ein, die der Vermittlung von Nachrichten an mobile Endgeräte (*Leecher*[28]) dienen. Gleichzeitig ist dieses Konzept

[28] angelehnt an das englische Wort „leech“ für Blutsauger oder Egel beschreibt Leecher ein

auch auf Endgeräte übertragbar, die keine Java-Unterstützung bieten und für die damit die Verwendung von PRNG einer anderen Plattform aufgrund einer potentiell unterschiedlichen Zufallszahlenkette keine Option ist [129].

Die verfügbaren Algorithmen unter Java wurden daher mit denen der plattformunabhängigen, auf C++ basierten Lösung Qt[29] [19] verglichen, die inzwischen auch als Framework für Android und iOS zur Verfügung steht [42]. Getestet wurden der PRNG der Klasse `java.util.Random`, die einen linearen Kongruenzgenerator (engl.: linear congruential generator, kurz LCG) verwendet [39], und ausgewählte Algorithmen [29] von C++, die laut Knuth [94] entsprechende Qualität besitzen:

- Linear congruential generator: `minstd_rand0`, `minstd_rand`
- Mersenne Twister: `mt19937`, `mt19937_64`
- Subtract with carry: `ranlux24_base`, `ranlux48_base`
- Discard block: `ranlux24`, `ranlux48`
- Shuffle order: `knuth_b`

Für den durchgeführten Test wurden alle PRNG mit dem gleichen Startwert (engl. *seed*) initialisiert und eine Kette von 100 Zufallszahlen generiert. Ein Vergleich der Ergebnisse deckt die Unterschiede zwischen der Java-Implementierung und den C++-Varianten auf, denn für keinen der verwendeten Algorithmen wird die gleiche Zahlenkette wie unter Java erzeugt. Standardmäßig vorhandene PRNG unter verschiedenen Compilern einzusetzen, ist aus diesem Grund nicht möglich.

Die Verwendung eines eigenen Algorithmus, der Zufallszahlen aus einer anderen Quelle bezieht, z.B. aus einer Hash-Funktion, ist eine mögliche Lösung. Prinzipiell sind zur Berechnung der Anzahl der Nachbarn die ID des Knotens v und die Anzahl an Knoten n im Netzwerk erforderlich. Werden diese beide Werte verkettet und an eine Hash-Funktion h übergeben, kann über Gleichung 4.1 eine zufällige Anzahl Nachbarn N_v berechnet werden.

$$h(\text{„ID“} + v + \text{„SIZE“} + n) \bmod n = N_v \tag{4.1}$$

Auch für die IDs I_{v_i} des i-ten Nachbarn von v lässt sich das Verfahren über eine Verkettung aller drei Parameter gemäß Gleichung 4.2 nutzen.

$$h(\text{„ID“} + v + \text{„SIZE“} + n + \text{„\#“} + i) \bmod n = I_{v_i} \tag{4.2}$$

Konzept, bei dem von vorhandenen Informationen profitiert wird ohne eigene Informationen anzubieten

[29] Webseite des Qt-Projekts: `http://www.qt.io`

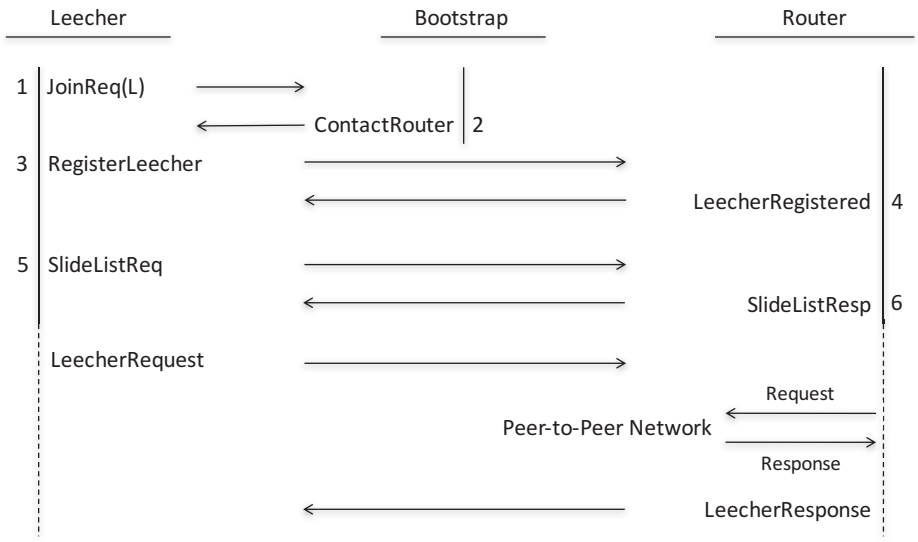


Abbildung 4.6: Schematische Darstellung des Leecher-Konzepts, das die Verwendung von Endgeräten mit stark limitierten Ressourcen oder ohne Java-Unterstützung erlaubt.

Die Verteilung der errechneten Nachbarschaftgrößen und IDs ist im Anschluss auf das erwartete Verteilungsmuster der Poisson-Verteilung abzubilden.

Für die Anbindung von Clients ohne Java-Unterstützung wurde letztendlich ein Konzept gewählt, dass auch für Endgeräte verwendet werden kann, die nur über sehr begrenzte Ressourcen verfügen. Dieses Konzept führt den neuen Knotentyp des *Leechers* ein, der lediglich Daten aus dem Netzwerk empfängt, jedoch selbst keine Daten zur Verfügung stellt. Leecher werden nicht wie gewöhnliche Peers im Netzwerk behandelt, stattdessen werden sie an normale Peers angedockt, die als Router für ihre Anfragen fungieren (vgl. Abb. 4.6).

Die Hosts, die als Router fungieren, markieren eine Kommunikation mit einem Leecher und leiten die Nachrichten als ihre eigenen weiter. Erhalten sie eine Antwort auf eine weitergeleitete Nachricht, wird diese an den entsprechenden Leecher übermittelt.

Das entwickelte Konzept ist zudem darauf ausgelegt auch ressourcenarme Endgeräte, z.B. Smartphones im Billigsektor, in das Netzwerk einbinden zu können. Diese müssen keine Routenberechnungen durchführen oder Daten an andere Knoten weiterleiten, was ihre ohnehin eingeschränkten Ressourcen zusätzlich entlastet.

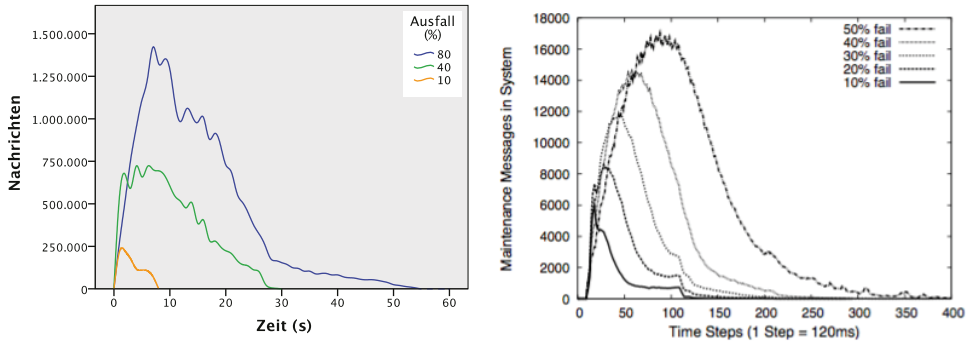
4.6 Evaluation

Hohe Fehlerraten sind in Peer-to-Peer-Netzwerken die Regel und keine Ausnahme [145]. Daher ist es erforderlich, dass alternative Pfade zur Lokalisation eines Knotens effizient berechnet werden können, falls sich herausstellt, dass ein Pfad durch den Ausfall aller Replikate eines Knotens nicht mehr verfügbar ist. Mit der Verwendung von Zufallsgraphen in Omentum ist die Anzahl alternativer und unabhängige Pfade zwischen zwei Knoten durch das Minimum ihrer Kapazität c definiert. Das Versagen eines Pfades ist einem Peer p zum Zeitpunkt der Routenplanung noch nicht bekannt und wird erst während der aktuellen Routingphase von einem Wegpunkt auf dem Pfad entdeckt. Durch die Verkettung der Replikate (vgl. Kap. 4.2) kann dieser Fall lediglich eintreten, wenn alle Replikate des Knotens ausgefallen sind. Andernfalls könnte der berechnete Pfad über ein anderes Replikat fortgesetzt werden.

Wird ein ausgefallener Peer erkannt, müssen im Gegensatz zu PathFinder keine Reparaturen an den Routing-Tabellen ausgeführt werden. Lediglich die lokale Nachbarschaft der Replikate muss für jeden ausgefallenen Peer repariert werden. Die dazu benötigten Nachrichten lassen sich über das in Kapitel 3.4 vorgestellte Verfahren ermitteln. Abhängig von der im System vorliegenden Last durch Anforderungen hat der gleichzeitige Ausfall vieler Knoten unterschiedlich starke Auswirkungen. So lässt sich in der Kommunikationsanalyse eines mittelgroßen Netzwerks ($N = 1.000.000$) ohne Berücksichtigung eines Lastprofils bei einem Ausfall von bis zu 80% der Replikate in 54,2 Sekunden ein fehlerfrei arbeitender Systemzustand erreichen. Dabei ist eine Verarbeitungskapazität in der Größenordnung von 10^4 Nachrichten pro Sekunde (vgl. Anhang A) auf jedem Peer vorausgesetzt und eine gleichverteilte Anordnung der Replikate sowie homogene Nachbarschaften der Ausgangspunkt. Die Reaktionsgeschwindigkeit der einzelnen Peers wurde über eine zufällige Zeitspanne zwischen 50 und 2.000 Millisekunden gestreut.

Das Diagramm in Abbildung 4.7(a) zeigt auf der x-Achse die vergangene Zeit seit dem unmittelbaren Knotenausfall und auf der y-Achse die Anzahl der Nachrichten, die zur Wiederherstellung eines fehlerfreien Zustands im System übertragen werden müssen.

Die durchschnittliche Last bleibt mit der nach Gleichung 3.24 berechneten Nachrichtenmenge von durchschnittlich 33 (8 + 8 + 16 + 1) Nachrichten lokal tolerierbar. Die Gesamtanzahl der Nachrichten für einen gleichzeitigen Ausfall von



(a) Omentum 1.000.000 Knoten (b) PathFinder 5.000 Knoten (aus [22])

Abbildung 4.7: Darstellung der Anzahl an Nachrichten bei einem simulierten Ausfall vieler Replikate bis zur Wiederherstellung eines fehlerfreien und konsistenten Gesamtsystems als Vergleich zwischen Omentum und PathFinder.

800.000 Knoten (80%) steigt zwar auf 1.422.118 Nachrichten an, liegt aber für die Erholung nach einem solch massiven Ausfall mit rund 63,7 MB Datenvolumen während der Leistungsspitzen im gesamten System noch in einem legitimen Rahmen. Verglichen mit einer Simulation von PathFinder und BubbleStorm [22, 163], die für das simulierte Netzwerk mit 5.000 Knoten zwischen 12 und 40 Sekunden für eine Wiederherstellung benötigen, liegt der Aufwand für Omentum durch die Verkettung der Replikate untereinander bedeutend niedriger. Unter PathFinder werden dabei ca. 18.000 Nachrichten im Peak versendet (vgl. Abb. 4.7(b)). Der Leistungsunterschied ist unter anderem in der Architektur des Zufallsgraphen begründet, da mit dem Wegfall eines Knotens in PathFinder auch die zugehörige Route entfernt werden muss, die über diesen Knoten führt. In Omentum verwalten mehrere Peers den gleichen Knoten. Aus diesem Grund lassen sich Ausfälle über andere Replikate aus der gleichen Nachbarschaft kompensieren. Lediglich die Koordinierung der einzelnen Replikate erfordert einen höheren Aufwand als bei PathFinder.

4.7 Verwandte Arbeiten

Neben der detaillierten Betrachtung von BubbleStorm und PathFinder, von dem das Omentum Overlay im Hinblick auf die Konstruktion und das Routing beeinflusst wurde, werden im Folgenden die Replikationsstrategien ausgewählter, strukturierter und unstrukturierter Overlays analysiert.

4.7.1 BubbleStorm & PathFinder

In den bereits vorgestellten Overlays von BubbleStorm und PathFinder (vgl. Kap. 3.1.2) werden Replikate auf verschiedene Arten erzeugt.

Das unstrukturierte Overlay BubbleStorm [162–164] geht davon aus, dass jeder Knoten Objekte erzeugen kann, die anschließend im Netzwerk repliziert werden müssen und nach Ausscheiden des Knotens wieder aus dem Overlay entfernt werden müssen [108]. Wird ein neues Objekt erstellt, wird es mittels eines *Push*-Algorithmus über einen *Random Walk* auf zufällige Knoten repliziert. Um keine fixe Anzahl von Replikaten aufrecht erhalten zu müssen, wird ein probabilistischer Ansatz gewählt, um die ungefähre Anzahl der Replikate während der gesamten Gültigkeitsdauer des Objektes (bis zur Löschung oder zum Knotenaustritt) beizubehalten.

Im Omentum Overlay werden Daten anhand ausgewählter Bewertungskriterien repliziert, wodurch die exakte Anzahl vorhandener Replikate jederzeit bestimmt und eine schnelle Anpassung der Replikatanzahl an die aktuelle Anforderungssitutation gewährleistet wird.

In PathFinder [23, 24] erhält ein Peer A seine virtuellen Knoten direkt beim Netzwerkbeitritt. Dazu wird A zunächst ein virtueller Knoten entsprechend des Hash-Werts seiner IP-Adresse zugewiesen. Sein Bootstrapknoten leitet den Beitrittswunsch anschließend zu einem Peer B weiter, der einen virtuellen Knoten passend zum Hash-Wert von A verwaltet. Anschließend werden A ein oder mehrere virtuelle Knoten von B zugeordnet und von B entfernt, wobei B mindestens einen virtuellen Knoten für sich behält. Verlässt A das Netzwerk regulär, übergibt er seine virtuellen Knoten an seine Nachbarn, die daraufhin für diese verantwortlich sind. Der Netzwerkbeitritt oder das Verlassen erfordern in PathFinder durchschnittlich $c + \frac{\ln N}{\ln c}$ Nachrichten, wobei c der Kapazität des Knotens und N der Knotenanzahl im Netzwerk entspricht.

Die für Omentum vorgeschlagene Architektur erfordert durch die Synchronisierung der Replikate einen höheren Nachrichtenaufwand bei einem Wechsel der Nachbarschaft als PathFinder. Im Gegenzug sind in Omentum direkt Leistungskennzahlen aller Knoten und die Größen ihrer Nachbarschaften auf jedem verantwortlichen Peer ablesbar, was die dynamische Replikationsstrategie optimal unterstützt.

4.7.2 Replikation

Diverse Replikationsstrategien wurden im Rahmen der Evolution erster Peer-to-Peer-Overlays entwickelt. Viele Verfahren zur Replikation in den strukturierten Overlays beschreiben die Erzeugung und Verteilung von Replikaten sehr detailliert, vernachlässigen jedoch die Aktualisierung und den Unterhalt der Replikate.

Gnutella

In Gnutella [1, S.62–79], einem der ersten unstrukturierten Overlays das komplexe Suchen unterstützt, werden alle Daten ausschließlich lokal gespeichert, was den Einsatz von Replikationsverfahren überflüssig macht. Erst die Einführung von Verwaltungsknoten (Super-Peers), die für eine kleine Gruppe von Peers verantwortlich sind, erforderte eine Replikation von Daten der Peers auf die Super-Peers. In Gia [36] wurde dieses Konzept auf die Replikation der Daten in die gesamte direkte Nachbarschaft eines Peers übertragen. Für Suchanfragen in den Overlays replizieren sowohl Gnutella, wie auch Gia, die entsprechenden *Query*-Objekte, wobei Gnutella die Suchreichweite anhand von Hops einschränkt und Gia die Länge der verwendeten *Random Walks* limitiert.

Chord / Pastry

Chord [153] repliziert Daten als temporäre Objekte auf eine Anzahl im Ring nachfolgender Knoten, überlässt jedoch sämtliche Aktualisierung und Unterhaltung der Replikate dem Anwendungsentwickler.
Ähnlich hierzu speichert Pastry [141] Replikate auf Knoten, die dem Key, der den Daten zugeordnet ist, am nächsten liegen. In diesem Overlay und seiner Erweiterung PAST [142] überwacht jeder Peer alle anderen Knoten, die für das gleiche Replikat verantwortlich sind. Auf diese Weise können durch die Operationen JOIN und LEAVE sowie durch Fehler verursachte Änderungen unter den Replikaten ausgetauscht werden. Zur Beschleunigung der Beantwortung von Anfragen speichert Pastry Daten bereits auf dem Suchpfad zwischen und versieht diese temporären Replikate mit einer festgelegten TTL.

Kademlia

Kademlia [118] repliziert Daten, ähnlich zu Pastry, ebenfalls auf Knoten, die dem zugeordneten Key am nächsten sind. Eine separate Caching-Strategie – ähnlich

zu der in Pastry – optimiert die Bearbeitung von Anfragen und soll eine lokale Überlastung von Peers reduzieren. Der für ein Replikat verantwortliche Peer kontaktiert stündlich alle weiteren verantwortlichen Peers. Das Ersetzen ausgefallener Peers ermöglicht so eine persistente Replikation von Objekten, die von der verwalteten Replikation, durch eine tägliche Erneuerung des ursprünglichen Besitzers der Daten, erweitert wird. Werden Daten eines Replikats angefragt, so wird eine Kopie mit einer limitierten Gültigkeitsdauer auf dem letzten Knoten des Routing-Pfades gespeichert, was einer temporären Replikation entspricht. Die Verwendung von drei unterschiedlichen Replikationsstrategien ist insgesamt betrachtet eine Verschlechterung. Neben der Komplexität, die auf diese Weise im System entsteht, bestimmt die schwächste Strategie die Leistungsfähigkeit des Systems. Erfolgt keine Erneuerung der Daten vom ursprünglichen Besitzer, werden die Replikate nach einem Tag entfernt. Obwohl durch die persistente und verwaltete Replikation prinzipiell Aktualisierungen der Replikate möglich wären, verhindert die zusätzliche temporäre Replikation durch das Caching entsprechende Mechanismen.

CAN / Tapestry

CAN [134] verwendet unterschiedliche Hash-Funktionen zur Ermittlung verantwortlicher Peers für einen gegebenen Key. Tapestry [176] erweitert den Ansatz zur Replikation von CAN durch zusätzliche Speicherung von Replikaten auf allen Knoten des Routing-Pfades, was bei CAN lediglich überlasteten Peers ermöglicht wird. Zusätzlich implementiert CAN, wie auch Pastry und Kademlia, einen Caching-Mechanismus um lokale Extrema zu vermeiden. Die periodische Erneuerung eines Replikats und die Verwendung von definierten Zeitintervallen für die Gültigkeit der replizierten Daten gestatten CAN und Tapestry eine Form der verwalteten Replikation.

Erweiterungen für konsistente Aktualisierungen

Die vorgestellten Replikationsverfahren DHT-basierter Overlays separieren die Caching-Strategie von der Replikationsstrategie. Diese mangelnde Koordination birgt im Hinblick auf Konsistenz ein Risiko für die fehlerfreie Aktualisierung der Daten, die auf diese Weise nicht gewährleistet werden kann. Eine Lösung wird von den jeweiligen Autoren selten vorgeschlagen oder diskutiert, was die Entwicklung von entsprechenden Erweiterungen für DHT-basierte Overlays erforderlich macht.

Akbarinia et al. versuchen mit dem *Update Managment System* [5] Konflikte während der Aktualisierung von Replikaten zu verhindern, indem eindeutige Zeitstempel (engl.: timestamp) verwendet werden, die über einen *Timestamp Service* bezogen werden. Dieser Service wird von dem verantwortlichen Peer für jeden Key der DHT zur Verfügung gestellt, der die Updates serialisiert [5]. Dieses Verfahren entspricht einer verwalteten Replikation mit einem veränderlichen Besitzer.

Für die P-Ring DHT [44] schlagen *Antony et al.* eine Replikations- und Updatestrategie vor, die Replikate an eine Kette von Nachfolgern weiter gibt, ähnlich zu Chord. Replikate werden dabei nur propagiert, wenn ein Peer das Update auch angenommen hat. Eine Bestätigung des letzten Peers der Kette in umgekehrter Reihenfolge lässt jeden Knoten das Update auf den lokal gespeicherten Daten ausführen [44]. Dieser Ansatz entspricht ebenfalls einer verwalteten Replikation mit einem veränderlichen Besitzer.

Aufgrund der Abhängigkeit von einer korrekt arbeitenden und konsistenten DHT, auf der die genannten Verfahren basieren, können durch eine häufig auftretende Inkonsistenz in den Routing-Tabellen [67] unvorhersehbare Ergebnisse produziert werden, die keinen erwartungsgemäßen Betrieb mehr erlauben.

Aberer et al. stellen mit dem P-Grid Overlay [2] eine epidemische Aktualisierungsstrategie vor, die Updates rekursiv auf verantwortlichen Knoten speichert (*push*) und neue oder zurückkehrende Knoten veranlasst möglicherweise verpasste Updates anzufragen (*pull*) [46]. Simultane Aktualisierungen werden von diesem Ansatz nicht betrachtet.

Für unstrukturierte Overlays schlagen *Wang et al.* mit UPTReC [168] ein Verfahren vor, das Updates über eine Replikatskette verteilt. Um die Kette bei einem gleichzeitigen Ausfall vieler Knoten nicht zu unterbrechen, speichern alle Peers eine Liste mit mehreren Vorgängern und Nachfolgern. Da Updates entlang der Kette in beiden Richtungen erfolgen, wird ein Timestamp-basierter Ansatz zur Erkennung simultaner Aktualisierungen verwendet.

4.8 Zusammenfassung

In diesem Kapitel wurde eine neuartige, speziell für Omentum entwickelte Replikationsstrategie vorgestellt. Zur Umsetzung wird eine Replikatsnachbarschaft etabliert, die aus Peers besteht, die eine Kopie des jeweiligen virtuellen Knotens verwalten. Diese Replikatsnachbarschaft ist vollständig selbst organisierend. Das Erstellen und Auflösen von Replikaten erfolgt synchronisiert durch das jüngste

Replikat einer Nachbarschaft, wodurch die zusätzliche Last nicht immer vom gleichen Peer getragen werden muss. Im Vergleich zu Gnutella, BubbleStorm oder P-Ring wird damit das Problem der Aktualisierung gelöst ohne auf *Flooding* oder rekursive Verfahren zurückzugreifen.

Über PriorityAssistanceRequests (PAR) ist das jüngste Replikat der Nachbarschaft in der Lage bei längerfristiger Unterschreitung einer Leistungsgrenze die Unterstützung durch neue Replikate anzufordern. Diese aktive Unterstützungsstrategie ist neuartig und wird in dieser Form von keinem Overlay bisher unterstützt. Sie bietet neben den umgesetzten reaktiven Strategien eine gezielte Steuerung, die aufgrund ihrer begrenzten Replikation den lokalen Kontext der einzelnen Peers durchbrechen kann und damit lokalen Extrema entgegenwirkt.

Das konzipierte Nachrichtensystem, das von allen Peers im System zur Kommunikation verwendet wird, arbeitet mit einer lokalen Rückwärtsfehlerkorrektur, die einen Knoten nach einem Fehlerfall in einen konsistenten Zustand überführen kann. Zur Abstimmung mit den übrigen Knoten wird eine knotenspezifische Nachrichtenliste gepflegt, die es erlaubt die letzte Kommunikation zwischen zwei Knoten vollständig zu rekonstruieren. Vor allem bei kurzen Ausfällen oder Kommunikationsfehlern führt die lokale Wiederherstellung eines gültigen Zustands zu einer Ersparnis an übertragenen Nachrichten. Die eingeführten Checkpoints finden lediglich für wichtige Systemnachrichten Verwendung, die bei einer möglichen Wiederherstellung relevant sind.

Für den Transfer von möglicherweise ebenfalls zu übertragenen Daten wurden, in Abhängigkeit von den bereits angeforderten Daten, heuristische Verfahren diskutiert, um die Zeit für die Anforderung neuer Daten zu minimieren. Die erhobenen Anforderungsmuster können von jedem Replikat eines Knotens lokal ausgewertet werden, so dass eine systemweite Abstimmung nicht erforderlich ist.

Die Anbindung ressourcenarmer, mobiler Knoten wird über ein neu eingeführtes Routing-Konzept realisiert, das es erlaubt mobile Geräte (*Leecher*) an einen bereits verbundenen Peer (*Router*) anzuhängen. Auf diese Weise können Leecher die Routen-Berechnungen auf den Router auslagern und müssen diese nicht selbst ausführen. Ein weiterer Vorteil ist die Möglichkeit auch Geräte einzubinden, die nicht Java-fähig sind und damit keinen Zugriff auf die Routenberechnung über die in Kapitel 3.1 beschriebenen PRNG haben.

Die Ergebnisse von Simulationen zeigen für ein System mit 10^6 Knoten eine sehr gute Widerstandsfähigkeit gegen Knotenausfälle ohne Kompensationsbedarf für ungültige Routen. Die verwendete Replikationsstrategie übertrifft im Hinblick

auf Ausfallsicherheit und Lastverteilung vorgestellte Strategien aus strukturierten und unstrukturierten Overlays. Besonders sind hier das aktive Lastmanagment der Knoten über PriorityAssistanceRequests und die Routenkonstanz auch bei Ausfall vieler Replikate zu nennen.

KAPITEL 5

USPFS – ein portables Dateisystem zur optimierten Verwaltung sehr vieler kleiner Dateien

Das Userspace File System (USPFS) ist ein abstrakter Entwurf eines hierarchischen Dateisystems und ist mit dem Fokus auf Datenintegrität, Portabilität und einer großen Menge zu speichernder Objekte konzipiert worden [125]. Der in Kapitel 2 vorgestellte Anwendungsfall eines verteilten virtuellen Mikroskops stellt aufgrund der großen Datenmengen hohe Ansprüche an eine optimale Lastverteilung. Vor allem die Aufbereitung der Daten zu kleineren Paketen, die leichter verteilbar sind, führt zu einer sehr hohen Anzahl an Dateien, die zu einem vollständigen Datensatz gehören. Unter einigen Dateisystemen ist die Menge an speicherbaren Objekten beschränkt (z.B. NTFS) oder durch die Größe des Volumes definiert (z.B. EXT3).

Die Verteilung der Dateigrößen innerhalb des gesamten Systems und eines Datensatzes im Speziellen (vgl. Kap. 2.2) lässt erkennen, dass durch die gewählte, pyramidale Aufbereitung der Bilder die Menge kleiner Dateien eine optimierte Handhabung im Dateisystem erforderlich macht. Dies beeinflusst sowohl die Zeit zum Auffinden einzelner Dateien, wie auch die Speicherverwaltung im Hinblick auf Blockgröße und Metadaten. Hier muss eine geeignete Größe der Speicherblöcke gewählt werden, damit die hohe Anzahl sehr kleiner Dateien nicht zu einer starken Fragmentierung führt und gleichzeitig nicht zu viele Metadaten für große Dateien gespeichert werden müssen.

Die Verfügbarkeit von Integritätsprüfungen aller Dateien ist ebenfalls von zentraler Bedeutung. Eine versehentliche oder auch vorsätzliche Manipulation der Daten ohne Integritätsprüfung könnte womöglich zur Replikation fehlerhafter

Daten führen. Das Überschreiben korrekter Daten durch fehlerhafte Duplikate wäre schlimmstenfalls die Konsequenz.

Dass ein vom Anwender eingesetztes Dateisystem alle oben skizzierten Anforderungen erfüllt, kann aufgrund der heterogenen Vielfalt verfügbarer Dateisysteme unter diversen Betriebssystemen nicht sichergestellt werden. Weiterhin kann von einem Anwender nicht verlangt werden, speziell für die Verwendung einer Applikation sein bevorzugtes Dateisystem zu ersetzen. Selbst wenn dies hingenommen würde, ist noch nicht sichergestellt, dass ein von der Anwendung gefordertes Dateisystem auch auf den meisten Plattformen verfügbar ist und dort die gleiche native Unterstützung mit allen Funktionen bietet.

Die gesuchte Lösung muss also von einer breiten Masse verfügbarer Desktop-Betriebssysteme unterstützt werden können. Optimalerweise liegt sie im Userspace, um notwendige Eingriffe in den Kern des Betriebssystems zu vermeiden.

Die skizzierten Anforderungen werden mit einem innovativen, plattformunabhängigen und anpassungsfähigen Dateisystem adressiert. Seine nachfolgend erläuterte Architektur ist in der Lage, sehr große Datenmengen mit einer theoretisch unbegrenzten Menge an Objekten und geringem Metadaten-Overhead zu verwalten.

Üblicherweise verwenden Dateisysteme eine designierte Partition, die exklusiv auf einem beliebigen physikalischen Datenträger bereitgestellt wird. Dies führt zu der offensichtlichen Einschränkung, dass ein Dateisystem nicht mehr Blöcke adressieren kann, als von seinem physikalischen Speichermedium zur Verfügung gestellt werden. Einige Dateisysteme greifen hier zu einem *VolumeManager*, der es erlaubt verschiedene Speicherbereiche und -medien zu aggregieren und als einen zusammenhängenden Speicherbereich darzustellen [21]. In gleicher Weise soll auch USPFS nicht von dieser Einschränkung betroffen sein. Zur Unterstützung sind demnach multiple Speicherbereiche und eine zusätzliche Verwaltungsebene erforderlich.

Die Portierbarkeit auf die meisten Desktopbetriebssysteme wird durch die Speicherung von USPFS-relevanten Daten in den bereits vorhandenen Dateisystemen eines Rechners gelöst, wie u.a. bei HDFS [150].

Da durch den Anwendungskontext in Omentum das Hauptaugenmerk von USPFS auf der Speicherung sehr vieler kleiner Dateien liegt, wird eine effiziente zweistufige Adressierung der Daten gewählt. Dazu wird der Speicherbereich von USPFS in mehrere Segmente (Slices) unterteilt. Auf diese Weise kann zunächst das passende Slice adressiert werden, bevor in einem zweiten Schritt die byte-

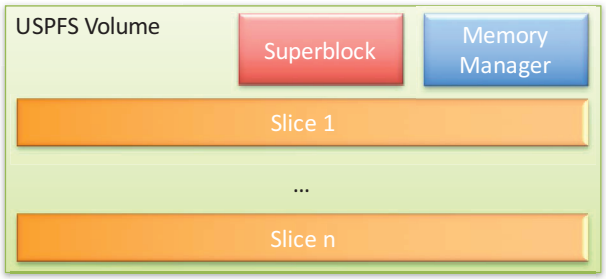


Abbildung 5.1: Aufbau eines Volumes in USPFS.

weise Adressierung eines Objektes innerhalb des gewählten Slices erfolgt. Beide Adressierungen werden intern über Referenzen mit 64 Bit geregelt. Dies begrenzt die Größe eines einzelnen Slices auf 2^{63} Bytes, was etwa 9.000 Petabyte entspricht, eröffnet aber durch die ebenfalls 2^{63} ($\approx 9 \cdot 10^{18}$) adressierbaren Slices einen Speicherbereich von ungefähr $8 \cdot 10^{37}$ Bytes, der für Omentum wie auch für viele andere Applikationen ausreichend ist.

Im Anwendungskontext von Omentum ist die Anzahl an Dateien, die in einem einzelnen Verzeichnis gespeichert werden können, ebenfalls von besonderer Bedeutung. Da hier der Fokus auf kleinen Dateien liegt, wird eine effiziente Metadatenverwaltung und eine Index-freie Organisation verwendet, um die Anzahl an speicherbaren Objekten nicht einzuschränken.

5.1 Architektur

Eine einzelne USPFS-Instanz besteht aus einem *Superblock*, einer Speicherverwaltung und den jeweiligen Slices, die letztlich die gespeicherten Objekte enthalten (s. Abb. 5.1). Durch die Unterteilung in Slices entsteht der weitere Vorteil, dass Benutzer auch hier unterschiedliche Partitionen, Datenträger oder Netzlaufwerke zu einem Laufwerk aggregieren können.

Der Superblock identifiziert die Instanz des Dateisystems. Hier sind Basis-Informationen zu USPFS (Versionsnummer, Laufwerksname, Blockgröße) und Daten über die Speicherverwaltung sowie eventuell aktivierte Filter gespeichert. Weiterhin können hier Slice-Indices und die maximale Größe einzelner Slices ausgelesen werden, sofern eine entsprechende Begrenzung konfiguriert ist. Für den Superblock wird bei schreibenden Zugriffen seiner Inhalte, z.B. bei Änderungen der verfügbaren Slices, eine aktuelle Kopie gespeichert. Aufgrund der geringen Größe wird so eine hohe Redundanz erreicht, die zu einer hohen Fehlertoleranz

im Falle eines beschädigten Superblocks führt, da ein nicht länger lesbarer Block durch eine aktuelle Kopie ersetzt werden kann.

Die separate *Speicherverwaltung* organisiert die freien und bereits allozierten Speicherblöcke und löst feste Strukturen für diese Aufgabe innerhalb des Laufwerks ab, um eine Beschränkung wie u.a. in der Familie der EXT-Dateisysteme [34] zu vermeiden. Die Speicherverwaltung ermöglicht abhängig vom jeweiligen Anwendungskontext eine optimierte Anpassung für spezifische Fragestellungen und kann bei der Initialisierung des Dateisystems entsprechend festgelegt werden. Dies führt zu einer Optimierung der Leistung des Dateisystems und des Speicherbedarfs der Speicherverwaltung.

Ein I/O-System, das sich um den Zugriff auf die eigentlichen Datenträger und Verzeichnisse des Hosts kümmert, gestattet das Lesen und Schreiben in USPFS. Durch einen modularen Aufbau in einer flexiblen API ist es Nutzern möglich, je nach Anwendungsfall entsprechende Module zu implementieren, die neben der Standardquelle eines lokalen Dateisystems z.B. die Verwendung von Netzwerkdateisystemen oder Sockets als Quellen ermöglichen. Weiterhin können die gelesenen und geschriebenen Daten durch eine anpassbare Filterkette geleitet werden, die ebenfalls bei der Laufwerksinitialisierung festgelegt wird und z.B. durch ähnliche Funktionen in FUSE [158] bekannt ist.

Dabei beschreibt ein *Filter* zwei komplementäre Funktionen, die sowohl die gelesenen als auch die geschrieben Daten modifizieren. Mehrere Filter können miteinander verkettet werden, wobei die Reihenfolge der Bearbeitung dem „Last In – First Out“-Prinzip folgt. Typische Funktionen für Filter sind u.a. Kompression oder Verschlüsselung.

Bei aktivierter Integritätsprüfung wird für jede Datei über einen Algorithmus, der ebenfalls zum Zeitpunkt der Laufwerksinitialisierung festgelegt wird, eine Prüfsumme in ihren Metadaten gespeichert. Diese wird anschließend bei jedem Zugriff auf die Datei validiert.

5.1.1 Organisation der Metadaten

Alle Daten sind entsprechend der verwendeten Speicherverwaltung in Blöcken organisiert, deren Größe (üblicherweise ein Vielfaches von 512 Bytes) im Superblock festgelegt ist. Auf diese Weise wird die Speicherverwaltung vereinfacht und effizienter. Eine *Inode* indiziert jede Datei (oder jeden Anteil davon) und speichert alle benötigten Informationen. Dazu gehört wie bei POSIX u.a. der Dateityp (Datei, Verzeichnis, symLink), die Zugriffsrechte, benutzerdefinierte Informationen, Da-

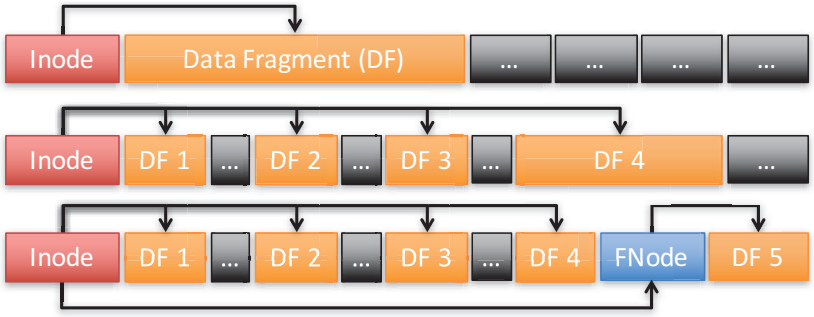


Abbildung 5.2: Vereinfachtes Beispiel für drei mögliche Layouts von Metadaten einzelner Dateien innerhalb eines Slices.

tum und Uhrzeit der letzten Änderung und des letzten Zugriffs, Informationen zu Besitzer und Gruppe, der Dateiname und die -größe sowie die Prüfsumme. Die Inode enthält außerdem Verweise auf bis zu vier Fragmente für einen schnelleren Dateizugriff (s. Abb. 5.2), wie es z.B. von extents aus EXT4 [117] bekannt ist.

Ein *Fragment* beschreibt einen Speicherblock im Dateisystem mit einer Länge, die einem ganzzahligen Vielfachen der im Superblock definierten Blockgröße entspricht. Das Design der Speicherverwaltung ermöglicht es Daten, unabhängig von ihrer Länge, in einem einzigen Fragment während einer einzelnen Schreiboperation zu speichern. Dies führt zu kontinuierlichen Lese- und Schreiboperationen, was wiederum die Geschwindigkeit aufgrund fehlender Sprünge erhöht. Bei stärkerer Fragmentierung einer Datei, referenziert die Inode eine zusätzliche Fnode.

Jede *Fnode* enthält weitere Verweise auf zusätzliche Fragmente, was sie zu indirekten Referenzen macht. Mehrere Fnodes bilden eine doppelt verkettete Liste, deren erstes Element von der Inode referenziert wird. In dieser Organisation ist die maximale Anzahl an Fragmenten für eine einzelne Datei theoretisch unbegrenzt, was zu einer maximalen Dateigröße (Metadaten eingeschlossen) führt, die nahezu der Laufwerksgröße entspricht.

5.2 Implementierung - JUSPFS

Für die Umsetzung des Konzeptes wurde Java als Programmiersprache gewählt (Java-USPFS, kurz JUSPFS). Hierdurch wird ein Kompilieren für unterschiedliche Plattformen umgangen und die Portierbarkeit erhöht. Die Integration in das ebenfalls in Java entwickelte Omentum Overlay wird zusätzlich erleichtert. Nichtsdestotrotz ist es möglich, auch jede andere Programmiersprache für eine

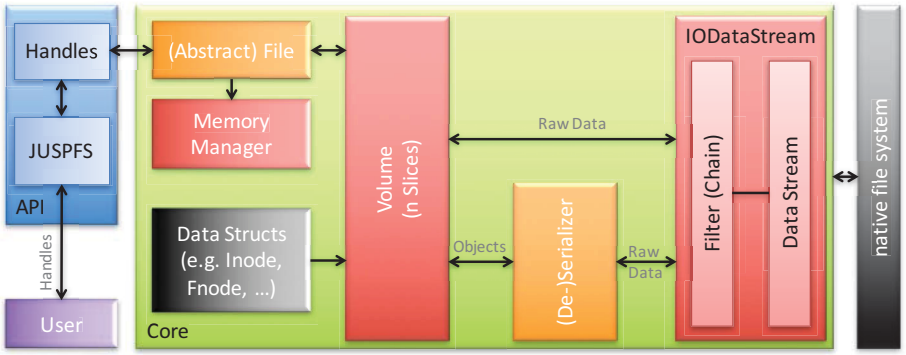


Abbildung 5.3: Struktur und Kommunikationspfade der Kernkomponenten von JUSPFS.

Umsetzung von USPFS zu verwenden, um damit z.B. auch Plattformen zu erreichen, die Java nicht unterstützen.

Die Implementierung verwendet ausschließlich Klassen aus den nativen Paketen *java.io*, *java.nio* und *java.util* ohne weitere externe Abhängigkeiten. Der Aufbau und das Zusammenspiel der einzelnen Komponenten ist in Abbildung 5.3 dargestellt und wird im Folgenden für ausgewählte Komponenten detailliert erläutert.

Das Interface *IODataStream* muss für jede Quelle, die für einen Datenzugriff verwendet werden soll, entsprechend implementiert werden. Die Klasse *IODataStreamFile* verwendet für den Zugriff auf Dateien die Java-Klasse *RandomAccessFile*, die ein gezieltes Aufsuchen einer bestimmten Position unterstützt. *IODataStream* kann ebenfalls anderweitig implementiert werden um weitere Quellen zu unterstützen, wie z.B. Netzwerk-Sockets oder andere nicht Datei-ähnliche Speicherorte.

Jede Organisationseinheit der Metadaten (Superblock, Inode, Fnode) wird durch ihre eigene Klasse repräsentiert. Diese Klassen müssen jeweils für Lese- und Schreibzugriff über *IODataStream* (de-)serialisiert werden können.

Die zentrale Komponente für den größten Teil der Logik in JUSPFS ist die Klasse *File*. Hier wird sowohl das Management der Metadaten einzelner Dateien, Interaktion mit der Speicherverwaltung zum Freigeben und Allozieren einzelner Blöcke geregelt, wie auch das Lesen und Schreiben von Rohdaten. Die automatische Erweiterung der Metadaten bei steigender Fragmentierung einer Datei wird hier ebenfalls adressiert. Da USPFS als hierarchisches Dateisystem konzipiert ist, implementieren *FileHandle* und *DirectoryHandle* die Entsprechungen für Datei

und Verzeichnis. Letzteres verwendet eine *HashMap* zur Verwaltung von Elementen und erlaubt damit das Speichern von 2^{31} Einträgen in jedem Verzeichnis.

Für die Nutzung stehen bislang Handles für die Arbeit mit Dateien und Verzeichnissen zur Verfügung. Die modulare und leicht erweiterbare API lässt es zu, dass Nutzer auch neue Handles, z.B. für Links, implementieren, die für einen speziellen Anwendungsfall angepasst sind. Damit sind auch nicht hierarchisch organisierte Dateisysteme mit diesem Konzept umsetzbar, da Handles u.a. auch direkt für Objekte verwendet werden können.

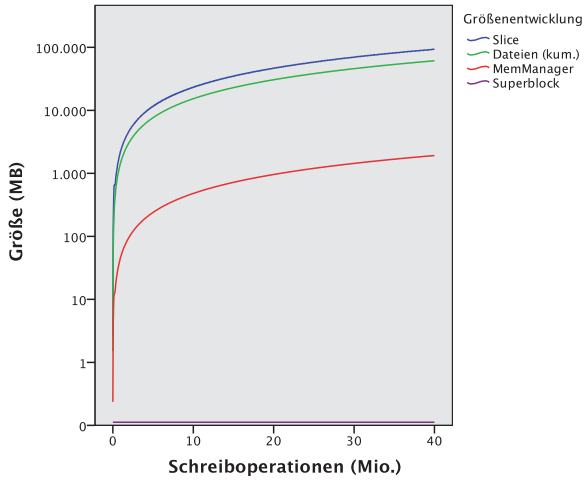
5.3 Evaluation

Für Dateisysteme ist es nicht nur wichtig, die Schreib- und Lesegeschwindigkeit zu ermitteln, sondern auch den Overhead für die Verwaltung der Metadaten aller gespeicherten Elemente. Letzterem kommt in Omentum eine besondere Bedeutung zu, da mit sehr vielen kleinen Dateien gearbeitet wird und eine effiziente Verwaltung der Metadaten neben einer möglichst minimalen Fragmentierung zur Reduktion des Speicherbedarfs unumgänglich ist.

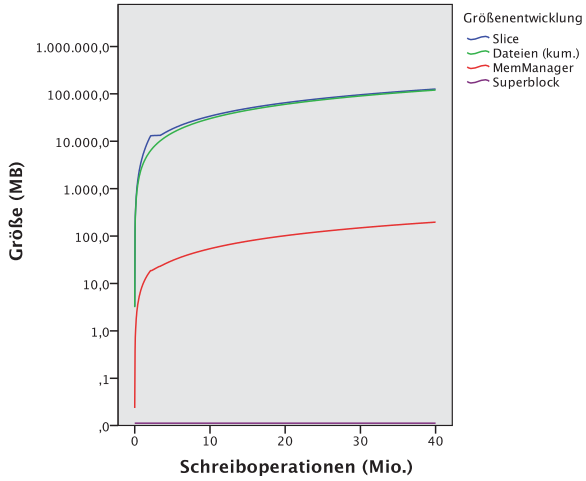
5.3.1 Metadaten

Typische Datensätze aus Omentum bilden die Grundlage für die Evaluation des Dateisystems JUSPFS (vgl. Kap. 2.2). Verglichen mit der Menge an gespeicherten Informationen fallen die Metadaten kaum ins Gewicht. Der in Abbildung 2.4(a) exemplarisch dargestellte Datensatz enthält 98.170 Dateien mit insgesamt 3.111.791.334 Bytes an Nutzdaten. Die durchschnittliche Dateigröße liegt bei 31.698 Bytes, wobei 289 Bytes das Minimum und 86.751 Bytes das Maximum markieren. Für diesen Datensatz fallen Metadaten im Umfang von etwa 7 MB Volumen an. Zusätzlich belaufen sich die Metadaten der Speicherverwaltung für das gesamte Volume auf 4 MB. Die insgesamt 11 MB ergeben ein Verhältnis von Metadaten zu Nutzdaten von sehr guten 0,3%.

Der in Abbildung 5.4 dargestellte Vergleich der kumulierten Dateigröße im nativen Dateisystem und den zugehörigen Metadaten in JUSPFS für einen Test mit 40 Millionen Schreiboperationen von Dateien mit unterschiedlicher Größe zeigt, dass für ein Slice ein entsprechender Anteil an Metadaten erforderlich ist. Dieser Anteil kann aus der Differenz der nativen Dateigröße und der Größe des Slice berechnet werden.



(a) Metadaten für kleine Dateien (1.600 Byte)



(b) Metadaten für größere Dateien (31.000 Byte)

Abbildung 5.4: Vergleich des Speicherverbrauchs zur Analyse des Verhältnisses zwischen Metadaten (Slice, MemoryManager sowie Superblock) und Nutzdaten (kum. Dateigröße) in Abhängigkeit vom Füllungsgrad des Dateisystems für Dateien unterschiedlicher Größe.

Der Speicherverbrauch steigt entsprechend der Größe der gespeicherten Dateien linear, wobei die gewählte Blockgröße von 512 Byte einen kontinuierlichen Speicherverlust von 20 bis 25% für Dateien mit einer Größe zwischen 1580 und 1612 Byte bedeutet (vgl. Abb. 5.4(a)). Bei Dateien, deren Größe dem ermittelten

Durchschnitt von rund 30 KB entspricht, ist der Verbrauch an Platz für Metadaten signifikant niedriger (vgl. Abb. 5.4(b)).

5.3.2 Lesen und Schreiben

Die Schreibleistung wurde auf Konstanz bei steigender Auslastung des Volumes evaluiert. Dazu wurden alle Tests auf einem neu installierten System[30] vorgenommen, das neben dem Betriebssystem nur das Testprogramm ausführt. Die in Abbildung 5.5 zu sehenden, äquidistanten Peaks, scheinen mit einem spezifischen Taskscheduling des jeweiligen Betriebssystems zusammen zu hängen. Sie können unter allen Betriebssystemen beobachtet werden, unabhängig vom jeweils verwendeten Dateisystem, wobei der Abstand je nach Kombination variieren kann.

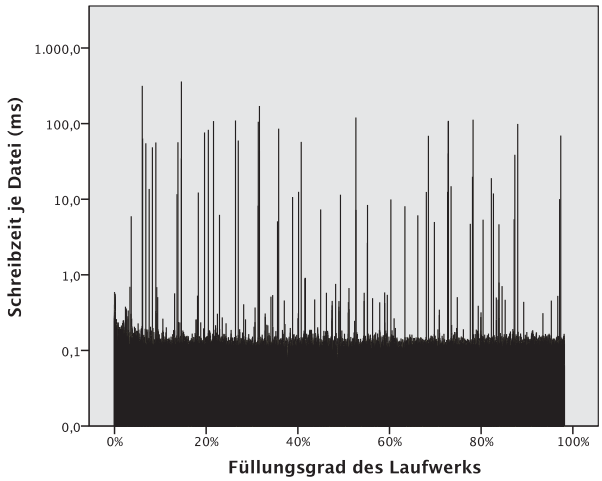


Abbildung 5.5: Schreibleistung von JUSPFS in Abhängigkeit vom Füllungsgrad des Volumes.

Um eine konstante Leseleistung sicherzustellen, wurde eine zufällige Auswahl von Dateien aus einem im Vorfeld gefüllten Dateisystem gelesen. Dabei variiert die Größe der gelesenen Dateien wie auch ihre Position im Dateisystem. Zusätzlich wurde ein Experiment mit deaktiviertem Cache durchgeführt, um ein Worst-Case-Szenario zu erhalten.

Die Geschwindigkeitsmessungen wurden auf einem Testsystem mit SSD [31] durchgeführt. Für den Metadatenvergleich wurde aufgrund der Größe der ge-

[30] iMac 27“, 2,7 GHz Intel Core i5 CPU, 16 GB DDR RAM und 1 TB SATA HDD

[31] MacBook Pro 15„ 2,5 GHz Intel Core i7 CPU, 16 GB DDR RAM, 512 GB SSD (Apple SSD SM0512G)

schriebenen Daten eine externe SATA Festplatte mit 3TB (Western Digital My Book) verwendet. Die nativen Geschwindigkeiten der SSD liegen unter Mac OS X bei 1794 MB/s lesend und 713 MB/s schreibend[32].

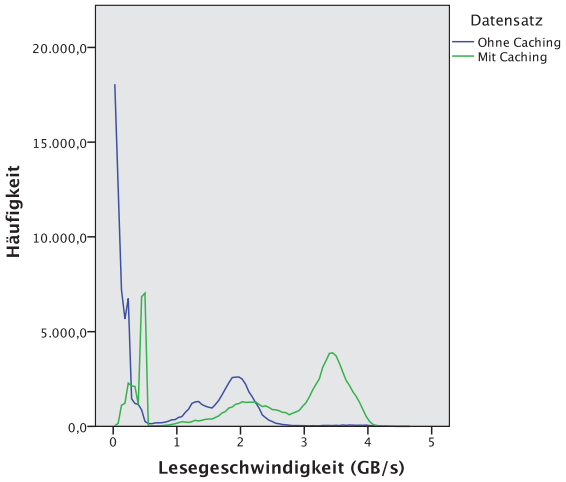


Abbildung 5.6: Leseleistung von JUSPFS in Abhängigkeit vom Füllungsgrad des Volumes und Caching.

Bei konventionellen Zugriffsmustern in Omentum liegt die Lesegeschwindigkeit aufgrund von wiederholten Zugriffen auf die gleiche Datei im Mittel bei 953 MB/s. Nach Deaktivierung des Caches im Betriebssystem, sinkt die Leseleistung auf durchschnittlich 15,41 MB/s. Die maximale Datenrate, mit der ein Knoten von seinen Nachbarn belastet wird, liegt bei etwa 6 MB/s.

Die Leistungsfähigkeit von JUSPFS liegt aufgrund der prototypischen Implementierung in der Einschränkung auf einen Single-Thread-Betrieb deutlich unter der Leistung des nativen Zugriffs auf das Speichermedium. Bei entsprechender Anpassung sind Leseraten um 1.300 MB/s und Schreibraten um 400 MB/s zu erwarten. Die übrige Leistung wird für die Verkettung der JVM mit den nativen Bibliotheken aufgewendet.

Weiterhin hat das Caching einen starken Einfluss auf die Ergebnisse der Schreib- und Leseleistung, was die Frage nach einer Vergleichbarkeit der Leistung bei unterschiedlichen Betriebs- und Dateisystemkombinationen aufwirft.

Drei Tests mit unterschiedlichen Betriebssystemen und ihrem plattformtypischen Dateisystem zeigten voneinander abweichende Ergebnisse ohne signifikant

[32] Werte ermittelt mit AJA System Test v2.1 (Webseite: `https://www.aja.com/de/products/aja-system-test`).

Tabelle 5.1: Leistung von unterschiedlichen Betriebs- und Dateisystemkombinationen bei Einsatz von JUSPFS.

Betriebssystem	**Dateisystem**	**spezifisch**	**FAT32**
Ubuntu 13.04 (64bit)	ext4	1,00	0,97
MacOS 10.7.5 (64bit)	Mac OS Extended (Journaled)	0,88	0,84
Microsoft Windows 7 (64bit)	NTFS	0,54	0,76

deutbare Einzelparameter. Daher wurde eine weitere Testreihe mit unterschiedlichen Betriebssystemen und FAT32 als konstantem Dateisystem durchgeführt. Die Wahl von FAT32 wurde aufgrund der Verfügbarkeit unter allen getesteten Betriebssystemen motiviert, da so auch unterschiedliche Betriebssysteme auf dem gleichen Dateisystem mit JUSPFS getestet werden konnten. Für keine Kombinationen wurden Optimierungen des jeweiligen Dateisystems durch Manipulation möglicher Konfigurationsparameter vorgenommen. Lediglich die Verwendung der gleichen Blockgröße wurde sichergestellt.

Eine Übersicht über die erzielten Leistungen, normalisiert auf den höchsten Wert, stellt Tabelle 5.1 dar.

Die besten Ergebnisse wurden unter Linux mit EXT4 erzielt, während sich die Kombination Windows mit NTFS als Ungünstigste erwies. Ein Erklärungsansatz lässt sich in der Umsetzung der Dateisystem-Schnittstelle der JVM finden, die für diese Operationen unterschiedlich optimierten Bytecode liefert. Ferner gibt es Unterschiede in der Implementierung der einzelnen Dateisysteme unter den diversen Betriebssystemen. Auch die unterschiedliche Ausstattung der Dateisysteme mit standardmäßig aktivierten Optionen, lässt den Schluss zu, dass die Leistung von mehreren Faktoren unterschiedlich stark beeinflusst wird.

Die Messungen zeigen, dass aktuell der Einsatz von JUSPFS unter Linux vorzuziehen ist, da diese Kombination, abhängig vom nativen Dateisystem einen deutlichen Leistungvorteil von $12 - 14\%$ bietet.

5.4 Verwandte Arbeiten

Der Ansatz verteilter Dateisysteme, wie sie bereits seit den 80er Jahren z.B. durch ITC [146]) bekannt sind, ist in den letzten Jahren vor allem für die Arbeit mit sehr großen Datenmengen (Big Data) bedeutend geworden. Als grundsätzliche Vereinbarung der meist hierarchischen Umsetzungen sind alle Dateien über ei-

ne Kombination aus Name und Pfad, ausgehend von der Wurzel des jeweiligen Namensraums, eindeutig identifizierbar.

Das erklärte Ziel verteilter Dateisysteme ist eine für den Nutzer transparente Einbindung in das jeweils verwendete Betriebssystem.

Die meisten dieser Konzepte, die z.B. durch Protokolle wie NFS [143] beschrieben werden, basieren auf einer Client-Server-Architektur und nutzen die Ressourcen des Clients zur Datenspeicherung. Für reine Peer-to-Peer-Systeme sind diese Ansätze daher nicht ohne Weiteres verwendbar.

Konzepte wie das Google File System GFS [66] fokussieren vor allem „Multi-GB“ Dateien, die sich in der Regel nicht oder nur höchst selten ändern. Eine freie Implementierung von GFS ist das in Java implementierte Hadoop Distributed File System [150] der Apache Foundation.

Ein Ansatz wie dieser ist prinzipiell auch für Omentum denkbar, allerdings müssten hierfür alle Daten zur Laufzeit von den jeweils verantwortlichen Clients entpackt und aufbereitet werden oder auch im Extremfall ganze Datensätze mit mehreren Gigabyte vor einer Anzeige geladen werden.

Die speziell für Peer-to-Peer-Systeme entwickelte Lösung BlobSeer [124] hat ihren Anwendungsbereich ebenfalls in sehr großen Binärdateien, die von sehr vielen Clients konkurrierend geschrieben, gelesen und erweitert werden können.

Die Klasse portabler Dateisysteme ist ebenfalls seit längerem bekannt und wurde bereits aus unterschiedlichen Perspektiven untersucht, wobei Portierbarkeit hier nicht bedeutet, dass die Dateisysteme tatsächlich nativ unter den jeweiligen Betriebssystemen verfügbar sind. So hat z.B. *Mazières* ein Toolkit vorgestellt, dass NFS loopback Server nutzt, um ein Dateisystem im User Space auf UNIX-artigen Betriebssystemen zur Verfügung zu stellen [119].

Ein ebenfalls bekanntes und weit verbreitetes Framework zur Entwicklung von Dateisystemen im User Space ist FUSE [158], das es ermöglicht portable Versionen von Dateisystemen für unterschiedliche Betriebssysteme zu erstellen [133]. Als Framework läuft FUSE selbst im Kernel Space und ermöglicht es im User Space entsprechende Dateisysteme zu starten.

Einzelne Konzepte und Ideen bestehender Dateisysteme spielten beim Entwurf von USPFS eine unterschiedlich große Rolle. So wurde USPFS in der Designphase stark von ZFS [21] motiviert, da hier 128bit große Zeiger verwendet wurden, um die Adressierung für eine sehr große Speicherkapazität zu ermöglichen. Weiterhin hatten einige Varianten der EXT-Familie Einfluss auf das Design, da zum einen die 16GB-Grenze einzelner Dateien (bei 4K Blockgröße) aufgebro-

chen werden sollte. Außerdem sollte auch die maximale Kapazitätgrenze von 10^{18} Bytes unter EXT aufgehoben werden. Die Extents, wie sie z.B. aus EXT und dem Journaled File System (JFS) [16] bekannt sind, regten einen Leistungszuwachs für Lese- und Schreiboperation an, während gleichzeitig die begrenzte Anzahl an speicherbaren Objekten aufgehoben wurde.

5.5 Zusammenfassung

Motiviert durch die Anforderungen des Anwendungsfalls eines verteilten virtuellen Mikroskops wurde das Design eines neu entwickelten, portablen Dateisystems vorgestellt. Im Speziellen wurde eine plattformunabhängige Integration über den User Space präsentiert, um die Hürden eines Einsatzes für den Benutzer möglichst niedrig zu halten. Grundlegend ist neben der Portierbarkeit auch die geforderte Verfügbarkeit erwarteter Sicherheitsmechanismen zur Zugriffsteuerung, Überprüfung der Datenintegrität oder die Verschlüsselung gespeicherter Daten unabhängig von der nativ eingesetzten Kombination aus Betriebssystem und Dateisystem sichergestellt. Verglichen mit anderen portablen Dateisystemen, wie z.B. Implementierungen unter FUSE, wird kein separates Modul im System Space benötigt. Stattdessen kann eine beliebige Kombination aus Betriebs- und Dateisystem als Host verwendet werden.

Besonders die Behandlung sehr vieler kleiner Dateien wurde explizit im Hinblick auf den möglichen Overhead der Metadatenverwaltung optimiert. Eine Inode enthält die Datei-assoziierten Metdadaten (ähnlich zu POSIX) und entsprechende Verweise auf die Nutzdaten, die auch über Extends verteilt organisiert werden können.

Zur adäquaten Verwaltung sehr großer Datenmengen wurde eine zweistufige 128-Bit-Adressierung präsentiert, die eine Unterteilung des Volumes in separate Slices unterstützt. Auf diese Weise lassen sich 2^{63} Slices zu je 2^{63} Byte adressieren, was einer theoretischen Speichergröße von $8 \cdot 10^{19}$ EB inklusive aller Metadaten entspricht.

Die Ergebnisse experimenteller Analysen zeigen, dass der Overhead für die Metadaten verglichen mit der Menge realer Nutzdaten bei lediglich 0,3% liegt.

KAPITEL 6

Sicherheitsaspekte

Dieses Kapitel diskutiert die Sicherheitsaspekte sowie die entwickelten resp. adaptierten Lösungsstrategien für das Omentum Overlay vor. Authentifizierung in Peer-to-Peer-Netzwerken ist aufgrund der häufig unklaren Vertrauensstellungen zwischen einzelnen Peers ein komplexes Themengebiet [71], dem in der Praxis z.B. über ein Web-of-Trust (WoT) begegnet wird. Neben den allgemeinen Anforderungen, die vor allem den Datenschutz und die Persönlichkeitsrechte betreffen, wird ein Verfahren vorgestellt, dass der eindeutigen Identifikation der Knoten im System dient. An einmal identifizierten Knoten können sich nun Benutzer anmelden, deren Daten von ihrer Heimateinrichtung einmalig validiert werden und anschließend von jedem Knoten leicht überprüft werden können. Die entwickelte Authentifizierung nutzt dazu signierte Zertifikate, die über eine kaskadierte Kette verifizierbar sind. Abschließend wir das Konzept gegen bereits existierende Verfahren abgegrenzt.

6.1 Anforderungen

Für die kooperative Vernetzung von teilnehmenden Partnern, z.B. Universitäten, ist eine autarke und dezentrale Login-Infrastruktur erforderlich. Dies soll zum einen die zentrale Aggregation von Login-Informationen vermeiden, was die Sicherheit dieser sensiblen Daten erhöht [89], und zum anderen die zumindest teilweise Verfügbarkeit von Login-Funktionen bei Ausfall einzelner Komponenten der Login-Infrastruktur ermöglichen. Gleichzeitig entspricht dieses Konzept eher einer realen Anwendung, bei der die Kontrolle der Nutzerinformationen den individuellen Konzepten beteiligter Instituten obliegt. Die Authentifizierung der Nutzer soll für jede Session mit einmaligem Kontakt zum Login-Server auskommen und die lokale Überprüfung der Vertrauensstellung ermöglichen.

Medizinische Daten enthalten möglicherweise persönliche Informationen, die es zu schützen gilt. Aus diesem Grund ist die Anonymisierung von Bilddaten zwingend erforderlich, die dem Bereitsteller der Daten überantwortet wird. Die technische Entwicklung, z.B. im Rahmen von Gesichtserkennung, erlaubt möglicherweise auch die nachträgliche Identifizierung von Personen auf dreidimensional rekonstruierten radiologischen Aufnahmen. Daher wird für die Bereitstellung solcher Datensätze zwingend das Einverständnis der abgebildeten Personen benötigt[33]. Zur Dokumentation ist daher jeder Datensatz eindeutig einem Bereitsteller zuzuordnen und die Bereitstellung neuer Datensätze lediglich ausgewählten Knoten zu ermöglichen.

6.2 Identifikation von Knoten

Da lediglich ausgewählte Knoten in der Lage sein dürfen neue Datensätze in das System einzubringen, muss diese Berechtigung verlässlich überprüft werden können.

Die eindeutige Identifizierung der Knoten ist dazu zwingend erforderlich, was über einen systemspezischen, alphanumerischen Code realisiert wird, der auf den UUIDs der eingesetzten Hardware basiert. Über eine kryptographische Hash-Funktion wird die erzeugte Zeichenfolge in einen 64 Bit langen, hexadezimalen Zahlenschlüssel (*NodeID*) umgewandelt. Die Berechnung der NodeID ist plattformspezifisch unterschiedlich und berücksichtigt möglicherweise je nach verwendetem Betriebssystem unterschiedliche Hardwareeigenschaften oder Seriennummern (vgl. Tab. 6.1).

Zur Aktivierung der Bereitstellungsfunktionen wird für berechtigte Knoten von einer zentral verwalteten Instanz ein Serverzertifikat ausgestellt, dass die NodeID enthält. Beim Start des Knotens wird lokal auf die Existenz dieses Zertifikates geprüft und bei gültiger Zertifikatskette sowie übereinstimmender NodeID die Bereitstellung aktiviert.

6.3 Authentifizierung

Die benötigte Login-Infrastruktur wurde über eine hierarchische Anordnung der einzelnen Komponenten realisiert [116]. Zur Authentifizierung werden die an einem Peer erhobenen Benutzerinformationen, z.B. Benutzername und Passwort

[33] Details dazu regelt §§ 22 ff. KunstUrhG [32]

Tabelle 6.1: Kommandos zur Ermittlung der UUID oder Seriennummer unter verschiedenen Betriebssystemen und ihre Zugangsbeschränkung für Konten mit Verwaltungsberechtigung

OS	Root	Kommando	Ergebnis
Mac OS X	-	`system_profiler SPHardwareDataType`	System SN
Ubuntu	+	`dmidecode -t system`	BIOS SN
MS Windows	-	`wmic bios get serialnumber`	BIOS SN

sowie der öffentliche Teil eines sitzungsbezogen berechneten Schlüsselpaares, entweder an den *Omentum-Login-Service* (OLS) oder direkt an den verantwortlichen Server (*Identity Provider*, IP) der angeschlossenen Institution gesendet. Dort werden diese Informationen ggf. mit Hilfe von Intermediate-Instanzen validiert und ein Token in Form eines X.509-Zertifikats [43] zur Teilnahme am Peer-to-Peer-Netzwerk erstellt (vgl. Abb. 6.1). Dieses Zertifikat weist den angemeldeten Benutzer als authentifizierten Teilnehmer am Peer-to-Peer-Netzwerk aus.

Die Gültigkeitsdauer des Benutzerzertifikats ist auf eine Sitzung limitiert. Es werden keine Informationen zum privaten oder öffentlichen Schlüssel gespeichert, weshalb für jede Session ein neues Zertifikat erzeugt werden muss. Auf diese Weise ist es Nutzern auch möglich von zwei unterschiedlichen Endgeräten mit dem Netzwerk zu interagieren, da beide Geräte als eigenständige Instanzen mit unterschiedlichen Zertifikaten agieren.

Viele andere Authentifizierungsverfahren (vgl. Kap. 6.3.2) für Peer-to-Peer-Netzwerke speichern Zugangsinformationen verteilt oder basieren auf einem Web-of-Trust (WoT). Bei einem Web-of-Trust handelt es sich um die indirekte Überprüfung einer Vertrauensstellung, die von zwei wesentlichen Merkmalen geprägt wird:

Gültigkeit Zur Bestimmung ob einem Schlüssel grundsätzlich vertraut werden kann, können unterschiedliche Parameter herangezogen werden. So kann eine nicht zu überschreitende Zeitspanne seit seiner Signierung definiert werden oder eine bestimmte Instanz für die Signatur vorausgesetzt werden.

Vertrauen Im Wesentlichen sind hier zwei Aspekte zu betrachten. So muss zum einen der Tatsache vertraut werden, dass ein Schlüssel tatsächlich dem angenommenen Besitzer zuzuordnen ist und zum anderen das Signierverhalten des Schlüsselinhabers den eigenen Sicherheitsansprüchen genügt.

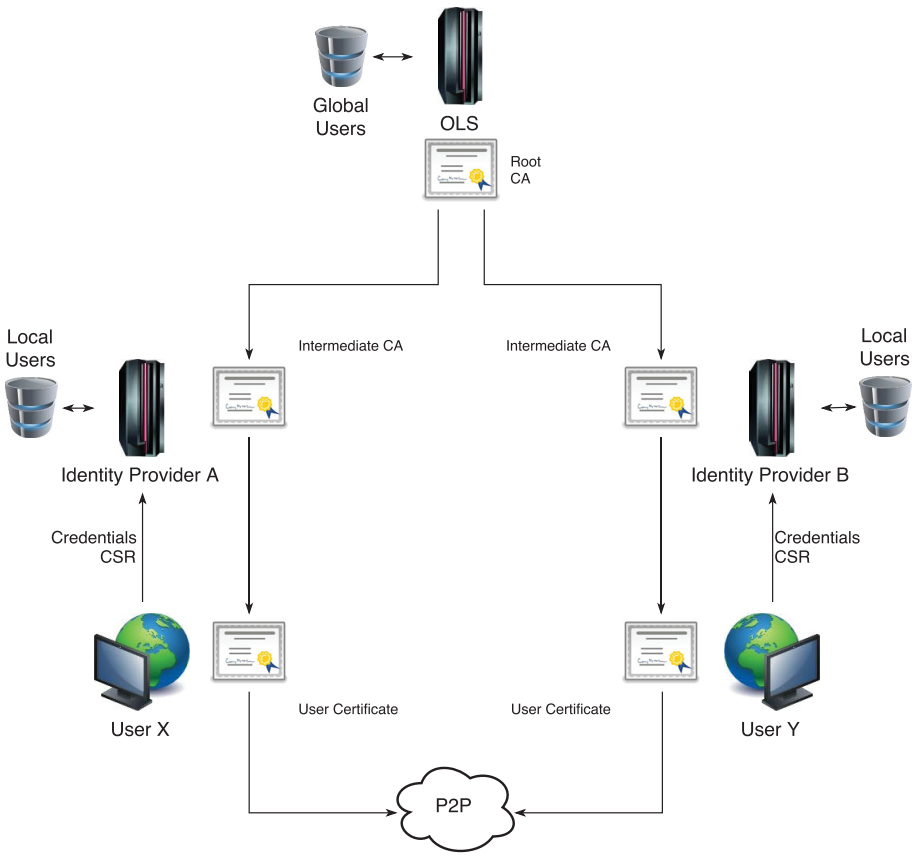
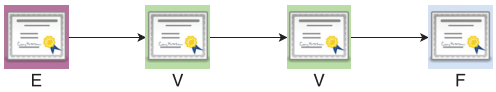
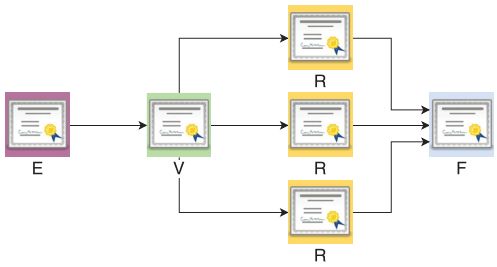


Abbildung 6.1: Schematische Darstellung der Login-Infrastruktur mit zwei zusätzlich beteiligten Institutionen zur Authentifizierung von Nutzern.

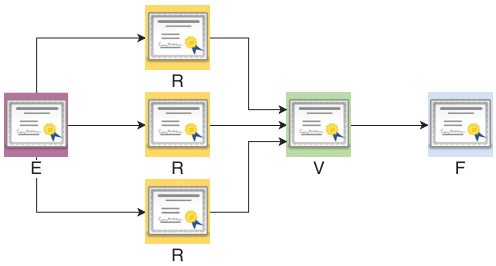
Beide Merkmale sind nicht voneinander abhängig. So kann die Gültigkeit eines Schlüssels mitunter leicht verifiziert werden ohne dem Signierverhalten des Inhabers das nötige Vertrauen entgegen zu bringen. Außerdem könnte dem Signierverhalten eines Schlüssels vertraut werden, aber eine Unsicherheit darüber bestehen, ob der Schlüssel dem erwarteten Besitzer zuzuordnen ist. Verwendet man kaskadierte Vertrauensverhältnisse für die Schlüssel, können die Vertrauensstellungen der einzelnen Schlüssel untereinander geteilt werden. Wurde für Schlüssel A eine Vertrauenstellung mit einem Partner B etabliert und vertraut B einem weiteren Schlüssel C, so könnte A auch C transitiv als vertrauenswürdig ansehen. Die Verwendung komplexerer Validierungsstrategien, wie sie z.B. bei PGP [123] zum Einsatz kommen, ermöglicht eine detailliertere Abstufung von vertrauenswürdigen Schlüsseln (vgl. Abb. 6.2). Im dargestellten Beispiel werden mindestens drei



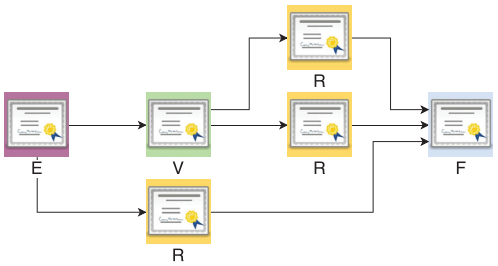
(a) einfache, transitive Vertrauensstellung



(b) kaskadierte Vertrauensstellung über einen vertrauensvollen Schlüssel und drei rudimentärvertrauenswürdigen Schlüssel



(c) kaskadierte Vertrauensstellung über drei rudimentär vertrauenswürdige Schlüssel, die einen vertrauensvollen Schlüssel beglaubigen



(d) kaskadierte Vertrauensstellung über eine gemischte Vertrauensstellung durch drei rudimentär vertrauenswürdige Schlüssel

Abbildung 6.2: Schematische Darstellung unterschiedlicher Vertrauensaggregationen zwischen verschiedenen Schlüsseln (nach [99]). *E* kennzeichnet jeweils den eigenen Schlüssel, *V* kennzeichnet Schlüssel mit vollständigem Vertrauen und *R* markiert Schlüssel, denen nur rudimentär vertraut wird. Mit *F* sind Fremdschlüssel gekennzeichnet, deren Vertrauensstellung über eine Kette bewertet wird.

rudimentär vertrauenswürdige oder ein vollständig vertrauenswürdiger Schlüssel zur elementaren Beglaubigung eines Fremdschlüssels benötigt.

Dieses Verfahren erlaubt die Erzeugung und Signierung von Schlüsseln auf Benutzerebene. Für den vorliegenden Anwendungsfall widerspricht dies der Zugangssteuerung durch die beteiligten Hochschulen.

Das neu entwickelte Verfahren zur verteilten Authentifizierung führt daher eine strikte Zugangskontrolle über *Single Sign On* (SSO) auf Kosten eines Single Point-of-Failures ein. Im Gegensatz zu ähnlichen Systemen wie Shibboleth [165] oder Kerberos [152] wird die Kommunikation mit dem Knoten, der für die Zugangskontrolle verantwortlich ist, jedoch auf einen einzigen Kontakt zu Beginn einer Session beschränkt. Ist die Authentifizierung einmal erfolgreich abgeschlos-

sen, kann bis zum Ablauf des Zertifikats oder der aktuellen Session mit anderen Teilnehmern kommuniziert werden, ohne das ein weiterer Kontakt zum Authentifizierungsknoten erforderlich ist.

6.3.1 Berechtigungen

Individuelle Berechtigung sind neben der grundlegenden Authentifizierung zugelassener Benutzer ebenfalls von zentraler Bedeutung. So kann u.a. der Zugriff auf bestimmte Datensätze oder Annotationen eingeschränkt werden. Die Verwendung von X.509-Zertifikaten als Token nach einer erfolgreichen Authentifizierung erlaubt über die definierbaren Extensions eine granulare Abstimmung einzelner Berechtigungen. Sie können von den Identy Providern aller beteiligten Institutionen selbst vergeben werden und lassen sich sowohl auf Datensatzebene wie auch auf Gruppenebene realisieren.

Der jeweilige OLS kann die Zuordnung eines Benutzers zu einer oder mehreren Gruppen nach erfolgreicher Authentifizierung umsetzen und als Erweiterung im ausgestellten Zertifikat direkt vermerken. Dazu werden eindeutige Object Identifier (OID) erzeugt und entsprechend des PKIX Standards in das Zertifikat integriert. Da die OIDs weltweit eindeutig sein müssen, ist eine zentrale Registrierung unabdingbar. OIDs können z.B. über die Internet Assigned Numbers Authority (IANA) im Bereich „Private Enterprises“ kostenlos erzeugt[34] werden.

Die Möglichkeit einen OID in einem X.509-Zertifikat als *critical* zu markieren, zwingt die Anwendung ein Zertifikat zu verwerfen, sollte die entsprechende Extension nicht ausgewertet werden können. Grundsätzlich empfiehlt der PKIX-Standard die Verwendung von *noncritical*, da in der Regel nicht alle Anwendungen alle privaten Zertifikatserweiterungen jeder Firma erkennen oder verarbeiten können, aber speziell für den vorliegenden Anwendungsfall ist diese zusätzliche Eingrenzung sehr hilfreich. Zum einen kann so sichergestellt werden, dass die Zertifikate an anderen Stellen kaum Verwendung finden können und zum anderen lassen sich Zertifikate ohne diese Extension direkt verwerfen.

6.3.2 Verwandte Arbeiten

Die *Public Key Infrastructure* (PKI) [78] ist vermutlich die bekannteste Authentifizierungsmethode. PKI nutzt zur Authentifizierung hochverfügbare Server, sogenannte Certification Authorities (CA). Das Vertrauen besteht dabei zwischen

[34] Webseite zur OID Registrierung: `http://pen.iana.org/pen/PenApplication.page`

dem CA-Manager, der prinzipiell auch ein Benutzer im System sein kann, und dem Benutzer eines Knotens. Für Peer-to-Peer-Systeme ist die Hochverfügbarkeit der Knoten in der Regel nicht gewährleistet [145], weshalb die Verwendung einer PKI für Peer-to-Peer-Systeme grundsätzlich schwierig ist.

Ein möglicher Ansatz wäre Pretty Good Privacy (PGP) [65], da hier keine permanent verfügbaren Server erforderlich sind. PGP erlaubt eine dezentrale Authentifizierung über ein Web of Trust (WoT), in dem Vertrauensstellungen der Knoten untereinander realisiert werden. Auf diese Weise können Knoten einen Public Key von einem anderen Knoten bekommen, dem sie vertrauen. Problematisch ist hier die Aggregation aller Public Keys, da PGP z.B. keine Routing Tabellen für diese Informationen besitzt. Aus diesem Grund erfordert der Austausch der Schlüssel sehr viele Nachrichten. Weiterhin ist das Management der Schlüssel sehr speicherintensiv. Da eine effiziente Methode für die Abfrage öffentlicher Schlüssel fehlt, kann eine effektive Authentifizierung in Peer-to-Peer-System mit PGP nicht realisiert werden.

Mit Hash-based Distributed Authentication Method (HDAM) [159, 160] stellen *Takeda et al.* ein System zur verteilten Authentifizierung vor, das öffentliche Schlüssel nutzt, die nicht auf einem zentral erreichbaren Server gespeichert, sondern über eine DHT auf die Peers verteilt werden. Der grundlegende Gedanke dieser Authentifizierungsmethode ist ein WoT, über dessen Teilnehmer die öffentlichen Schlüssel aller anderen Teilnehmer erfragt werden können. Vorteilhaft ist die fehlende Bindung an einen erforderlichen Server, allerdings kann in dieser Architektur keine direkte Zugangskontrolle implementiert werden, da die Teilnehmer sich gegenseitig authentifizieren können. Für stärkere Zugangsbeschränkungen müsste die DHT mit den öffentlichen Schlüsseln zentral gewartet werden und es dürfte keine Option zur lokalen Manipulation der DHT existieren.

6.4 Peer-to-Peer und NAT

Aufgrund der Beschränkung von IPv4-Adressen auf 32 Byte sind bei vielen Providern kaum noch freie Adressbereiche zu finden. Aus diesem Grund werden NAT-Gateways eingesetzt. Sie teilen die externe Adresse des Gateways mit allen Geräten, die dahinter über private IP-Adressen verbunden sind. Dabei wird bei Antwortpaketen, die dann das NAT-Gateway erreichen, eine Weiterleitung zum eigentlichen Empfänger vorgenommen. Für Peer-to-Peer-Systeme stellten NAT-Gateways zu Beginn ein größeres Problem dar und konnte häufig nur mit teils

zweifelhaften Verfahren wie *Spoofing* oder DNS-Manipulationen umgangen werden. Durch eine flächendeckende Einführung von IPv6 wird die Notwendigkeit von NAT-Gateways zunächst reduziert. Bis dahin stehen diverse Techniken für die Überwindung von NAT-Gateways zur Verfügung. So lässt sich TCP unter anderem im User Space auf UDP aufsetzen. Weiterhin lassen sich mit STUN [140], TURN [114] und ICE [139] NAT-Gateways überwinden[35]. Die Analyse von STUN-Ergebnisse kann zur Lokalisierung von Knoten verwendet werden, die nicht hinter einem NAT-Gateway verborgen sind, womit das Bootstrapping in Peer-to-Peer-Netzwerken erleichtert werden kann.

Grundsätzlich lassen sich diese Verfahren in einer Firewall unterbinden, allerdings kommt dies z.B. durch die dazu erforderliche Untersuchung des TCP-Headers eher einer Zensur gleich. Neue Konzepte zur Förderung und Unterstützung von Peer-to-Peer-Protokollen, wie z.B. *WebRTC* und *Open Peer* sind in der Entwicklung[36].

Hole punching gilt aktuell als Technik der Wahl und wird daher im Folgenden kurz erläutert. Häufig wird UDP dazu als Übertragungsrotokoll verwendet, was jedoch im vorgestellten Anwendungsfall durch TCP ersetzt wird. Das verwendete Verfahren läuft nahezu analog zu UDP.

Der häufigste Fall dürfte der Kommunikationsversuch zwischen zwei Peers A und B sein, die jeweils hinter unterschiedlichen NAT-Gateways verborgen sind. Soll zwischen beiden Peers eine Verbindung aufgebaut werden, muss ein gemeinsamer Rendezvous-Server R verwendet werden. Dieser kann durch seine unmittelbare Erreichbarkeit die privaten und öffentlichen Adressen von A und B bereits auf Protokollebene ermitteln. Peer A bittet nun Server R um die beiden Adressen von B. Daraufhin antwortet R mit den entsprechenden Informationen und sendet gleichzeitig die Adressen von A an B. Beide Peers versuchen nun unabhängig voneinander über asynchrone Kommunikationskanäle sowohl die öffentliche, wie auch die private Adresse ihres Gegenübers zu erreichen (vgl. Abb. 6.3).

Unter realistischen Bedingungen versuchen beide Hosts zu unterschiedlichen Zeitpunkten eine Verbindung zu den jeweils öffentlichen Adressen ihrer Kommunikationspartner herzustellen. Ist Peer A hinter NAT_A verborgen und Peer

[35] Anschauliche Darstellung zur Funktionsweise von STUN, TURN und ICE: `https://www.quora.com/Internet-Security-How-does-ICE-STUN-work-in-the-context-of-firewall-penetration`

[36] Blogeintrag von 2012 zur Vorstellung von WebRTC und Open Peer: `https://blog.p2pfoundation.net/open-peer-to-allow-direct-peer-to-peer-signaling-to-initiate-connection-between-people-using-browsers/2012/11/19`

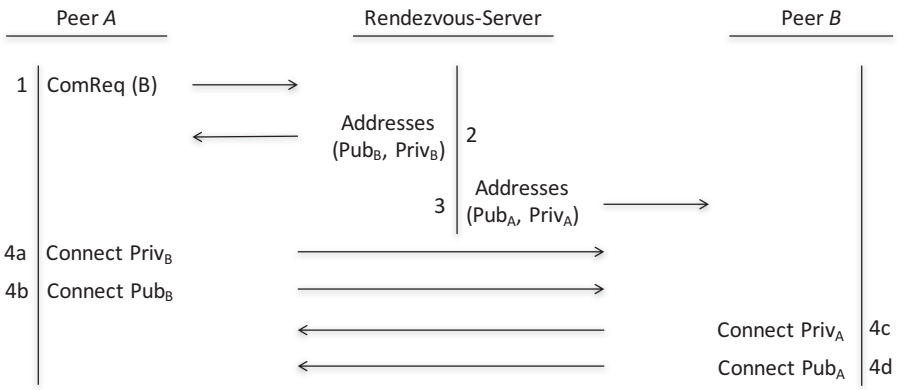


Abbildung 6.3: Protokoll bis zum initialen Verbindungsversuch zwischen zwei Peers hinter unterschiedlichen NAT-Gateways während des *hole punching*.

B hinter NAT_B, dann besteht die Möglichkeit, dass ein SYN-Paket von A bei NAT_B eintrifft, bevor NAT_B ein SYN-Paket von B erhalten hat. In diesem Fall wird das SYN-Paket von A verworfen, da es keine passende Verbindung für dieses Paket gibt. In NAT_A öffnet das Paket hingegen einen Tunnel für mögliche Antworten. Erreicht das SYN-Paket von B anschließend NAT_A, wird das Paket als zugehörig zur Kommunikation zwischen A und B erkannt. Für eine mögliche Betriebssystem-spezifische Implementierung sind an diesem Punkt zwei Varianten grundlegend zu unterscheiden:

- Die TCP-Implementierung von A kann die öffentliche Gegenstelle, die B für das SYN-Paket verwendet hat, aufgrund eines anderen Ports nicht der Gegenstelle zuordnen, zu der A sich ursprünglich verbinden wollte. Aus diesem Grund verknüpft A das ursprüngliche zum Verbindungsaufbau verwendete Socket mit der aktuellen Gegenstelle von B. Da das SYN-Paket von B kein von A erwartetes ACK enthält, antwortet A mit einem SYN-ACK, das als SYN-Anteil lediglich eine Kopie des vorher gesendeten SYN-Pakets enthält.

- Die TCP-Implementierung von A könnte auch erkennen, dass A ein empfangsbereites Socket für eingehende Verbindungen auf dem ursprünglich gewählten Port bereit hält. Das von B gesendete SYN-Paket würde dann wie ein eingehender Verbindungsversuch gewertet, woraufhin A ein neues Stream Socket auf der jetzt neu erstellten TCP Session öffnet. In dem Fall stimmen jedoch die Adressen der beiden zu öffnenden Sockets überein, weshalb der asynchrone Verbindungsaufbau mit einem Verbindungsfehler

für eine bereits belegte Adresse fehlschlagen sollte, was allerdings von der Anwendung ignoriert werden kann, die damit auf jeden Fall eine gültige TCP-Verbindung erhält.

Kommt eine Verbindung zustande, wird der Host authentifiziert. Ist diese Authentifizierung nicht erfolgreich, wird die Verbindung wieder getrennt und der Peer wartet auf weitere eingehende Verbindungen, respektive Antworten auf Verbindungsanfragen.

Als Rendezvous-Server können in Omentum z.B. die Super-Peers fungieren, die von den teilnehmenden Hochschulen zur Bereitstellung ihrer Datensätze verwendet werden. Diese bilden einen vermittelnden Dienst zur Namensauflösung und können so die Informationen aus den unterschiedlichen Nachbarschaften aggregieren.

6.5 Zusammenfassung

Authentifizierung in Peer-to-Peer-Netzwerken ist aufgrund der häufig unklaren Vertrauensstellungen zwischen einzelnen Peers ein komplexes Themengebiet, dem in der Praxis z.B. über ein Web-of-Trust begegnet wird. Hierbei lässt sich das Vertrauen über eine Kette von mehr oder weniger vertrauenswürdigen Schlüsseln herstellen. Für Omentum wurde eine Strategie basierend auf Single Sign On (SSO) umgesetzt, die sich grundlegend von existierenden SSO-Systemen wie Kerberos, Shibboleth oder einem Web-of-Trust unterscheidet. Im Gegensatz zu den genannten Systemen ist bei Omentum lediglich ein einzelner Kontakt zu einem Login-Knoten pro Session erforderlich, da dieser Knoten ein X.509 Zertifikat mit begrenzter Haltbarkeit ausstellt, über dessen Zertifizierungskette sich die Teilnehmer untereinander authentifizieren können. Ein weiterer Vorteil der Verwendung dieser Zertifikate liegt in der Möglichkeit einer flexiblen Erweiterung um Zugriffssteuerungen und Berechtigungen für einzelne Nutzer. Diese können aufgrund der hierarchischen Authentifizierung von den jeweiligen Identity Providern gesondert vergeben werden.

Die Kontrolle von Zugangsinformationen ist mit den vorgestellten Systemen PKI, PGP und HDAM nicht in vollem Umfang für ein Peer-to-Peer-System erfüllbar. Lediglich die Authentifizierung über verteilt gespeicherte öffentliche Schlüssel ist möglich. In Omentum wird die Zugangskontrolle über zertifizierte Schlüsselpaare realisiert, deren öffentlicher Schlüssel für die Validierung der Zertifikatskette herangezogen wird. Nach dem für jede Verbindung einmaligen Authentifi-

zierungsprozess lassen sich anwendungsspezifische Rechte, die vom Login Service eingetragen werden, ebenfalls aus dem verwendeten X.509-Zertifikat auslesen.

Die Umgehung der NAT durch hole punching wird über ein separates Super-Peer-Overlay zur Namensauflösung innerhalb von Omentum ermöglicht. Diese Super-Peers sind als Datenquellen der einzelnen Provider ohnehin erreichbar und können diese zusätzliche Vermittlung übernehmen.

KAPITEL 7

Zusammenfassung und Ausblick

Die Bedeutung der Entwicklung von Omentum für die praktische Anwendung in der medizinischen Lehre wird in Kapitel 7.1 vorgestellt und von der Aufstellung der wissenschaftlichen Ergebnisse dieser Arbeit in Kapitel 7.2 ergänzt, bevor abschließend ein Ausblick auf mögliche weitere Forschungsaspekte in Kapitel 7.3 gegeben wird.

7.1 Praktische Anwendung

Im Rahmen dieser Arbeit wurde mit Omentum das erste Peer-to-Peer-System entwickelt, das sich auf virtuelle Mikroskopie – als Beispielanwendung für verteilte interaktive Lernplattformen – fokussiert. Als besondere Herausforderungen sind zum einen die sehr große Datenmenge von rund 50 GB je Datensatz bei mehreren tausend Datensätzen und zum anderen der hohe Grad an Interaktionen zwischen vielen Benutzern zu nennen. Für den Einsatz in der Lehre, der bereits in unterschiedlichen Entwicklungsstufen seit 2014 mit jährlich etwa 500 Studierenden etabliert ist, wurde eine geeignete Datenaufbereitung entwickelt, mit der die viele Gigabyte großen und meist proprietären Eingangsdaten in eine JPG-Pyramide überführt wurden, die eine effiziente Partitionierung der Daten für eine verteilte Speicherung im Netzwerk und eine Datenanforderung aus multiplen Quellen ermöglicht. Neben den 2D-Datensätzen wurde das Anwendungsspektrum auf dreidimensionale Volumendaten ausgeweitet, für die das Prinzip des Texture Cubes im Rendering-Prozess um Subcubes ergänzt wurde, die sich direkt in die entwickelte pyramidale Architektur der aufbereiteten Daten integrieren lassen.

Um die Daten optimal auf den Peers speichern zu können, wurde ein neues portables User Space Dateisystem (USPFS) implementiert, das unabhängig von den auf den Peers verwendeten Betriebssystem-Dateisystem-Kombinationen

die Verfügbarkeit von grundlegenden Funktionen und Sicherheitsmechanismen, wie eine effiziente Kompression oder Verschlüsselung, garantiert. Der maximale, theoretisch mögliche Speicher, der aggregiert aus mehreren Quellen mit einer USPFS-Instanz verwaltet werden kann, liegt mit $8 \cdot 10^{19}$ EB oberhalb derzeitig verfügbarer Speichermedien und ist damit zukunftssicher. Speziell der Speicherbedarf für die Metadaten stand hier aufgrund der vielen sehr kleinen Dateien im Vordergrund und konnte mit nur 0,3% der Nutzdaten minimiert werden.

Besonders ressourcenarme mobile Endgeräte wurden ebenfalls als Clients in das System integriert und sind als Leecher über Routing-Peers von den komplexen Berechnungen für den Peer-to-Peer-Inhalt befreit. So kann ein großer Anteil der geringen Bandbreite in Mobilfunknetzen eingespart werden.

7.2 Resultat

Aufgrund der kontextspezifischen Interaktionsmuster und der gespeicherten Datenvolumina sind bestehende Peer-to-Peer-Systeme nicht für einen Einsatz als Infrastruktur-Plattform für virtuelle Mikroskopie geeignet. Das neu entwickelte Omentum Overlay verwendet die durch die Aufbereitung entstandenen Partitionen als virtuelle Knoten im Netzwerk, die auf beitretende Peers verteilt werden. Vergleichend wurden unterschiedliche Implementierungen strukturierter Overlays analysiert und es konnte gezeigt werden, dass strukturierte Overlays für Index-basierte Suchen in dynamischen Umgebungen in der Regel keine kürzeren Pfade ermöglichen, sondern mitunter zu komplexeren Nachbarschaften führen, womit sie schlechter skalieren, als die vorgestellte, unstrukturierte Architektur für Omentum. Unstrukturierte Overlays bieten aufgrund ihrer Architektur in der Regel keine Möglichkeiten für Index-basierte Lookups. Das Omentum Overlay bietet neben der effizienten Kombination aus komplexen Suchen und Index-basierten Abfragen eine verkehrslose Routenermittlung, deren Länge mit $\frac{\log N}{\log c}$ in etwa der einer vergleichbaren DHT entspricht. Ein Vorteil des Omentum Overlays ist, dass die verwendete Nachbarschaft c klein und völlig unabhängig von der Größe des Netzwerks N ist.

In der aktuellen Routing-Phase kann die Routenlänge dank einer innovativen, zweistufigen Pfadoptimierung nochmals reduziert werden, in dem die lokalen Informationen einer Replikatsnachbarschaft mit einbezogen werden. Die Verkürzung der Pfade ist dabei abhängig von der Größe des Netzwerks und erlaubt in größeren Netzwerken eine häufigere Reduktion. Ferner erlauben die Replikatsnachbar-

schaften unter Berücksichtigung der Leistungen einzelner Peers und aggregierter Leistungsinformationen auf Ebene der Nachbarschaften virtueller Knoten eine optimierte Replikationsstrategie, die sowohl wechselnde Anfragehäufigkeiten einzelner Daten wie auch Lastspitzen effektiv handhaben kann.

Da die Super-Peers in einem separaten Overlay untereinander verbunden sind und die Peers mit gemeinsamen Replikaten ebenfalls verknüpfte Nachbarschaften bilden, ist das Omentum Overlay besonders robust und Knotenausfälle sind leicht zu kompensieren sowie Lasten sehr effektiv zu verteilen. Die einzelnen Peers besitzen in Omentum die einzigartige Eigenschaft in Abhängigkeit ihrer eigenen Leistungsfähigkeit bei Überlastung gezielt das übrige Netzwerk um Unterstützung ihrer Aufgaben zu fragen. Durch Ausnutzung der System-spezifischen Architekturmerkmale ist im Omentum Overlay auch bei Ausfall von 80% der Knoten in einem simulierten Netzwerk mit einer Million Knoten die Wiederherstellung eines stabilen und fehlerfreien Zustands bereits in knapp unter einer Minute möglich.

Für die eindeutige Authentifizierung der Benutzer wurde eine kaskadierte Struktur auf Basis dynamisch erzeugter Schlüsselpaare nach dem Prinzip des *Single Sign On* über signierte X.509-Zertifikate realisiert, die mit lediglich einer einzigen Validierung der Login-Informationen zum Session-Beginn eine zuverlässige und ressourcenschonende Peer-to-Peer-Authentifikation erlaubt. Sämtliche Kommunikation im Peer-to-Peer-Netzwerk wird anhand dieser Zertifikate über eine leistungsfähige Kommunikationsschnittstelle, die zusätzlich mit einer Rückwärtsfehlerkorrektur eine effektive Wiederherstellung relevanter Nachrichten im Fehlerfall ermöglicht, gesichert.

Die Leistungsbewertung der einzelnen Peers im Overlay wurde durch eine neuartige Durchsatzabschätzung mittels *passive probing* über TCP mit Congestion Control optimiert, da speziell der verfügbaren Bandbreite in Peer-to-Peer-Netzwerken eine besondere Bedeutung zukommt. Die Reaktionsgeschwindigkeit auf eine Bandbreitenänderung durch die Verwendung bereits erhobener Messdaten beeinflusst die Wahrnehmung von einzelnen Lastspitzen. Es konnte eine sehr schnelle Reaktion mit einer Messgenauigkeit von 95% bei einem stabilen Verlauf nachgewiesen werden.

7.3 Ausblick

Neben den klassischen und sehr bildlastigen Fächern der Medizin wie Histologie, Pathologie und Radiologie, die mit Omentum bereits im Lehr- und Prüfungskon-

text bedient werden können, lassen sich durch die modulare Struktur z.B. auch Compiler für verschiedene Sprachen im Rahmen der naturwissenschaftlichen Bildung integrieren. Ebenso könnten Module für die Berechnung sowie Darstellung von mathematischen Funktionen entwickelt und Schnittstellen zu bestehenden Anwendungen wie z.B. MATLAB implementiert werden.

Da das Omentum Overlay seit 2014 bereits praktisch und mit großem Erfolg in vielen Lehrveranstaltungen eingesetzt wurde, wäre eine Analyse der Interaktionsmuster der Anwender sehr nützlich um das System von einem reinen Lernwerkzeug zu einem Wissensnetzwerk auszubauen. So könnte aufgrund des Arbeitsverhaltens einzelner Nutzer die Anforderungsstrategien der Daten individuell optimiert werden.

Weiterhin könnten neben den allgemein zugänglichen Bereichen auch speziell Forschungsgruppen weltweit verteilt an gemeinsamen Datenbeständen arbeiten oder bestehende Forschungssysteme wie z.B. OMERO[37] an Omentum angebunden werden. Ein wichtiger Punkt hier ist definitiv die Datensicherheit, damit die Forschungsprojekte aller Beteiligten untereinander autark bleiben und eine Manipulation von Ergebnissen ausgeschlossen werden kann.

Auch die aufkommende Verbreitung von augmented Reality (AR) und virtual Reality (VR) wäre eine Möglichkeit das Interaktionspotential von Omentum auszubauen. Entsprechende Datensätze könnten analog zu den bereits unterstützten 3D-Daten aufbereitet und die Steuerung der jeweiligen Eingabe- und Darstellungsgerät abstrahiert werden.

Das portable Dateisystem (USPFS), das unter Java entwickelt wurde, und ohne externe Bibliotheken auskommt, könnte um ein zentrales Logging und entsprechende Roll-Back-Funktionen erweitert werden, um den Grad an Sicherheit weiter zu steigern. Ferner kann durch einen koordinierten Schreib- und Lese-Zugriff über mehrere Threads sicherlich noch eine deutliche Optimierung der Leistung erreicht werden.

Im Umfeld verteilter Systeme gilt es die aus den Messungen mit mehreren Hundert Benutzern antizipierten Verhaltensmuster mit realen Daten sehr vieler Benutzer abzugleichen. Neben den Interaktionsmustern sollte auch das Lastverhalten der einzelnen Knoten in einem großen System real analysiert und mit den Annahmen abgestimmt werden, um mögliche Optimierungsparameter zu ermitteln. Gleichzeitig wird mit Omentum speziell für die Entwicklung ausgewogener heuristischer Algorithmen zur intelligenten und effizienten Datenanforderung aus

[37] OMERO Webseite: `http://www.openmicroscopy.org/site/products/omero`

analysierten Interaktionsmustern ein breites Anwendungsfeld eröffnet, auf dem unterschiedlichste Strategien evaluiert werden können.

ANHANG A

Nachrichtenübertragung

In Omentum kommt der Übermittlung von Nachrichten eine besondere Bedeutung zu, da häufig zeitgleich auf Nachrichten von vielen Kommunikationspartnern reagiert werden muss. Auch sind geöffnete Verbindungen nicht unbegrenzt gültig und werden häufig auf- und wieder abgebaut. Die für die Authentifizierung dynamisch erzeugten Schlüssel und Zertifikate werden hier in einem KeyStore vorgehalten, der lediglich im flüchtigen Speicher existiert. Um den Anforderungen gerecht zu werden, wurde das nachfolgend vorgestellte Konzept für asynchrone Sockets optimiert und um das vorgestellte Authentifizierungsverfahren erweitert. Auf diese Weise werden blockierende Aufrufe der Schnittstelle vermieden und die Sicherheit nicht beeinträchtigt.

Unterschiedliche Threadpools für wichtige Teilbereiche der Kommunikation erlauben ein hohes Maß an Parallelisierung in der Nachrichtenverarbeitung (vgl. Abb. A.1). Der in einem eigenen Thread arbeitende *Acceptor* kümmert sich ausschließlich um die Annahme von neuen Verbindungsanfragen und reicht diese mit dem entsprechenden *Channel* an den ebenfalls autonomen *Dispatcher* weiter. Dieser Prozess läuft im Millisekundenbereich ab, so dass der Acceptor sehr schnell wieder für die Annahme von neuen Verbindungsanfragen zur Verfügung steht.

Im Dispatcher wird auf Empfängerseite der Channel registriert und ein spezifischer *Handler* angehängt. Dieser Handler ist für die Arbeit mit den Ein- und Ausgabe-Puffern verantwortlich, die zum Lesen und Schreiben im Channel verwendet werden. Anschließend wird der Channel als *connectable* markiert, was auf der Senderseite dazu führt, dass der Dispatcher den Channel mit einem zugehörigen Handler registriert und anschließend dem *ConnectionManager* die Verbindung zum initialen Verbindungsaufbau und zur weiteren Verwaltung überlässt.

Nach einem erfolgreichen Verbindungsaufbau können beide Seiten den Channel zur Kommunikation verwenden und entsprechend ein asynchrones Lese- und/

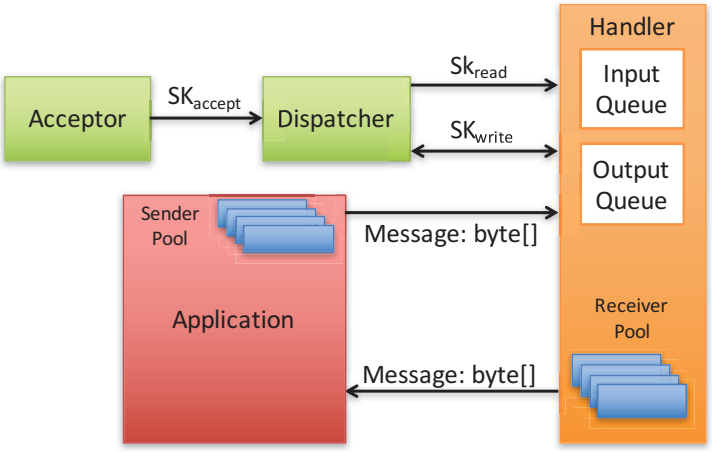


Abbildung A.1: Schematische Darstellung der Kommunikationsschnittstelle mit eigenen Threads für den Acceptor, den Dispatcher und den separaten Threadpools (blau) für die Verarbeitung auf Sender- und Empfängerseite.

oder Schreibinteresse anmelden. Findet Kommunikation auf einem Channel statt, wird der angehängte Handler mit der Bearbeitung beauftragt. Er liest die eingegangenen Bytes auf dem Channel ein und leitet diese, sobald die Nachricht vollständig empfangen wurde, über einen separaten Threadpool an den per *Reflection API* angebundenen *InputHandler* der Anwendung weiter.

Jeder Nachricht wird unmittelbar vor dem Versand eine Längeninformation vorangestellt, anhand derer die Anzahl jeweils zugehöriger Bytes vom Handler ermittelt werden kann. Eine spezifische Markierung für ein Nachrichtenende lässt sich aufgrund der heterogenen Daten, die zu übertragen sind, nicht allgemeingültig einführen. Die Entkopplung der einzelnen Komponenten und der modulare Aufbau ermöglichen die protokollunabhängige und individuell erweiterbare Implementierung einer robusten und performanten Kommunikationsschnittstelle.

Im Rahmen des Aufbaus einer sicheren Verbindung zwischen zwei Knoten, die auf TLS [49] basiert, werden die Zertifikate beider Knoten authentifiziert. Ein Kommunikationskanal zwischen den Knoten wird erst dann geöffnet, wenn die Authentifizierung auf beiden Seiten erfolgreich abgeschlossen wurde. Im Fehlerfall wird der Kommunikationsversuch abgebrochen. Daher können Nachrichten ausschließlich zwischen berechtigten Knoten ausgetauscht werden, an denen ein Benutzer angemeldet ist. Eine Ausnahme hier bilden die in Kapitel 4 beschriebenen Super-Peers, denn sie stehen auch ohne angemeldeten Benutzer als Datenquelle zur Verfügung. Anfordern können sie Daten jedoch nicht.

Für das vorgeschlagene Verfahren ist die Erreichbarkeit eines Login-Servers erforderlich. Da ein Konzept für mehrere Login-Server vorgesehen ist, die Teilbereiche des Benutzerraumes verwalten, können beim Ausfall eines Login-Servers bereits angemeldete Nutzer sowie auch alle Nutzer mit anderem Identity Provider die Anwendung weiterhin ohne Einschränkungen verwenden.

Durch die Verfügbarkeit unterschiedlicher Angriffsverfahren auf TLS/SSL ist die Effektivität von TLS zur Kommunikationssicherung seit einigen Jahren angezweifelt [148].

ANHANG B

Nachrichten

B.1 Message: Der gemeinsame Header aller Nachrichten

Type	**MessID**	**CorrID**	**SendID**	**SendIP**	**RecID**	**RecIP**	**CP**
1 Byte	8 Byte	8 Byte	4 Byte	4 Byte	4 Byte	4 Byte	1 Byte

Type Über diesen Parameter werden Nachrichtentypen voneinander differenziert.

MessID Die MessageID identifiziert eine Nachricht für jeden Sender eindeutig und findet hauptsächlich in der Nachrichtenverwaltung und dem Checkpointing Verwendung.

CorrID Über die CorrespondenceID kann eine Nachricht zwischen beteiligten Kommunikationspartnern eindeutig einer spezifischen Unterhaltung zugeordnet werden.

SendID/RecID Dies sind die systemweit eindeutigen Identifikationsnummern für den Sender und Empfänger. Diese IDs werden aus der Hardware der beteiligten Geräte ermittelt.

SendIP/RecIP Diese Parameter enthalten die IP-Adressen des Senders und des Empfängers.

CP Mit diesem Schalter wird das System darüber informiert, ob die Nachricht in den Checkpointing-Prozess zu integrieren ist.

B.2 Nachrichten der JOIN-Operation

B.2.1 JoinRequest: Beitrittsanfrage bei einem Bootstrapknoten

Leecher
1 Byte

Leecher Über diesen Parameter werden mobile Endgeräte beim Beitritt zum Netzwerk identifiziert, da sie hier separat betrachtet werden (vgl. Kap. B.4).

B.2.2 JoinResponse: Antwort auf eine Beitrittsanfrage

vNodeID
4 Byte

vNodeID Über dieses Feld wird dem beitretenden Knoten mitgeteilt, an welchen virtuellen Knoten er sich für die initiale Replikation eines Datensatzes wenden soll. Ist dieses Feld 0, wurde der Beitritt abgelehnt.

B.2.3 SlideListRequest: Liste der verfügbaren Datensätze

ListHash
4 Byte

ListHash Dieser Parameter wird auf jedem Knoten als Hashwert aus der Liste ihm bekannter Datensätze berechnet. Auf der Empfängerseite der Nachricht wird dieser Hashwert als Indikator verwendet, ob eine Aktualisierung der Datensätze auf der Senderseite erforderlich ist.

B.2.4 SlideListResponse: Übertragung der verfügbaren Datensätze

ID	**PathLength**	**Path**	**Type**	**Format**	**OwnerNr**
8 Byte	4 Byte	p Byte	1 Byte	1 Byte	4 Byte

SlideNr	**Width**	**Height**	**TWidth**	**THeight**	**SMI**
4 Byte	4 Byte	4 Byte	4 Byte	4 Byte	92+s Byte

Res	**Mag**	**Planes**	**DefPlane**	**Visible**	**ScanLen**
8 Byte	4 Byte	4 Byte	4 Byte	1 Byte	4 Byte

ScanText	**Parts**	**ThLen**	**Thumb**
t Byte	4 Byte	4 Byte	i Byte

ID Dieser Wert identifiziert einen Datensatz systemweit eindeutig.

PathLength, Path In diesen Feldern ist die Länge des Verzeichnisnamens sowie der Verzeichnisname selbst enthalten. Dieser spiegelt häufig den original verwendeten Namen des Datensatzes wieder.

Type Über dieses Feld werden unterschiedliche Datensatztypen (Dicom, 3D etc.) voneinander differenziert.

Format In diesem Feld ist das Farbformat der zum Datensatz gehörenden Kacheln festgelegt. Auf diese Weise lassen sich unterschiedliche Bilddaten, z.B. für die Grauwertfensterung mit einer Hounsfield-Skala [135], betrachten.

OwnerNr Mit diesem Feld wird der ursprüngliche Besitzer des Datensatzes identifiziert.

SlideNr Dieses Feld enthält die fortlaufende Nummerierung der Datensätze eines Besitzers.

Width, Height In diesen Parametern ist die Breite und Höhe (in Pixeln) des gesamten Datensatzes gespeichert.

TWidth, THeight In diesen Parametern ist die Breite und Höhe (in Pixeln) einer einzelnen Kachel des Datensatzes gespeichert.

SMI Mit diesem Feld wird die Zuordnung zu einem systemweit eindeutigen Eintrag für die Metadaten zu diesem Datensatz festgelegt. Zu diesen Metadaten gehören unter anderem eine Kurz- und Langbeschreibung des Datensatzes sowie Informationen über die verwendete Methoden zur technischen Bildakquise, Aufbereitung und die Herkunft des dargestellten Probenmaterials. Die entgültige Länge ist daher abhängig von den mitgelieferten Zusatzinformationen.

Res Dieser Wert legt die Auflösung eines einzelnen Pixels in Mikrometern fest.

Mag In diesem Feld ist die maximale Vergrößerung des Datensatzes beschrieben.

Planes Hier sind die Anzahl der im Datensatz verfügbaren Schichten in der z-Achse definiert.

DefPlane Mit diesem Parameter wird die Ebene festgelegt, die für die initiale Anzeige des Datensatzes verwendet werden soll.

Visible Über dieses Feld wird die Sichtbarkeit des Datensatzes gesteuert.

ScanLen, ScanText Diese Felder enthalten die Länge des Textes für die Information über den verwendeten Scanner, sowie den Text selbst.

Parts In diesem Feld ist die Anzahl der berechneten Partitionen des Datensatzes gespeichert.

ThumbLen, ThumbData Diese Felder enthalten die Länge der Bilddaten, sowie die Bilddaten selbst, die für das Vorschaubild des Datensatzes verwendet werden.

B.2.5 CreateReplicaRequest: Anfrage zur Erzeugung eines Replikats

vNodeID	**NumParts**	**Reps**	**NeighIn**	**NeighOut**	**NeighReps**
4 Byte	4 Byte	$8 \cdot x$ Byte	$4 \cdot x$ Byte	$4 \cdot x$ Byte	$(4 + 8 \cdot x) \cdot y$ Byte

vNodeID Dieser Wert identifiziert die Partition/den virtuellen Knoten, der repliziert werden soll.

NumParts Dieses Feld gibt an, aus wie vielen Teilen (Kacheln) die Partition besteht.

Reps In dieser Liste werden die übrigen Replikate genannt, damit die Replikation aus mehreren Quellen simultan erfolgen kann.

NeighIn, NeighOut Über diese Liste werden die Nachbarknoten, die über eingehende/ausgehende Kante verbunden sind, definiert.

NeighReps Diese Liste enthält die Adressen der physikalischen Peers, die für die Nachbarknoten verantwortlich sind.

B.2.6 CreateReplicaResponse: Antwort auf eine Anfrage zur Replikaterstellung

vNodeID
4 Byte

vNodeID Dieser Wert identifiziert die Partition/den virtuellen Knoten, für den die Replikaton gestartet werden soll. Ist dieser Wert mit 0 angegeben, gilt die Replikation als abgelehnt.

B.2.7 NewActiveReplicaNotification: Mitteilung über ein neu erstelltes Replikat

vNodeID	**Replicas**
4 Byte	$8 \cdot x$ Byte

vNodeID Dieser Wert identifiziert die Partition/den virtuellen Knoten, der vom Sender vollständig repliziert wurde.

Replicas Die vom Sender aktualisierte Liste der Replikate wird zur Synchronisierung der Nachbarschaft verwendet, da sie wertvolle Informationen über die jeweiligen Auslastungen der beteiligten Peers im Rahmen der Replikation enthält.

B.2.8 DataRequest: Anforderung der Kachel aus einer Partition

vNodeID	TileID
4 Byte	4 Byte

vNodeID Dieser Parameter identifiziert den virtuellen Knoten (die Partition), aus der die nachfolgend spezifizierte Kachel angefordert werden soll.

TileID Mit dieser innerhalb einer Partition eindeutigen Nummer wird die exakte Kachel spezifiziert, die angefordert werden soll.

B.2.9 DataResponse: Übertragung einer Kachel aus einer Partition

vNodeID	TileID	ImageLength	ImageData
4 Byte	4 Byte	4 Byte	x Byte

vNodeID Dieser Parameter identifiziert den virtuellen Knoten (die Partition), aus der die nachfolgend spezifizierte Kachel stammt.

TileID Mit dieser, innerhalb einer Partition eindeutigen Nummer wird die exakte Kachel spezifiziert, die mit dieser Nachricht gesendet wird.

ImageLength In diesem Parameter ist die Länge der nachfolgenden Bilddaten definiert.

ImageData In diesem Feld sind die Bilddaten hinterlegt, die zur übertragenen Kachel gehören.

B.3 Nachrichten der LEAVE-Operation

B.3.1 LeaveNotification: Information zum Verlassen der Replikatsnachbarschaft

Reason
1 Byte

Reason Über diesen Parameter kann den Empfängern ein möglicher Grund für das Ausscheiden des Peers mitgeteilt werden. So kann eine Wiederaufnahme in die entsprechenden Nachbarschaften vereinfacht werden.

B.3.2 LeaveRegistered: Bestätigung zum Ausscheiden eines Replikats

Diese Nachrichten dienen der Bestätigung der Auflösung eines Replikats, damit der auflösende Knoten erkennt, dass seine Benachrichtigung verarbeitet wurde.

B.3.3 ReplicaResolved: Information zur Replikatsauflösung

Diese Nachricht wird vom ausscheidenden Peer an das koordinierende Replikat versendet, sobald alle anderen Replikate das Ausscheiden bestätigt haben.

B.3.4 ReplicaListUpdate: Aktualisierung der Replikatsliste

Replicas
$8 \cdot x$ Byte

Replicas Diese Liste enthält die aktuelle Replikatsnachbarschaft, die vom koordinierenden Peer zum Abgleich an alle Replikate versendet wird.

B.4 Nachrichten für mobile Clients

B.4.1 ContactRouter:

Router
8 Byte

Router Dieser Parameter liefert die Kontaktinformationen des ausgewählten Routers.

B.4.2 RegisterLeecher: Anfrage zur Registrierung als Leecher

Mit dieser Nachricht wendet sich ein Leecher an den Router und bittet um die lokale Registrierung als Leecher. Zusätzliche Parameter sind nicht erforderlich, da lediglich der Nachrichtentyp ausgewertet wird.

B.4.3 LeecherRegistered: Antwort auf eine Registrierungsanfrage

Status
1 Byte

Status Der Status gibt an, ob eine Registrierung angenommen wird (0) oder aus welchem Grund sie abgelehnt wurde.

B.5 Nachrichten für dynamische Objekte

B.5.1 AnnotationAdded: Hinzufügen einer neuen Annotation

Annotation
x Byte

Annotation Dieses Feld definiert die zu speichernde Annotation mit all ihren Parametern (vgl. Kap. 2.3.1). Die Größe des Feldes ist von der Komplexität der Annotation und der Länge ihrer Beschreibung abhängig.

B.5.2 AnnotationSynchronize: Änderung einer Annotation

Diese Nachricht ist eine Erweiterung der Nachricht zur Erzeugung einer neuen Annotation und stellt lediglich einen neuen Subtyp zur Verarbeitung des Ereignisses bereit.

B.5.3 AnnotationRequest: Suchanfrage nach Annotationen eines Datensatzes

ID	**Search**
8 Byte	x Byte

ID Dieses optionale Feld identifiziert den Datensatz zu dem die Annotationen abgefragt werden sollen. Ist es mit 0 initialisiert, werden alle Datensätze durchsucht.

Search Über diesen Parameter kann ein Suchstring angegeben werden, der mit den Annotationen zu einem Knoten verglichen werden soll. Ist kein Wert angegeben, werden alle Annotationen angezeigt.

B.5.4 AnnotationResponse: Antwort auf eine Suchanfrage nach Annotationen

Annotations
x Byte

Annotations Dieses Feld enthält die Annotationen, die den Suchparametern der *AnnotationRequest*-Nachricht entsprechen.

B.5.5 InteractionAdded: Hinzufügen einer aufgezeichneten Interaktionsfolge

UserAction
variabel

UserAction Dieses Feld definiert die zu speichernde Interaktionsfolge mit all ihren Parametern (vgl. Kap. 2.3.2). Die Größe des Feldes ist von der Komplexität der Aktionsfolge und der Länge ihrer Beschreibung abhängig.

B.5.6 InteractionSynchronize: Ändern einer aufgezeichneten Interaktionsfolge

Diese Nachricht ist eine Erweiterung der Nachricht zur Speicherung einer neuen Interaktionsfolge und stellt lediglich einen neuen Subtyp zur Verarbeitung des Ereignisses bereit.

B.5.7 InteractionsRequest: Suchanfrage nach aufgezeichneten Interaktionsfolge

ID	**Search**
8 Byte	variabel

ID Dieses optionale Feld identifiziert den Datensatz zu dem die Interaktionsfolgen abgefragt werden sollen. Ist es mit 0 initialisiert, werden alle Datensätze durchsucht.

Search Über diesen Parameter kann ein Suchstring angegeben werden, der mit den Beschreibungen der Interaktionsfolgen zu einem Knoten verglichen werden soll. Ist kein Wert angegeben, werden alle Interaktionsfolgen angezeigt.

B.5.8 InteractionsResponse: Suchantwort für gespeicherte Interaktionen

UIList
x Byte

UIList Dieses Feld enthält die Liste der aufgezeichneten Benutzeraktionen, die den Suchparametern der *InteractionsRequest*-Nachricht entsprechen.

B.6 Nachrichten für statische Objekte

B.6.1 VNodeJoinRequest: Hinzufügen eines neuen Datensatzes

vNodeID
4 Byte

vNodeID Dieses Feld enthält die systemweit eindeutige Identifikation für den virtuellen Knoten der die Daten der entsprechenden Partition repräsentiert.

B.6.2 VNodeJoinResponse: Antwort auf das Hinzufügen eines neuen Datensatzes

vNodeID	**Neighbors**
4 Byte	$(4 + (8 \cdot x) \cdot y)$ Byte

vNodeID Dieses Feld enthält die systemweit eindeutige Identifikation für den virtuellen Knoten der die Daten der entsprechenden Partition repräsentiert.

Neighbors Über diese Liste wird dem speichernden Peer die berechnete Nachbarschaft für den neuen virtuellen Knoten mitgeteilt.

B.6.3 TileRequest: Anfrage eines statischen Bildelements

ID	**x**	**y**	**z**	**Layer**
8 Byte	4 Byte	4 Byte	4 Byte	4 Byte

ID Dieses Feld enthält die systemweit eindeutige Identifikation für den Datensatz, dem dieses Bildelement zugeordnet ist.

x,y,z In diesen Parametern ist die Position des Bildelements im Koordinatenraum des Datensatzes definiert.

Layer Über diesen Wert wird die Fokusebene/Zeitachse definiert, zu der das Bildelement gehört.

B.6.4 TileResponse: Antwort auf eine Anfrage statischer Bildelemente

ID	**x**	**y**	**z**	**Layer**	**Image**
8 Byte	4 Byte	4 Byte	4 Byte	4 Byte	x Byte

ID Dieses Feld enthält die systemweit eindeutige Identifikation für den Datensatz, dem dieses Bildelement zugeordnet ist.

x,y,z In diesen Parametern ist die Position des Bildelements im Koordinatenraum des Datensatzes definiert.

Layer Über diesen Wert wird die Fokusebene/Zeitachse definiert, zu der das Bildelement gehört.

Image Diese Feld enthält die dem angeforderten Objekt zugeordneten Bilddaten.

B.7 Sicherheitsrelevante Nachrichten

B.7.1 CSR: Benutzeranfrage zur Authentifizierung beim OLS

Username	**Password**	**CSR**
x Byte	x Byte	x Byte

Username, Password In diesen Parametern werden die erhobenen Zugangsdaten verschlüsselt übertragen, die vom OLS validiert werden sollen (vgl. Kap. 6).

CSR Dieses Feld enthält den *Certificate Signing Request*, der vom OLS nach Authentifizierung der Zugangsdaten signiert werden soll.

B.7.2 CSRResponse: Antwort des OLS auf eine Authentifizierungsanfrage

Certificate
x Byte

Certificate Diese Feld enthält das vom OLS signierte X.509-Zertifikat des Benutzers, falls die Validierung der Zugangsdaten erfolgreich war. Andernfalls wird kein Zertifikat ausgeliefert und eine Teilnahme am Netzwerk ist nicht möglich.

B.8 Allgemeine Nachrichten

B.8.1 RoutedMessage: Container für die unbestimmte Weiterleitung von Nachrichten an einen unbekannten Empfänger

Route	**Message**
$4 \cdot x$ Byte	x Byte

Route Diese Feld enthält IDs der virtuellen Knoten, über die eine Route zum eigentlichen Ziel der Nachricht führt.

Message Der Parameter enthält die ursprüngliche Nachricht, die am Zielknoten ausgewertet wird und durch die mitgelieferten Kontaktinformationen direkt beantwortet werden kann.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc. (2001)

[2] Aberer K, Cudré-Mauroux P, Datta A, Despotovic Z, Hauswirth M, Punceva M and Schmidt R: "P-Grid: A Self-organizing Structured P2P System". *ACM SIGMOD Record* 32(3):29–33 (2003)

[3] Afework A, Beynon MD, Bustamante F, Cho S, Demarzo A, Ferreira R, Miller R, Silberman M, Saltz J, Sussman A and Tsang H: "Digital dynamic telepathology–the Virtual Microscope". *Proc AMIA Symp* pages 912–916 (1998)

[4] Ahmed R and Boutaba R: "Plexus: A Scalable Peer-to-peer Protocol Enabling Efficient Subset Search". *IEEE/ACM Trans Netw* 17(1):130–143 (2009)

[5] Akbarinia R, Pacitti E and Valduriez P: "Data Currency in Replicated DHTs". In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 211–222 (2007)

[6] Allman M, Paxson V and Stevens W: "TCP Congestion Control (RFC 2581)" (1999). URL `http://www.rfc-base.org/txt/rfc-2581.txt`

[7] Alpern S: "The Rendezvous Search Problem". *SIAM J Control Optim* 33(3):673–683 (1995)

[8] Aperio: "WebScope Demo Site". URL `http://images.aperio.com/BigTIFF/BreastCancer.tif/view.apml`

[9] Apostolopoulos JG, tian Tan W and Wee SJ: "Video Streaming: Concepts, Algorithms and Systems". Technical Report HPL-2002-260, Mobile and Media Systems Laborartory, HP Laboratories Palo Alto (2002)

[10] Banerjee S, Bhattacharjee B and Kommareddy C: "Scalable Application Layer Multicast". In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 205–217 (2002)

[11] Barbian A, Barbian B, Malenica D, Brzoska P and Filler T: "Omentum - eine verteilte, interdisziplinäre Lehr- und Lernplattform". In *Gemeinsame Jahrestagung der Gesellschaft für Medizinische Ausbildung (GMA) und des Arbeitskreises zur Weiterentwicklung der Lehre in der Zahnmedizin (AKWLZ). Leipzig, 30.09.-03.10.2015*, GMA, pages 135–136 (2015)

[12] Barbian A, Malenica D, Filler TJ and Schoettner M: "Omentum - A Peer-to-Peer Approach for Internet-Scale Virtual Microscopy". In *Proceedings of the 8th International Conference on Internet and Distributed Computed Systems*, IDCS, pages 273–284 (2015)

[13] Barbian A, Nothaas S, Filler TJ and Schoettner M: "A portable and platform independent file system for large scale peer-to-peer systems and distributed applications". In *Proceedings of the 16th International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT (2015)

[14] Baset SA and Schulzrinne H: "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol". Technical Report CUCS-039-04, Cornell University (2004)

[15] Battiti R: *Modern Heuristic Search Methods*, chapter Reactive search: Toward self-tuning heuristics. John Wiley and Sons Ltd. (1996)

[16] Best S: "JFS log: How the journaled file system performs logging". In *Proceedings of the 4th Annual Linux Showcase and Conference*, volume 4 of *ALS*, pages 163–168 (2000)

[17] Bharambe AR, Agrawal M and Seshan S: "Mercury: Supporting Scalable Multi-attribute Range Queries". In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 353–366 (2004)

[18] Blake CA, Lavoie HA and Millette CF: "Teaching Medical Histology at the University of South Carolina School of Medicine: Transition to Virtu-

al Slides and Virtual Microscopes". *Anat Rec (New Anat)* 275B:196–206 (2003)

[19] Blanchette J and Summerfield M: *C++ GUI Programming with Qt 4*. Prentice Hall PTR (2006)

[20] Bolot JC: "End-to-end Packet Delay and Loss Behavior in the Internet". *SIGCOMM Comput Commun Rev* 23(4):289–298 (1993)

[21] Bonwick J, Ahrens M, Henson V, Maybee M and Shellenbaum M: "The Zettabyte File System". Technical Report, SUN Microsystems (2003)

[22] Bradler D: *Peer-to-Peer Concepts for Emergency First Response*. Ph.D. thesis, Technical University Darmstadt, Computer Science (2010)

[23] Bradler D, Krumov L, Mühlhäuser M and Kangasharju J: "PathFinder: Efficient Lookups and Efficient Search in Peer-to-Peer Networks". Technical Report TU-CS-2010872, Technical University Darmstadt, Computer Science (2010)

[24] Bradler D, Krumov L, Mühlhäuser M and Kangasharju J: "PathFinder: Efficient Lookups and Efficient Search in Peer-to-Peer Networks". In *Proceedings of the 12th International Conference on Distributed Computing and Networking*, ICDCN, pages 77–82 (2011)

[25] Brakmo LS, O'Malley SW and Peterson LL: "TCP Vegas: New Techniques for Congestion Detection and Avoidance". *SIGCOMM Comput Commun Rev* 24(4):24–35 (1994)

[26] Braun MW and Kearns KD: "Improved Learning Efficiency and Increased Student Collaboration Through Use of Virtual Microscopy in the Teaching of Human Pathology". *Anat Sci Educ* 1:240–246 (2008)

[27] Brochhausen C, Winther H and Kirkpatrick C: "PATE - Virtuelle Mikroskopie Mainz". URL `http://pate.um-mainz.de/`

[28] Bronstejn I, Musiol G and Mühlig H: *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main (1999)

[29] Brown WE: "N3551: Random Number Generation in C++ 11" (2013)

[30] Brownrigg DRK: “The Weighted Median Filter”. *Comput Gr Image Process* 27(8):807–818 (1984)

[31] Buades A, Coll B and Morel JM: “A Review of Image Denoising Algorithms, with a New One”. *Multiscale Model Simul* 4(2):490–530 (2005)

[32] Bundesministerium der Justiz und für Verbraucherschutz: “Gesetze im Internet (Gesetz betreffend das Urheberrecht an Werken der bildenden Künste und der Photographie)”. URL `http://www.gesetze-im-internet.de/kunsturhg/`

[33] Bundesministerium der Justiz und für Verbraucherschutz: “Gesetze im Internet (Medizinproduktgesetz)”. URL `http://www.gesetze-im-internet.de/mpg/`

[34] Card R, Ts’o T and Tweedie S: “Design and Implementation of the Second Extended Filesystem”. In *Proceedings of the First Dutch International Symposium on Linux* (1994)

[35] Carter RL and Crovella ME: “Measuring Bottleneck Link Speed in Packet-switched Networks”. *Perform Eval* 27-28:297–318 (1996)

[36] Chawathe Y, Ratnasamy S, Breslau L, Lanham N and Shenker S: “Making Gnutella-like P2P Systems Scalable”. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 407–418 (2003)

[37] Chen X and Jarvis SA: “Design and implementation of efficient range query over DHT services”. In *IEEE International Conference on Signal Processing and Communications*, ICSPC, pages 571–579 (2008)

[38] Clip2: “The Gnutella Protocol Specification v0.4” (2000). URL `https://cryptnet.net/fsp/cpcd/gnutella_protocol_0.4.pdf`

[39] Coffey N: “How does java.util.Random work and how good is it?” (2013). URL `http://www.javamex.com/tutorials/random_numbers/java_util_random_algorithm.shtml#.VtwTKsfqEhk`

[40] Coleman R: “Can histology and pathology be taught without microscopes? The advantages and disadvantages of virtual histology”. *Acta histochem* 111:1–4 (2009)

[41] Combs G and et al: “Wireshark”. URL `http://www.wireshark.org`

[42] Company TQ: “Qt Documentation - Officially Supported Platforms” (2016). URL `http://doc.qt.io/QtSupportedPlatforms/index.html`

[43] Cooper D, Santesson S, Farrell S, Boeyen S, Housley R and Polk W: “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile” (2008). URL `http://www.rfc-base.org/txt/rfc-5280.txt`

[44] Crainiceanu A, Linga P, Machanavajjhala A, Gehrke J and Shanmugasundaram J: “P-ring: An Efficient and Robust P2P Range Index Structure”. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD, pages 223–234 (2007)

[45] Dang TNH, D DS and Sang DV: “A Denoising Method Based on Total Variation”. In *Proceedings of the Sixth International Symposium on Information and Communication Technology*, SoICT, pages 223–230 (2015)

[46] Datta A, Hauswirth M and Aberer K: “Updates in Highly Unreliable, Replicated Peer-to-Peer Systems”. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS, page 76 (2003)

[47] Deniz H and Cakir H: “Design Principles for Computer-assisted Instruction in Histology Education: An Exploratory Study”. *J Sci Educ Technol* 15(5):399–408 (2006)

[48] Department of Pathology, Arizona Health Service Science: “Ronald S. Weinstein” (2013). URL `http://pathology.arizona.edu/profile/ronald-s-weinstein-md-fcap-fata`

[49] Dierks T and Rescorla E: “The Transport Layer Security (TLS) Protocol Version 1.2” (2008). URL `http://www.rfc-base.org/txt/rfc-5246.txt`

[50] Dovrolis C, Ramanathan P and Moore D: “What do packet dispersion techniques measure?” In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2 of *INFOCOM*, pages 905–914. IEEE (2001)

[51] Drake RL, McBride JM, Lachman N and Pawlina W: “Medical Education in the Anatomical Sciences: The Winds of Change Continue to Blow”. *Anat Sci Educ* 2:253–259 (2009)

[52] Drebin RA, Carpenter L and Hanrahan P: "Volume Rendering". In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH, pages 65–74 (1988)

[53] Emule Developers: "Das Emule Projekt" (2002). URL `http://www.emule-project.net`

[54] Encyclopaedia Britannica: "Biography of Alan Mathison Turing". URL `http://www.britannica.com/biography/Alan-Turing`

[55] Encyclopaedia Britannica: "Biography of Carl Friedrich Gauss". URL `http://www.britannica.com/biography/Carl-Friedrich-Gauss`

[56] Encyclopaedia Britannica: "Biography of Charles E. Spearman". URL `http://www.britannica.com/biography/Charles-E-Spearman`

[57] Encyclopaedia Britannica: "Biography of Jakob Bernoulli". URL `http://www.britannica.com/biography/Jakob-Bernoulli`

[58] Encyclopaedia Britannica: "Biography of James Stirling". URL `http://www.britannica.com/biography/James-Stirling-British-mathematician`

[59] Encyclopaedia Britannica: "Biography of Siméon-Denis Poisson". URL `http://www.britannica.com/biography/Simeon-Denis-Poisson`

[60] Fellers TJ and Davidson MW: "CCD Noise Sources and Signal-to-Noise Ratio" (2015). URL `https://micro.magnet.fsu.edu/primer/digitalimaging/concepts/ccdsnr.html`

[61] Fernando R: *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Addison-Wesley Professional (2004)

[62] Ferreira R, Moon B, Humphries J, Sussman A, Saltz J, Miller R and Demarzo A: "The Virtual Microscope". *AMIA Annu Fall Symp* pages 449–453 (1997)

[63] Fogel DB: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, USA (1995)

[64] Fujita K and Crowley RS: "The Virtual Slide Set - a Curriculum Development System for Digital Microscopy". *AMIA Annu Symp Proc* page 846 (2003)

[65] Garfinkel S: *PGP: Pretty Good Privacy*. O'Reilly Media (1994)

[66] Ghemawat S, Gobioff H and Leung ST: "The Google File System". In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP, pages 29–43 (2003)

[67] Girdzijauskas S, Galuba W, Darlagiannis V, Datta A and Aberer K: "Fuzzynet: Ringless routing in a ring-like structured overlay". *Peer Peer Netw Appl* 4(3):259–273 (2010)

[68] Goldberg IG, Allan C, Burel JM, Creager D, Falconi A, Hochheiser H, Johnston J, Mellen J, Sorger PK and Swedlow JR: "The Open Microscopy Environment (OME) Data Model and XML file: open tools for informatics and quantitative analysis in biological imaging". *Genome Biol* 6(R47) (2005)

[69] Good NS and Krekelberg A: "Usability and Privacy: A Study of Kazaa P2P File-sharing". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI, pages 137–144 (2003)

[70] Görke W: *Handbuch der Informatik - Band 2.1 - Fehlertolerante Rechensysteme*. R. Oldenbourg Verlag (1989)

[71] Graffi K, Mukherjee P, Menges B, Hartung D, Kovacevic A and Steinmetz R: "Practical security in p2p-based social networks". In *Proceedings of the 34th annual IEEE conference on Local Computer Networks*, LCN (2009)

[72] Greenhill C, Holt FB and Wormald N: "Expansion Properties of a Random Regular Graph After Random Vertex Deletions". *Eur J Comb* 29(5):1139–1150 (2008)

[73] Grimes G, McClellan S, Goldman J, Vaughn G, Conner D, Kujawski E, McDonald J, Winokur T and Flemming W: "Applications of virtual reality technology in pathology". *Stud Health Technol Inform* 39:319–327 (1997)

[74] Harris T, Leaven T, Heidger P, Kreiter C, Duncan J and Dick F: "Comparison of a Virtual Microscope Laboratory to a Regular Microscope Laboratory for Teaching Histology". *Anat Rec (New Anat)* 265:10–14 (2001)

[75] Heidger Jr PM, Dee F, Leaven T, Duncan J and Kreiter C: "Integrated approach to teaching and testing in histology with real and virtual imaging". *Anat Rec (New Anat)* 269(2):107–112 (2002)

[76] Hildebrand R: “Contribution of Würzburg anatomy to the development of microscopic anatomy studies (1847 Kölliker–Petersen 1940)”. *Anat Anz* 155(1–5):115–122 (1984)

[77] Hilmer H: *Grundsätze und Lösungen fehlertoleranter Kommunikation in ereignisorientierten verteilten Echtzeitsystemen*. VDI Verlag (2002)

[78] Housley R, Polk W, Ford W and Solo D: “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile” (2002). URL `http://www.rfc-base.org/txt/rfc-3280.txt`

[79] Hu N and Steenkiste P: “Evaluation and Characterization of Available Bandwidth Probing Techniques”. *IEEE J Sel A Commun* 21(6):879–894 (2006)

[80] Information Science Institute, University of Southern California: “Internet Protocal (RFC 791)”. URL `http://www.rfc-base.org/txt/rfc-791.txt`

[81] Information Science Institute, University of Southern California: “Transmission Control Protocal (RFC 793)”. URL `http://www.rfc-base.org/txt/rfc-793.txt`

[82] Jacobson V: “Congestion Avoidance and Control”. *SIGCOMM Comput Commun Rev* 18(4):314–329 (1988)

[83] Jaegermann A, Filler TJ and Schoettner M: “Distributed Architecture for a Peer-to-Peer-based Virtual Microscope”. In *Distributed Applications and Interoperable Systems*, DAIS, pages 199–204 (2013)

[84] Jaegermann A, Filler TJ, Toddenroth D, Frankewitsch T, Missler M and Marschall B: “Histologie im 21. Jahrhundert - Tradition und Innovation”. In *Tagungsband der Jahrestagung der Gesellschaft für Medizinische Ausbildung*, GMA, pages 148–149 (2010)

[85] Jaegermann A, Zanger K, Rohbeck B and Filler T: “Virtual microscopy extended to electron microscopy”. In *Tagungsband der 28. Arbeitstagung der Anatomischen Gesellschaft*, page 62 (2011)

[86] Jain M and Dovrolis C: “End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput”. *SIGCOMM Comput Commun Rev* 32(4):295–308 (2002)

[87] Jain M and Dovrolis C: "Pathload: a measurement tool for end-to-end available bandwidth". In *Proceedings of the Workshop on Passive and Active Measurement*, PAM (2002)

[88] Jain R: "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks". *SIGCOMM Comput Commun Rev* 19(5):56–71 (1989)

[89] Janiuk J, Mäcker A and Graffi K: "Secure distributed data structures for peer-to-peer-based social networks". In *Proceedings of the 2014 International Conference on Collaboration Technologies and Systems*, CTS (2014)

[90] Jeong WK, Schneider J, Turney S, Faulkner-Jones B, Meyer D, Westermann R, Reid R, Lichtman J and Pfister H: "Interactive Histology of Large-Scale Biomedical Image Stacks". *Trans Vis Comput Graphics* 16(6):1386–1395 (2010)

[91] Jiang H and Dovrolis C: "Why is the Internet Traffic Bursty in Short Time Scales?" *SIGMETRICS Perform Eval Rev* 33(1):241–252 (2005)

[92] Keshav S: "A Control-theoretic Approach to Flow Control". *SIGCOMM Comput Commun Rev* 21(4):3–15 (1991)

[93] Kirkpatrick S, Gelatt CD and Vecchi MP: "Optimization by Simulated Annealing". *Science* 220(4598):671–680 (1983)

[94] Knuth DE: *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1997)

[95] Kokash N: "An introduction to heuristic algorithms". Online at ResearchGate (2016). URL `https://www.researchgate.net/publication/228573156_An_introduction_to_heuristic_algorithms`

[96] Köpf-Maier P and Merker HJ: *Atlas der Elektronenmikroskopie: Zellen, Gewebe, Organe*. Ueberreuter Wissenschaft, Wien; Berlin (1989)

[97] Korhonen J, Tschofenig H, Arumaithurai M, Jones M and Lior A: "Traffic Classification and Quality of Service (QoS) Attributes for Diameter" (2010). URL `http://www.rfc-base.org/txt/rfc-5777.txt`

[98] Krippendorf BB and Lough J: "Complete and Rapid Switch From Light Microscopy to Virtual Microscopy for Teaching Medical Histology". *Anat Rec (New Anat)* 285B:19–25 (2005)

[99] Laging H: "Einführung in die Funktionsweise von OpenPGP / GnuPG (gpg)" (2015). URL `http://www.hauke-laging.de/sicherheit/openpgp.html#wot`

[100] Lai K and Baker M: "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay". *SIGCOMM Comput Commun Rev* 30(4):283–294 (2000)

[101] Lai K and Baker M: "Nettimer: A Tool for Measuring Bottleneck Link, Bandwidth". In *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*, USITS, pages 11–11 (2001)

[102] Latham VA: "What Is the Best Method of Teaching Microscopical Science in Medical Schools". *T Am Microsc Soc* 18:311–320 (1897)

[103] Lawrence Berkeley National Laboratory: "iPerf - The network bandwidth measurement tool". URL `https://iperf.fr`

[104] Lee JS: "Digital Image Enhancement and Noise Filtering by Use of Local Statistics". *IEEE Trans Pattern Anal Mach Intell* 2(2):165–168 (1980)

[105] Lee KC and Mak LS: "Virtual Electron Microscopy: A Simple Implementation Creating a New Paradigm in Ultrastructural Examination". *Int J Surg Pathol* 19(5):570–575 (2011)

[106] Leland WE, Taqqu MS, Willinger W and Wilson DV: "On the Self-similar Nature of Ethernet Traffic". *SIGCOMM Comput Commun Rev* 23(4):183–193 (1993)

[107] Leng C: *BubbleStorm: Replikation, Updates und Konsistenz in Rendezvous-Informationssystemen*. Ph.D. thesis, Technical University Darmstadt, Computer Science (2012)

[108] Leng C, Terpstra WW, Kemme B, Stannat W and Buchmann AP: "Maintaining Replicas in Unstructured P2P Systems". In *Proceedings of the 4th*

ACM International Conference on emerging Networking EXperiments and Technologies, ACM CoNEXT (2008)

[109] Lewiner T: "VSVR: a very simple volume rendering implementation with 3D textures". Technical Report MAT. 16/06, Department of Mathematics, Pontifícia Universidade Católica, Rio de Janeiro, Brazil (2006)

[110] Lin T and Wang H: "Search performance analysis in peer-to-peer networks". In *Proceedings of the Third International Conference on Peer-to-Peer Computing*, P2P, pages 204–205 (2003)

[111] Linkert M, Rueden CT, Allan C, Burel JM, Moore W, Patterson A, Loranger B, Moore J, Neves C, MacDonald D, Tarkowska A, Sticco C, Hill E, Rossner M, Eliceiri KW and Swedlow JR: "Metadata matters: access to image data in the real world". *J Cell Biol* 189(5):777–782 (2010)

[112] Lorensen WE and Cline HE: "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH, pages 163–169 (1987)

[113] Lua EK, Crowcroft J, Pias M, Sharma R and Lim S: "A Survey and Comparison of Peer-to-peer Overlay Network Schemes". *Commun Surveys Tuts* 7(2):72–93 (2005)

[114] Mahy R, Matthews P and Rosenberg J: "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)" (2010). URL `http://www.rfc-base.org/txt/rfc-5766.txt`

[115] Malenica D: *Development of a fault-tolerant service-service-communication in the educational context of virtual microscopy.* Bachelor thesis, Heinrich-Heine-University Duesseldorf (2012)

[116] Malenica D: *Distributed authentication in Peer-to-Peer networks on the basis of Omentum.* Master thesis, Heinrich-Heine-University Duesseldorf (2016)

[117] Mathur A, Cao M, Bhattacharya S, Dilger A, Tomas A and Vivier L: "The new ext4 filesystem: current status and future plans". In *Proceedings of the Linux Symposium*, volume 2, pages 21–33. Ottawa, Ontario, Canada (2007)

[118] Maymounkov P and Mazières D: "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS, pages 53–65 (2002)

[119] Mazières D: "A toolkit for user-level file systems". In *Proceedings of the 2001 USENIX Annual Technical Conference*. Boston, Massachusetts, USA (2001)

[120] Mell P and Grance T: "The NIST Definition of Cloud Computing". Special Publication 800-145, National Institute of Standards and Technology (2011)

[121] Moch H and Aguzzi A: "Histologiekurs Departement Pathologie der Universität Zürich". URL `http://www.pathol.uzh.ch/histologiekurs/`

[122] Navratil J and Cottrell RL: "ABwE: A Practical Approach to Available Bandwidth Estimation". In *Proceedings of the Workshop on Passive and Active Measurement*, PAM (2003)

[123] Network Associates, Inc: *An Introduction to Cryptography*. Network Associates, Inc. (1999)

[124] Nicolae B, Antoniu G and Bougé L: "BlobSeer: How to Enable Efficient Versioning for Large Object Storage Under Heavy Access Concurrency". In *Proceedings of the 2009 EDBT/ICDT Workshops*, EDBT/ICDT, pages 18–25 (2009)

[125] Nothaas S: *USP-FS: A portable user-space file-system for many files*. Bachelor thesis, Heinrich-Heine-University Duesseldorf (2013)

[126] Nothaas S: *Development of a concept for distributed storage of complex 3D models*. Master thesis, Heinrich-Heine-University Duesseldorf (2015)

[127] NYU School of Medicine: "NYU Virtual Microscope (NYUVM)". URL `http://education.med.nyu.edu/virtualmicroscope`

[128] OsiriX Advanced Imaging: "Dicom sample image sets: Surgical repair of facial deformity (PHENIX)". URL `http://www.osirix-viewer.com/datasets/DATA/PHENIX.zip`

[129] Paul RG: *Development of a mobile and cross platform peer to peer client in context of virtual microscopy*. Master thesis, Heinrich-Heine-University Duesseldorf (2016)

[130] Postel J: "User Datagram Protocol (RFC 768)" (1980). URL `http://www.rfc-base.org/txt/rfc-768.txt`

[131] Pouwelse J, Garbacki P, Epema D and Sips H: "The Bittorrent P2P File-sharing System: Measurements and Analysis". In *Proceedings of the 4th International Conference on Peer-to-Peer Systems*, IPTPS, pages 205–216 (2005)

[132] Pratt RL: "Are We Throwing Histology Out With the Microscope? A Look at Histology From the Physician's Perspective". *Anat Sci Educ* 2:205–209 (2009)

[133] Rajgarhia A and Gehani A: "Performance and Extension of User Space File Systems". In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC, pages 206–213 (2010)

[134] Ratnasamy S, Francis P, Handley M, Karp R and Shenker S: "A Scalable Content-addressable Network". In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 161–172 (2001)

[135] Razi T, Miknami M and Ghazani FA: "Relationship between Hounsfield Unit in CT Scan and Gray Scale in CBCT". *J Dent Res Dent Clin Dent Prospects* 8(2):107–110 (2014)

[136] Reynolds P and Vahdat A: "Efficient Peer-to-peer Keyword Searching". In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, MIDDLEWARE, pages 21–40 (2003)

[137] Ribeiro VJ, Navratil J, Riedi RH, Baraniuk RG and Cottrell L: "pathchirp: Efficient available bandwidth estimation for network paths". In *Proceedings of the Workshop on Passive and Active Measurement*, number SLAC-PUB-9732 in PAM (2003)

[138] Rojo MG, García GB, Mateos CP, García JG and Vincente MC: "Critical comparison of 31 commercially available digital slide systems in pathology". *Int J Surg Pathol* 14(4):285–305 (2006)

[139] Rosenberg J: "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols".

Internet Requests for Comments (2010). URL http://www.rfc-base.org/txt/rfc-5245.txt

[140] Rosenberg J, Mahy R, Matthews P and Wing D: "Session Traversal Utilities for NAT (RFC 5389)" (2008). URL http://www.rfc-base.org/txt/rfc-5389.txt

[141] Rowstron A and Druschel P: "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350 (2001)

[142] Rowstron A and Druschel P: "Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility". In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP, pages 188–201 (2001)

[143] Sandberg R, Golgberg D, Kleiman S, Walsh D and Lyon B: *Innovations in Internetworking*, chapter Design and Implementation of the Sun Network Filesystem, pages 379–390. Artech House, Inc. (1988)

[144] Sander B and Golas MM: "HistoViewer: An interactive e-learning platform facilitating group and peer group learning". *Anat Sci Educ* 6(3):182–190 (2013)

[145] Saroiu S, Gummadi PK and Gribble SD: "A Measurement Study of Peer-to-Peer File Sharing Systems". In *Proceedings of the 2002 SPIE Multimedia Computing and Networking Conference*, volume 4673 of *MMCN* (2002)

[146] Satyanarayanan M, Howard JH, Nichols DA, Sidebotham RN, Spector AZ and West MJ: "The ITC Distributed File System: Principles and Design". In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, SOSP, pages 35–50 (1985)

[147] Schneider FB: "Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial". *ACM Comput Surv* 22(4):299–319 (1990)

[148] Sheffer Y, Holz R and Saint-Andre P: "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)" (2015). URL http://www.rfc-base.org/txt/rfc-7457.txt

[149] Shooman ML: *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*. Wiley-Interscience (2001)

[150] Shvachko K, Kuang H, Radia S and Chansler R: "The Hadoop Distributed File System". In *IEEE 26th Symposium on Mass Storage Systems and Technologies*, MSST, pages 1–10 (2010)

[151] Srisuresh P and Egevang K: "Traditional IP Network Address Translator (RFC 3022)" (2001). URL `http://www.rfc-base.org/txt/rfc-3022.txt`

[152] Steiner JG, Neuman C and Schiller JI: "Kerberos: An Authentication Service for Open Network Systems". In *Proceedings of the 1988 USENIX Conference*, pages 18–28 (1988)

[153] Stoica I, Morris R, Karger D, Kaashoek MF and Balakrishnan H: "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM, pages 149–160 (2001)

[154] Strauss J, Katabi D and Kaashoek F: "A Measurement Study of Available Bandwidth Estimation Tools". In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC, pages 39–44 (2003)

[155] Sun X and Chen X: "SRing: A Structured Non Dht P2P Overlay Supporting String Range Queries". In *Proceedings of the 16th International Conference on World Wide Web*, WWW, pages 1193–1194 (2007)

[156] Swan JEI and Yagel R: "Slide-Based Volume Rendering". Technical Report OSU-ACCAD-1/93-TR1, Ohio State University (1993)

[157] Swedlow J, Allan C, Burel JM, Linkert M, Loranger B, MacDonald D, Moore W, Patterson A, Rueden C, Tarkowska A, Goldberg I and Eliceiri K: "The Open Microscopy Environment: Informatics and Quantitative Analysis for Biological Microscopy". *Microsc Microanal* 15:1520–1521 (2009)

[158] Szeredi M: "File Systems in Userspace". URL `http://fuse.sourceforge.net`

[159] Takeda A, Chacraborty D, Kitagata G, Hashimoto K and Shiratori N: "A New Scalable Distributed Authentication for P2P Network and Its Performance Evaluation". *Comp* 7(10):1628–1637 (2008)

[160] Takeda A, Kitagata G, Kinoshita T, Hashimoto K, Zabir S and Shiratori N: "A new authentication method with distributed hash table for P2P network". In *22nd International Conference on Advanced Information Networking and Applications - Workshops*, AINAW, pages 483–488. IEEE (2008)

[161] Tanenbaum AS and Steen MV: *Distributed Systems - Principles and Paradigms*. Pearson Education Inc., 2nd edition edition (2007)

[162] Terpstra WW: *BubbleStorm: Rendezvous Theory in Unstructured Peer-to-Peer Search*. Ph.d. thesis, Technical University Darmstadt, Computer Science (2015)

[163] Terpstra WW, Kangasharju J, Leng C and Buchmann AP: "BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search". In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM. Kyoto, Japan (2007)

[164] Terpstra WW, Leng C and Buchmann AP: "BubbleStorm: Analysis of Probabilistic Exhaustive Search in a Heterogeneous Peer-to-Peer System". Technical Report TUD-CS-2007-2, Technical University Darmstadt, Computer (2007)

[165] The Shibboleth Consortium: "Shibboleth". URL `http://shibboleth.net/`

[166] University of Dundee & Open Microscopy Environment: "The Bio-Formats Library". URL `http://www.openmicroscopy.org/site/products/bio-formats`

[167] Wang CW, Huang CT and Hung CM: "VirtualMicroscopy: ultra-fast interactive microscopy of gigapixel/terapixel images over internet". *Sci Rep* 5 (2015)

[168] Wang Z, Das SK, Kumar M and Shen H: "An Efficient Update Propagation Algorithm for P2P Systems". *Comput Commun* 30(5):1106–1115 (2007)

[169] Weinhold D: *Classification of network nodes in distributed systems by measuring bandwitdh with user data*. Master thesis, Heinrich-Heine-University Duesseldorf (2014)

[170] Weinstein RS: “Prospects for telepathology”. *Hum Pathol* 17(5):433–434 (1986)

[171] Windoffer R and Leube R: “Mainz Histo Maps”. URL `http://www.mhm.uni-mainz.de/`

[172] Wyvill G, McPheeters C and Wyvill B: “Data structure for soft objects”. *Visual Comput* 2:227–234 (1986)

[173] Yang KH and Ho JM: “Proof: A DHT-Based Peer-to-Peer Search Engine”. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI, pages 702–708 (2006)

[174] Zadeh LA: “Fuzzy Sets”. *Inf Control* 8:338–353 (1965)

[175] Zhang X, Liu J, Li B and shing Peter Yum T: “CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming”. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM (2005)

[176] Zhao BY, Kubiatowicz J and Joseph AD: “Tapestry: An infrastructure for fault-tolerant wide-area location and routing”. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California (2001)

[177] Zhuang L and Zhou F: “Understanding Chord Performance”. Technical Report CS268, UC Berkley (2003)

Eidesstattliche Erklärung

An Eidesstatt erkläre ich, dass ich diese Dissertation selbstständig und ohne unzulässige fremde Hilfe unter der Beachtung der „Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Heinrich-Heine-Universität“ angefertigt habe und dass ich diese Arbeit weder in der jetzigen noch in ähnlicher Form an dieser oder einer anderen Fakultät eingereicht habe.

Mettmann, 04. November 2016 Andreas Barbian