Protokoll-agnostische Netzwerkdienste und netzunterstützte Datentransportaufwertung durch den Einsatz von Zwischensystemen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

Dissertation

von

Helge Backhaus
aus Bremen

Tag der mündlichen Prüfung:	15. 07. 2016
Erste Gutachterin:	Prof. Dr. Martina Zitterbart
	Karlsruher Institut für Technologie (KIT)
Zweiter Gutachter:	Prof. Dr. Michael Welzl
	Universitetet i Oslo

Danksagung

Diese Arbeit entstand während meiner Anstellung als wissenschaftlicher Mitarbeiter am Institut für Telematik am Karlsruher Institut für Technologie (KIT). An erster Stelle gilt mein Dank meiner Chefin und Referentin dieser Arbeit, Frau Prof. Dr. Martina Zitterbart, für die Anstellung am Institut und die Betreuung dieser Arbeit. Während meiner gesamten Institutszeit stand sie mir unterstützend mit vielen hilfreichen Diskussionen und Ratschlägen zur Seite. Weiterhin bedanken möchte ich mich bei Herrn Prof. Dr. Michael Welzl, der sich trotz zahlreicher Verpflichtungen in Forschung und Lehre sofort bereit erklärte, das Koreferat für meine Promotion zu übernehmen und die weite Anreise aus Oslo zu meiner mündlichen Prüfung auf sich nahm.
Ich bedanke mich außerdem für die enge thematische Zusammenarbeit und hilfreichen Diskussionen mit Denis Martin und Hans Wippel. Besonderer Dank gilt auch Sören Finster, der sich lange Zeit mit mir ein Büro teilte und immer eine offenes Ohr für mich hatte. Fabian Hartmann, Martin Florian, Matthias Flittner und Robert Bauer gebührt ebenfalls Dank für ihre konstante Unterstützung, Zuverlässigkeit und angenehme Zusammenarbeit.
Nicht zuletzt bedanke ich mich bei allen weiteren Kollegen die ich in meiner Zeit am Institut kennenlernen durfte. Jeder einzelne hat auf seine Weise dazu beigetragen, dass ich mich die letzten Jahre sehr wohl gefühlt habe. Vielen Dank euch allen!

Helge Backhaus, im März 2017

Inhaltsverzeichnis

Abbildungsverzeichnis **ix**

Tabellenverzeichnis **xiii**

Auflistungsverzeichnis **xv**

1 Einleitung und Motivation **1**
1.1 Problemstellung 3
1.2 Anforderungen 4
1.3 Zielsetzung 5

2 Grundlagen **7**
2.1 Verteilte Anwendungen 7
2.2 Das Schichtenmodell des Internets 9
2.2.1 Das ISO/OSI Referenzmodell 9
2.2.2 Das DoD Schichtenmodell & das TCP/IP Referenzmodell 13
2.3 Begriffe und Definitionen 16

3 Die PASTE-API: Eine Anwendungsschnittstelle für zukünftige Netze **21**
3.1 Die Anwendungsschnittstelle heute 23
3.1.1 Probleme der Berkeley Socket-API 25
3.1.2 HTTP als neue Anwendungsschnittstelle 26
3.2 Die PASTE-API 28
3.2.1 Anforderungen 29
3.3 Dienstprimitive der PASTE-API 30
3.3.1 HANDLE Methoden 37
3.3.2 Umsetzung von Kommunikationsmustern 40
3.4 Namensschemata 42
3.4.1 Aufbau 42
3.4.2 Verwendung innerhalb der PASTE-API 43
3.5 Transportdiensteigenschaften und -beschreibungen 45
3.5.1 Transportdiensteigenschaften 46
3.5.2 Transportdienstbeschreibung 53
3.5.3 Hierarchische Transportdienstwahl & -beschreibung 57
3.6 Anwendungsfälle 61
3.6.1 Chat-Server mit zwei Chat-Clients 63

3.6.2 Online Daten Backupdienst 66
3.6.3 Video-On-Demand Streamingdienst 68
3.6.4 Publish-Subscribe Dienst 69
3.7 Zusammenfassung 71

4 Das PASTE Rahmenwerk 73
4.1 Protocol Agnostic Services & Transport Enrichment 77
4.2 PASTE Komponenten 81
4.3 PASTE Module & Objekte 85
4.4 Datenpfade im PASTE Rahmenwerk 86
4.5 Assoziationsmanagement 89
4.5.1 Anwendungsdienste anbieten 90
4.5.2 Assoziationszustandsobjekte 92
4.5.3 Aufbau von Kommunikationsassoziationen 95
4.5.4 Verkürzter Aufbau von Kommunikationsassoziationen 97
4.5.5 Namensauflösung & Protokollselektion beim Kommunikationsassoziationsaufbau 99
4.5.6 Der PASTE Rahmen und das GoodCompany Protokoll 100
4.5.7 Abbau von Kommunikationsassoziationen 103
4.5.8 Pausieren & Wiederaufnehmen von Kommunikationsassoziationen 104
4.6 Namensauflösung & Transportprotokollwahl 107
4.7 Companion Einsatz 109
4.7.1 Companiondienstmanager 112
4.7.2 Companionsuche & Companioncache 115
4.8 Aktivitätszustand 116
4.9 Companion Anwendungsfall 119
4.10 Zusammenfassung 123

5 Evaluation und Implementierung 125
5.1 Anwendbarkeit der PASTE-API 125
5.1.1 Content-Centric Networking (CCN) 126
5.1.2 Message Queue Telemetry Transport (MQTT): Publish/Subscribe 127
5.1.3 Video-on-Demand Dienst 129
5.1.4 Vergleich zwischen PASTE-API und Berkeley SocketAPI 130
5.2 Transportdienstbeschreibungen 137
5.3 Evaluation des PASTE Prototypen 141
5.3.1 Definition gemessener Größen 142
5.3.2 Szenario 1: PASTE Rahmenwerk mit Cache Companion 143
5.3.3 Szenario 2: PASTE Rahmenwerk mit Compress Companion . . . 147

5.3.4 Szenario 3: PASTE Rahmenwerk mit Cache & Compress Companion 150
5.4 Zusammenfassung 153

6 Zusammenfassung **155**
6.1 Ausblick und weiterführende Arbeiten 156

Glossar **159**

Akronyme **163**

Literatur **165**

Abbildungsverzeichnis

1.1 Stundenglas-Modell des Internets. Innovationen am unteren und oberen Ende; Stagnation im Kern. 2

2.1 Kommunikationsbeziehung zwischen zwei Anwendungsprozessen. . . 7
2.2 Client-Server und Peer-to-Peer Kommunikationsmuster. 8
2.3 Das ISO-OSI 7 Schichten Referenzmodell. 10
2.4 Das DoD bzw. TCP/IP Referenzmodell im Vergleich zum ISO/OSI Modell. 14
2.5 Der Datenpfad über verschiedene Systeme und Übertragungsmedien hinweg. 16

3.1 Stundenglas Modell des Internets für die Berkeley Socket-API (links), HTTP (mitte) und PASTE-API (rechts). 29
3.2 Dienstprimitive der PASTE-API zur HANDLE Erzeugung. Aufgeteilt in Client- und Server-seitige Befehle. 32
3.3 Mit einem HANDLE assoziierte Warteschlangen und die darauf lesend und schreibend operierenden HANDLE-Methoden. 33
3.4 Dienstprimitive der PASTE-API zur Client-seitigen Abfrage von Dienstinformationen, sowie der Abfrage von Sitzungsinformationen. 35
3.5 Sitzungsidentifikatoren und Assoziationsidentifikatoren der PASTE-API beziehungsweise des PASTE Rahmenwerkes. 37
3.6 Transportdiensteigenschaften und -beschreibungen unterschiedlicher Granularität auf verschiedenen Stufen. 58
3.7 Chat-Server S mit zwei Chat-Clients A & B. 64
3.8 Chat-Server S mit Logging Client L & Advertisement Client V. 65
3.9 Online Daten Backupdienst. 67
3.10 Video-on-Demand Streamingdienst. 68
3.11 Publish-Subscribe Dienst mit Broker B und Subscribern X, Y & Z. 70

4.1 PASTE Komponenten im Netzwerk. 78
4.2 Eine Kommunikationsassoziation zwischen zwei Anwendungsinstanzen mit *Companion* Unterstützung. 80
4.3 Aufbau und Komponenten des PASTE Rahmenwerks. 82
4.4 Datenpfade und GoodCompany Signalisierung im PASTE Rahmenwerk. 87
4.5 Binden einer Anwendungsinstanz an eine URI. 91
4.6 Pro Assoziationsobjekt von PASTE verwalteter Zustand. 93
4.7 Aufbau einer Kommunikationsassoziation zwischen zwei Endsystemen. 96

4.8 Verkürzter Aufbau einer Kommunikationsassoziation zwischen zwei Endsystemen, mittels Transportdienstidentifikator. 98
4.9 Lokaler Ablauf des Kommunikationsassoziationsaufbaus. 99
4.10 Aufbau des PASTE Rahmens für Nutz- beziehungsweise Signalisierungsdaten. 101
4.11 Abbau einer Kommunikationsassoziation. 103
4.12 Pausieren einer laufenden Kommunikationsassoziation. 105
4.13 Wiederaufnehmen einer pausierten Kommunikationsassoziation. 106
4.14 Teilpfade und Warteschlangen einer Kommunikationsassoziation beim Einsatz von *Companions*. 112
4.15 Registrieren eines Companiondienstes beim Companionverzeichnis. . . 113
4.16 Companiondienst Arten: Auf einem Knoten oder auf einem Pfad. 114
4.17 Zustandsautomat des Aktivitätszustands für eine per BIND erzeugte Assoziation. 117
4.18 Zustandsautomat des Aktivitätszustands für eine per GET, PUT, CONNECT, ACCEPT oder RESUME erzeugte beziehungsweise wiederaufgenommene Assoziation. 118
4.19 Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung. 120
4.20 Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung. 121
4.21 Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung. 122
4.22 Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung. 122
4.23 Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung. 123

5.1 Setup des Video-on-Demand Demonstrators. 130
5.2 Benutzerschnittstelle des Video-On-Demand Demonstrators im Webbrowser. (Quelle & Copyright der Bilder und Videos: Blender Foundation, www.blender.org) 135
5.3 Evaluationsszenario: Berkeley Socket-API Aufbau. 142
5.4 Evaluationsszenario: Datenzwischenspeicher Aufbau. 144
5.5 Datenempfangsrate & -dauer mit Datenzwischenspeicher. 145
5.6 Datensenderate & -dauer mit Datenzwischenspeicher. 146
5.7 Evaluationsszenario: Datenkompression Aufbau. 148
5.8 Datenempfangsrate & -dauer mit Datenkompression. 149
5.9 Datensenderate & -dauer mit Datenkompression. 149

5.10 Evaluationsszenario: Datenzwischenspeicher und -kompression Aufbau. 150
5.11 Datensenderate/-dauer mit Datenzwischenspeicher & -kompression. . 151
5.12 Datenempfangsrate/-dauer mit Datenzwischenspeicher & -kompression. 152

Tabellenverzeichnis

3.1 Überblick über sämtliche Dienstprimitive der PASTE API. 31
3.2 Überblick sämtlicher *HANDLE* Methoden. 38
3.3 Überblick sämtlicher Parameter Typen. 41

5.1 Größe der Beispiel Transportdienstbeschreibung in Byte im Vergleich. . 141
5.2 Gemessene Werte für das Szenario: PASTE Rahmenwerk mit Cache Companion 146
5.3 Gemessene Werte für das Szenario: PASTE Rahmenwerk mit Compress Companion 150
5.4 Gemessene Werte für das Szenario: PASTE Rahmenwerk mit Cache & Compress Companion 152

Auflistungsverzeichnis

1 Aufbaue eines Parameterbeschreibungsobjektes in JSON Notation. . . . 40
2 Aufbau des Transportdiensteigenschaften Katalogs. 47
3 Transportdiensteigenschaft vom Typ `set` bzw. `list`. 48
4 Transportdiensteigenschaft vom Typ `container`. 49
5 Auszug des Transportdiensteigenschaften Katalogs. 50
6 Beispiel `container` für die Transportdiensteigenschaft *Zuverlässigkeit*. 51
7 Beispiele für zeitabhängige Transportdiensteigenschaften. 51
8 Beispiele für Datenraten abhängige Transportdiensteigenschaften. . . . 52
9 Beispiele für Größen abhängige Transportdiensteigenschaften. 52
10 Beispiele für eine Transportdiensteigenschaft vom Typ *set*. 52
11 Transportdiensteigenschaften vom Typ *set* und *list*. 53
12 Aufbau einer Transportdienstbeschreibung. 55
13 Eine Transportdienstbeschreibung für einen partiell zuverlässigen Dienst. 56
14 Aufbau einer Transportdienstschablone. 59
15 Eine Transportdienstbeschreibung mit Transportdienstschablone. . . . 60
16 Die Transportdienstschablone für partielle Zuverlässigkeit. 61
17 Die Transportdienstbeschreibung für die partielle Zuverlässigkeit Transportdienstschablone. 62
18 Aufbau eines Transportdienstidentifikators. 62
19 Beispiel für eine Transportdienstdefinition. 63
20 Eine Transportdienstbeschreibung über einen Transportdienstidentifikator. 63

21 Dienstbeschreibung für partielle Zuverlässigkeit. 138
22 Dienstbeschreibung für partielle Zuverlässigkeit reduziert auf relevante Informationen. 139
23 Dienstbeschreibung für partielle Zuverlässigkeit in minimierter Form. . 140
24 Generische Transportdienstbeschreibung und Abschätzung einer durchschnittlichen Transportdiensteigenschaft. 140

Kapitel 1

Einleitung und Motivation

Das Internet ist unbestreitbar eine der wichtigsten Erfindungen in der Geschichte der Kommunikations- und Informationstechnologien und hat im 20. Jahrhundert mit Diensten wie *E-Mail* und vor allem dem *World Wide Web* die Gesellschaft und die Art und Weise wie wir kommunizieren nachhaltig geprägt. Heute kommt kaum noch eine Anwendung ohne Onlinekomponente aus: Sofern sie nicht selbst Teil oder Erbringer eines vernetzten Dienstes ist, nutzen die meisten Anwendungen heute verschiedenste Netzwerkdienste. Beispielsweise für automatische Updates oder das Speichern und Synchronisieren von Daten und Einstellungen auf Onlinespeichern beziehungsweise in der *Cloud*. Häufig werden Anwendungen auch nicht mehr nur auf einem, sondern parallel auf mehreren Geräten genutzt. Zur Unterstützung eines nahtlosen Wechsels zwischen Anwendungsinstanzen, beispielsweise vom Smartphone zu einem Desktop PC, wird ebenfalls die Unterstützung des Netzwerks benötigt, um den Anwendungszustand über verschiedene Instanzen hinweg zu synchronisieren.

Seit seiner Entstehung verzeichnet das Internet einen kontinuierlichen Anstieg seiner Nutzer. Mit der Anzahl der Nutzer ist auch die Komplexität des Internets kontinuierlich gewachsen. Nicht nur die Anzahl der Internetanschlüsse wächst beständig, sondern auch die Anzahl der mit dem Internet verbundenen Geräte pro Nutzer beziehungsweise Haushalte, hat stetig zugenommen. Neue Geräte, Anwendungen und Dienste bedingen dabei sich stetig ändernde Anforderungen an das Netzwerk und einen gestiegenen Bedarf an Ressourcen wie Rechenleistung, Speicherplatz und Bandbreite. Um diesen Anforderungen gerecht zu werden, wird eine flexible und gut skalierende Architektur für das Internet benötigt. Diese setzt seit jeher auf eine Trennung unterschiedlicher Funktionalitäten in verschiedene Schichten, welche jeweils klar definierte Aufgaben haben. Wohldefinierte Schnittstellen zwischen den Schichten machen die einzelnen Schichten theoretisch unabhängig voneinander. Diese Trennung ist auch heute noch

gültig und wird weiterhin aufrechterhalten. Einzelne Schichten können so erweitert oder im optimalen Fall ganz ausgetauscht werden, ohne dass dies eine Anpassung der anderen Schichten erfordert.

Dass dieses Modell funktioniert, zeigt sich an der Vielzahl an heute existierenden Diensten beziehungsweise Anwendungen, welche über das Internet bereitgestellt werden. Ebenso an zahlreichen Innovationen auf den unteren Schichten. Die maximal erreichbare Datenübertragungsrate ist durch neue *Ethernet* Standards stetig gewachsen und verschiedene kabellose Datenübertragungstechnologien wie *Wireless Lan* oder *Bluetooth* sind im Laufe der Jahre hinzugekommen, ohne große Änderungen an den Schnittstellen zwischen den unteren Schichten nach sich zu ziehen.

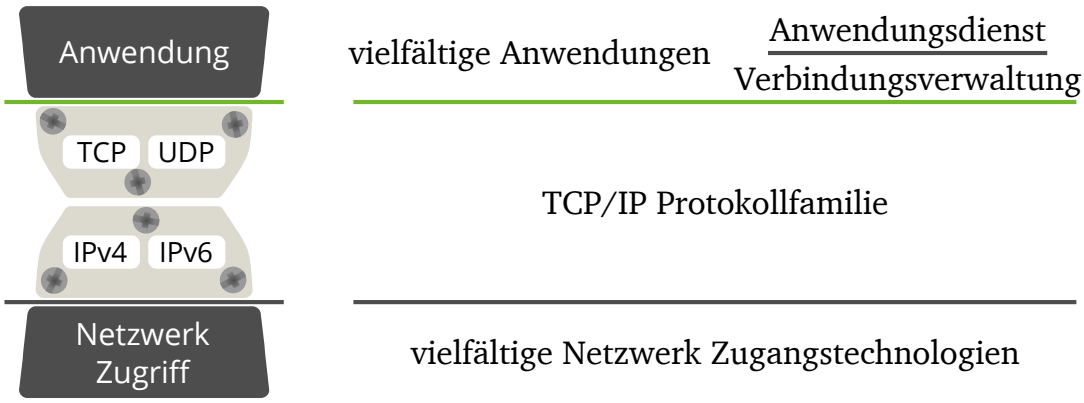


Abbildung 1.1: *Stundenglas-Modell des Internets. Innovationen am unteren und oberen Ende; Stagnation im Kern.*

Dabei hat sich allerdings ein starrer Mittelteil bestehend aus der Transport- und Vermittlungsschicht gebildet. Dieser besteht aus der sogenannten TCP/IP Protokollfamilie, welche als gemeinsame Basis für die verschiedenen Übertragungstechniken und Protokolle der Anwendungsschicht fungiert. Man spricht in diesem Zusammenhang auch vom Stundenglas-Modell des Internets, welches in Abbildung 1.1 zu sehen ist. Aufbauend auf dem Internet Protocol (IP) werden per Transmission Control Protocol (TCP) und User Datagram Protocol (UDP) grundsätzlich nur zwei verschiedene Transportdienste angeboten. Im Falle von TCP ein zuverlässiger verbindungsorientierter Dienst und im Falle von UDP ein unzuverlässiger verbindungsloser Dienst. Beiden liegt die Annahme zu Grunde, dass Daten vornehmlich zwischen zwei fixen Endpunkten ausgetauscht werden; ohne Einfluss auf Dienstgüteparameter wie eine möglichst geringe Verzögerung oder eine möglichst hohe Datenübertragungsrate nehmen zu können.

In der Praxis existieren heute viele Daten, deren Ursprung oder Ziel nicht mehr einem einzelnen festen Ort zugewiesen werden kann. Viele Geräte sind heute mobil. Einzelne Inhalte lassen sich häufig von unterschiedlichen Quellen beziehen und werden von vielen

Empfängern parallel konsumiert. Daraus resultiert ein Bedarf an alternativen Transportdiensten und die einfache Umsetzung neuer Kommunikationsmuster über die Eins-zu-Eins Kommunikation hinaus. Beispielsweise eine Gruppenkommunikation mit gleichzeitiger Angabe der Präferenz bestimmter Dienstgüteparameter durch die Anwendung.

Alternativen in Form von Transportprotokollen wie dem Stream Control Transmission Protocol (SCTP), Datagram Congestion Control Protocol (DCCP), Lightweight User Datagram Protocol (UDP-Lite) und Low Extra Delay Background Transport (LEDBAT) sind ebenso vorhanden, wie TCP-Erweiterungen wie MultiPath TCP (MPTCP) oder Data Center TCP (DCTCP). Mit Named Data Networking (NDN) beziehungsweise Content Centric Networking (CCN) existieren sogar Vorschläge für eine alternative Vermittlungsschicht um IP abzulösen. Da Anwendungen aber seit jeher die gleichen zwei Protokolle nutzen, sind neuere Alternativen in der Regel nicht flächendeckend verfügbar und aufgrund der schlechten Verfügbarkeit, setzen Anwendungsentwickler auch weiterhin auf bewährte und verfügbare Protokolle.

Dies führt unter anderem dazu, dass sich Anwendungen heute ebenfalls wie in Abbildung 1.1 dargestellt in zwei Schichten unterteilen lassen. Zunächst den grundsätzlichen Anwendungsdienst, den eine Anwendung erbringen soll und für dessen Umsetzung die Anwendung entworfen wurde. Dieser Teil einer Anwendung ist individuell auf ein zu lösendes Problem oder eine zu erledigende Aufgabe zugeschnitten und ihr Hauptzweck. Der individuelle Anwendungsdienst baut auf einem generischen Verbindungsmanagement auf. In diesem Teil finden sich Mechanismen für häufig auftretende netzwerkspezifische Transportaufgaben. Darunter fällt beispielsweise die Namensauflösung oder der Auf- und Abbau von Transportverbindungen. Dabei handelt es sich um wiederkehrende Probleme vernetzter Anwendungen, die individuell von Anwendungsentwicklern gelöst werden. Eine bessere Lösung wäre eine breitere Unterstützung gängiger Anwendungsanforderungen vom Betriebssystem oder dem Netzwerk selbst.

1.1 Problemstellung

Statt einer engen Taille ist auf Höhe der Transport- und Vermittlungsschicht eine Vielfalt und Flexibilität wünschenswert, wie sie sich in den darunter und darüber liegenden Schichten findet. Zum einen wird dies durch die heutige *Socket* Schnittstelle als de-facto Netzwerkabstraktions und -zugangspunkt Standard verhindert. Diese abstrahiert nicht genug von der unterliegenden Schicht und zwingt Anwendungen sich bereits beim Zugriff auf das Netzwerk auf ein konkretes Transportprotokoll festzulegen.

Zum anderen führt eine zunehmende Integration verschiedener *Middleboxes* in der Vermittlungsschicht zu einem Zustand, der die unkomplizierte Kommunikation von Endgeräten untereinander kontinuierlich erschwert. Bedingt durch einen Mangel an integrierter

Sicherheit finden sich heute in nahezu jedem Subnetz *Network Address Translators* (NAT) und *Firewalls*, die zunächst überwunden werden müssen. Der Wunsch nach einer einfachen Lösung dieses Problems, hat mit der Verbreitung des *World Wide Web* und der allgegenwärtigen Verfügbarkeit von *Webbrowsern* im Laufe der Jahre zur Etablierung des *Hypertext Transfer Protocols* (HTTP) als Quasi-Nachfolger der *Socket* Schnittstelle geführt; auf Kosten eines deutlich gestiegenen Kommunikationsmehraufwands und gestiegenen Verarbeitungskosten pro Verbindungsanfrage beziehungsweise übertragener Dateneinheit.

Durch die allgegenwärtige Verfügbarkeit beider Schnittstellen ist es schwierig geworden neue Protokolle auf der Transport- und Vermittlungsschicht zu etablieren. Internet Service Provider sehen keinen Anreiz, die Verbreitung alternativer Protokolle voranzubringen, solange kaum Anwendungen existieren, welche diese nutzen. Dies führt zu einem Henne-Ei-Problem, da neuere Protokolle damit für einen Anwendungsentwickler unattraktiv werden. Er kann sich nicht darauf verlassen, dass diese überall zur Verfügung stehen. Somit fällt ihm die Aufgabe zu immer auch eine Rückfalllösung auf TCP oder UDP selbst zu implementieren. Viele Protokolle bieten zudem aus Anwendungssicht ähnliche oder gar gleiche Transportdienste an, was die Festlegung auf ein Protokoll zur Entwicklungszeit erschwert. Häufig wird deswegen direkt zu TCP oder verschiedenen auf der *Socket* Schnittstelle (und UDP) aufsetzenden Middleware Lösungen gegriffen. Somit scheint weniger die Transportschicht selbst als vielmehr die Schnittelle zur Anwendungsschicht der Ursprung vieler Probleme zu sein.

Ein weiteres Problem ist, dass Anwendungsinstanzen bzw. deren Netzwerkzugangspunkte eindeutig mit Hilfe einer IP Adresse und eines Ports identifiziert werden (neben dem gewählten Transportprotokoll). Diese ändern sich beispielsweise bei mobilen Geräten jedoch regelmäßig. Zudem stehen mobilen Geräten im Allgemeinen mehrere Netzwerkzugänge zur Verfügung, sodass sie häufig über mehr als eine Adresse (parallel) erreichbar sind. Eine für die Anwendung transparente Unterstützung von Eigenschaften wie Mobilität und Multihoming wird damit stark erschwert. Ein generischer Identifikator, der von konkreten Adressen und Diensten abstrahiert und für die Laufzeit einer Anwendung konstant bleibt, sollte stattdessen automatisch von den unterliegenden Schichten bereitgestellt werden.

1.2 Anforderungen

Als Identifikatoren auf Anwendungsebene sollen eindeutige Namen verwendet werden, statt direkt mit den Adressen der Vermittlungsschicht in Kontakt zu kommen. Die Namensauflösung für die Anwendung findet dann transparent in der unterliegenden Schicht statt. Dadurch wird zum einen die nahtlose Kommunikation zwischen Endsystemen über

Adressänderungen hinweg erleichtert. Zudem ermöglicht es langfristig den Austausch des Adressschemas oder auch die parallele Verwendung unterschiedlicher Adressschemata in unterschiedlichen (Sub)netzen auf der Vermittlungsschicht. Eine Vielzahl anderer Arbeiten beschäftigt sich bereits mit diesem Problem unter dem Thema Identifier(ID)/Locator Split, so dass an dieser Stelle davon ausgegangen wird, dass eine zufriedenstellende Lösung des Problems existiert. Analog dazu soll die Adress- und Namensverwaltung, ähnlich dem heutigen DNS (Domain Name System) Dienst, unterhalb der Anwendungsschicht sitzen und die Auflösung eines Namens auf verschiedene Adressen transparent für die Anwendung ermöglichen.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es einzelne Elemente und Bausteine einer Alternative zur *Socket* Schnittstelle zu identifizieren und zu beschreiben und somit eine exemplarische Anwendungsschnittstelle beziehungsweise Transportdienstschicht zwischen der Anwendung und dem Netzwerk zu entwerfen, welche heutige Probleme vermeidet. Diese Anwendungsschnittstelle abstrahiert von konkreten Transportprotokollen und stellt stattdessen unterschiedliche Protokoll-agnostische Transportdienste zur Verfügung, um Innovationen im Netz zu fördern und eine bessere Unterstützung für neuere und zukünftige Transportprotokolle zu erreichen. Das Problem der Verfügbarkeit eines Protokolls in einem bestimmten Teil des Netzes und auf End- und Zwischensystemen, wird damit dem Anwendungsentwickler abgenommen. Ein weiterer Vorteil ist, dass erst zur Laufzeit ein konkretes Protokoll gewählt werden muss. Anwendungen können dadurch abhängig vom Kontext in dem sie ausgeführt werden, von einer adaptiven Protokollwahl profitieren, ohne dass eine Anpassung der Anwendung nötig ist. Damit werden Anwendungen automatisch flexibler und breiter einsetzbar.

Neben einer Menge von identifizierten Transportdiensten wird zusätzlich auch eine geeignete Abbildung besagter Dienste auf existierende Protokolle benötigt. Verschiedene verwandte Arbeiten haben sich bereits mit der Identifikation von geeigneten Diensten beschäftigt. Wichtig ist dabei vor allem die Liste der Dienste variabel und erweiterbar zu halten, da zu erwarten ist, dass im Laufe der Zeit neue Dienste benötigt werden. Im einfachsten Fall kann eine Anwendung aus der Liste dieser Dienste wählen. Im Rahmen dieser Arbeit wird zusätzlich vorgeschlagen auf der Anwendungsebene verschiedene benötigte und optionale Diensteigenschaften angeben zu können, um auf Basis dieser Diensteigenschaften eine Transportprotokollauswahl zu ermöglichen.

Internetdienstanbieter (ISPs) können ebenfalls von der Beschreibung der gewünschten Diensteigenschaften profitieren, da sie zusätzliche Informationen über angeforderte Transportverbindungen erhalten, die es ihnen ermöglichen können, den Netzwerkver-

kehr zu optimieren. Für den Anwendungsentwickler bedeutet dies nur einen geringen Mehraufwand, demgegenüber eine breitere Auswahl an Transportdiensten steht. Zudem sollen Anwendungsentwickler mit Hilfe der Anwendungsschnittstelle in der Lage sein, verschiedene Kommunikationsmuster intuitiv auf der Anwendungsebene umzusetzen.

Des Weiteren wird ein Rahmenwerk entworfen, welches die zuvor beschriebene Anwendungsschnittstelle umsetzt. Typische heute häufig auf der Anwendungsschicht verortete Aufgaben sollen dabei vom Rahmenwerk übernommen und Anwendungen als Dienst zur Verfügung gestellt werden. Dabei werden auch Aufgaben die klassischerweise auf der Sitzungsschicht zu finden sind übernommen. Das Rahmenwerk bietet Anwendungen die Möglichkeit Kommunikationsassoziationen untereinander zu erstellen, welche unabhängig von bestimmten Transportprotokollen sind und die Lebensdauer einzelner Transportprotokollverbindungen überdauern.

Die Anwendungsschnittstelle bietet einen erweiterten Satz von Dienstprimitiven um auf das Netzwerk zuzugreifen. Zusätzlich zur Unterstützung verschiedener Kommunikationsmuster, welche sich durch die Dienstprimitiven umsetzten lassen, schlägt diese Arbeit die Einführung von *Companion* genannten Zwischensystemen im Netz vor, welche transparent für die Anwendung Transportdienst-unterstützend agieren und nahtlos in Ende-zu-Ende Kommunikationsbeziehungen einbezogen werden können. Ein im Netzwerk platzierter Zwischenspeicher für große anfallende Datenmengen stellt beispielsweise einen Anwendungsfall für einen *Companion* dar. *Companions* können von Endsystemen selbst oder auch zum Beispiel von Internetdienstanbietern bereitgestellt werden. Heutige Lösungen wie zum Beispiel *MobileIP* setzten bereits auf die Existenz eines ähnlich gelagerten *Home Agents*, welcher als Zwischensystem fungiert und mobile Endgeräte unterstützt. Das Ziel dieser Arbeit ist es für solche existierenden und ähnlichen Anwendungsfälle generische Lösungen durch den Einsatz flexibler Zwischensysteme zu bieten.

Zur Evaluierung der in der Arbeit vorgestellten Konzepte wird exemplarisch ein Transportdienst definiert und anhand verschiedener Arbeiten die Einsetzbarkeit der neuen Anwendungsschnittstelle und Transportdienstbeschreibungen aufgezeigt. Das entworfene Rahmenwerk wird zudem prototypisch implementiert. Anhand von verschiedenen Szenarien, welche potentiell vom Einsatz der *Companion* Knoten profitieren, werden Testfälle mit Hilfe eines kleinen *Testbeds* emuliert und verschiedene Eigenschaften der Anwendungsschnittstelle und insbesondere der *Companion* Knoten untersucht.

Im Zentrum der Betrachtung steht dabei der zusätzlich durch die Anwendungsschnittstelle verursachte Kommunikationsaufwand gegenüber der reinen *Socket* Schnittstelle und der durch den Einsatz von *Companion* Knoten erzielte Leistungszuwachs im Hinblick auf Parameter wie dem erreichten Durchsatz oder der Gesamtdauer einer Datenübertragung für daran beteiligte Systeme.

Kapitel 2

Grundlagen

In diesem Kapitel werden die für diese Arbeit notwendigen Grundlagen beschrieben. Zunächst wird kurz erläutert was allgemein unter einer verteilten Anwendung zu verstehen ist und anschließend von der Anwendungsschicht an abwärts das Schichtenmodell des heutigen Internets erklärt. Dabei werden für die folgenden Kapitel notwendige Begriffe eingeführt, definiert und voneinander abgegrenzt.

2.1 Verteilte Anwendungen

Anwendungen werden als Prozess auf einem Rechner beziehungsweise Endsystem ausgeführt. Verteilte Anwendungen bestehen aus mehreren Komponenten die auf unterschiedliche Endsysteme verteilt sind und über ein Netzwerk miteinander kommunizieren. Diese Komponenten beziehungsweise Anwendungsprozesse werden im Folgenden auch Programme oder Anwendungsinstanzen genannt.

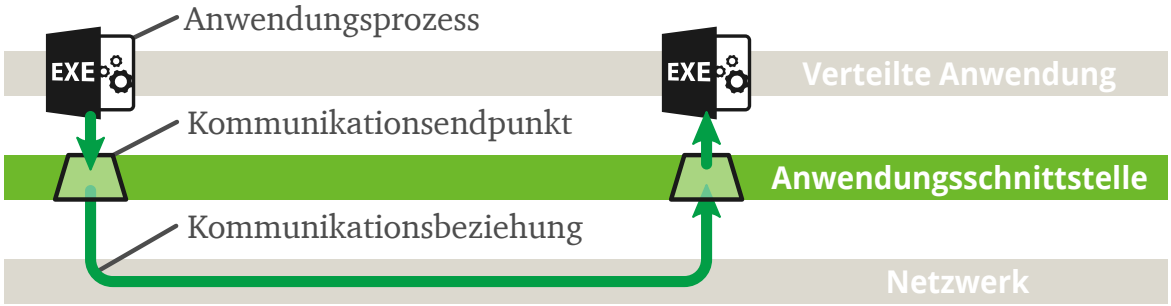


Abbildung 2.1: *Kommunikationsbeziehung zwischen zwei Anwendungsprozessen.*

Abbildung 2.1 zeigt zwei *Anwendungsprozesse*. Gemeinsam bilden diese die *verteilte Anwendung*. Um miteinander zu kommunizieren beziehungsweise Daten auszutauschen,

wird zwischen den Prozessen eine *Kommunikationsbeziehung* aufgebaut. Dies geschieht mit Hilfe einer **Anwendungsschnittstelle**, welche sich zwischen der Anwendung und dem Netzwerk befindet. Mit Hilfe der Anwendungsschnittstelle können Anwendungen *Kommunikationsendpunkte* erzeugen. Oberhalb der Anwendungsschnittstelle sprechen Anwendungen ihr eigenes Anwendungsprotokoll, welches zwischen den Anwendungsprozessen gesprochen wird. Unterhalb der Anwendungsschnittstelle befindet sich das Netzwerk, über das Daten zwischen Anwendungsprozessen ausgetauscht werden.

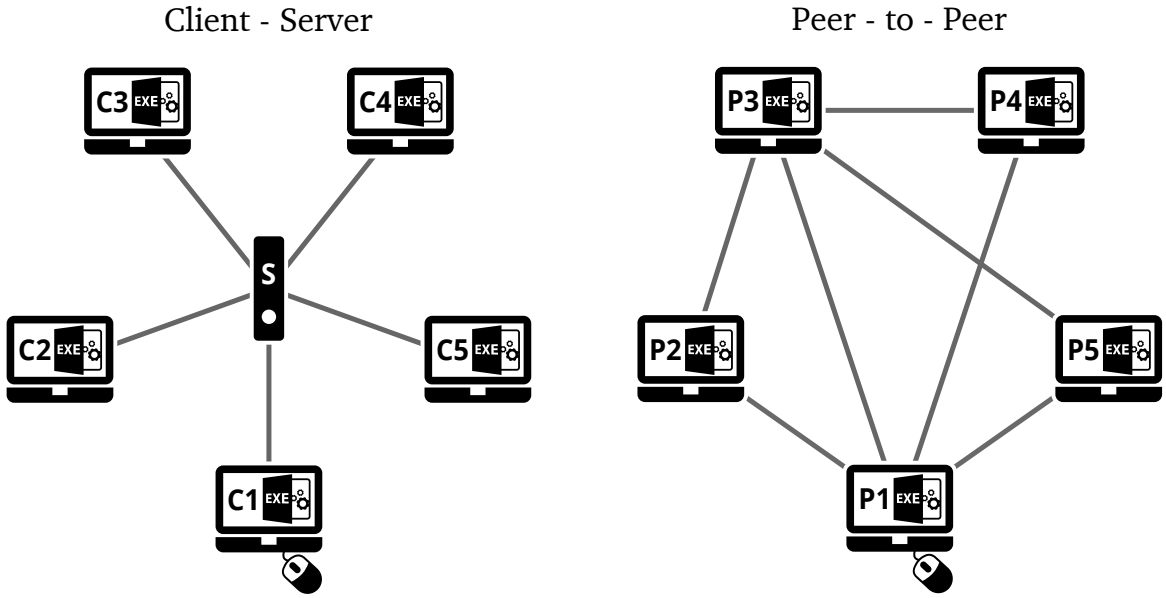


Abbildung 2.2: *Client-Server und Peer-to-Peer Kommunikationsmuster.*

Abbildung 2.2 zeigt die zwei gängigsten Konzepte zur Aufgabenverteilung auf verschiedene Anwendungsprozesse bei verteilten Anwendungen. Das Standardkonzept bildet das Client-Server-Modell (links). Der Server bietet einen bestimmten Dienst an der von einem oder mehreren Clients zur Lösung der eigenen Aufgaben oder Teilen davon genutzt werden kann. Klassische Anwendungen die nach diesem Prinzip funktionieren sind beispielsweise der *E-Mail* Versand oder der Zugriff auf das *World Wide Web* per *Webbrowser*. In diesen Fällen werden Client und Server häufig nicht gemeinsam, sondern getrennt voneinander entwickelt. Durch einheitliche Protokolle wie beispielsweise SMTP für den E-Mail Versand oder HTTP, welches von Webservern und -browsern verwendet wird, ist dennoch gewährleistet, dass Client und Server miteinander kommunizieren und interagieren können. Aber auch Anwendungen bei denen alle Komponenten vom gleichen Entwickler stammen, setzten häufig auf das Client-Server-Prinzip.

Die Kommunikation folgt hierbei in der Regel dem *Request Response Schema*. Der Server ist zunächst in Bereitschaft, verhält sich jedoch passiv und wartet auf Anforderungen (*Request*) eines Clients. Im Gegensatz dazu fordert der Client aktiv einen Dienst an. Er

sendet also ein *Request* auf das der Server mit einer entsprechenden Antwort antwortet (*Reponse*). Den Ablauf der Kommunikation zwischen Client und Server regelt dabei ein definiertes Protokoll. Dieses legt unter anderem das Format der Anfragen und der Antworten darauf fest und bestimmt die Bedeutung der zwischen Server und Client ausgetauschten Daten.

Beim Client-Server-Modell sind die Komponenten Client und Server grundsätzlich getrennt und auf verschiedene Programme verteilt. Im Unterschied dazu existiert das Peer-to-Peer-Modell (Abbildung 2.2 rechts), bei dem ein beteiligtes Programm auf einem System innerhalb des Netzwerkes gleichzeitig als Client und Server fungiert. In einem Peer-to-Peer-System sind alle Programme gleichberechtigt und können sowohl Dienste in Anspruch nehmen, als auch zur Verfügung stellen. Während im Client-Server-Modell mehrere Clients nur über einen zentralen Server miteinander kommunizieren, entfällt dieser im Peer-to-Peer-Modell und die einzelnen Programme (in diesem Falle Peers genannt) kommunizieren direkt untereinander. Doch auch im Peer-to-Peer-Modell lässt sich für jeden Kommunikationsvorgang zwischen jeweils zwei Systemen eine Client und eine Server Seite bestimmen, auch wenn Programme beide Aufgaben wechselnd übernehmen. Deswegen wird im weiteren Verlauf dieser Arbeit unabhängig vom verwendeten Modell, bezogen auf eine bestimmte Kommunikationsbeziehung zwischen zwei Programmen, von Server und Client Seite beziehungsweise Rolle gesprochen. Diese bezeichnen dann in der Regel das einen Dienst bereitstellende (Server) und anfordernde (Client) Programm an den jeweiligen Enden der Kommunikationsbeziehung.

2.2 Das Schichtenmodell des Internets

Bisher wurde das Netzwerk nur als Schicht unterhalb der Anwendungsschnittstelle eingeführt. Tatsächlich besteht das heute im Einsatz befindliche Internet aber ebenfalls aus mehreren logisch voneinander getrennten Schichten, welche alle unterschiedliche Aufgaben erfüllen, um im Zusammenspiel Kommunikationsdienste über ein Netzwerk realisieren zu können.

2.2.1 Das ISO/OSI Referenzmodell

Abbildung 2.3 zeigt das *Open Systems Interconnection* (OSI) Modell welches heute das Referenzmodell für Netzwerkarchitekturen als Schichtenarchitektur darstellt und 1984 von der *International Telecommunication Union* (ITU) und der *International Organization for Standardization* (ISO) als Standard veröffentlicht wurde.

Es dient der Kommunikation über verschiedenste Systeme hinweg und soll die Weiterentwicklung des Netzes begünstigen. Dazu definiert das Modell sieben aufeinander

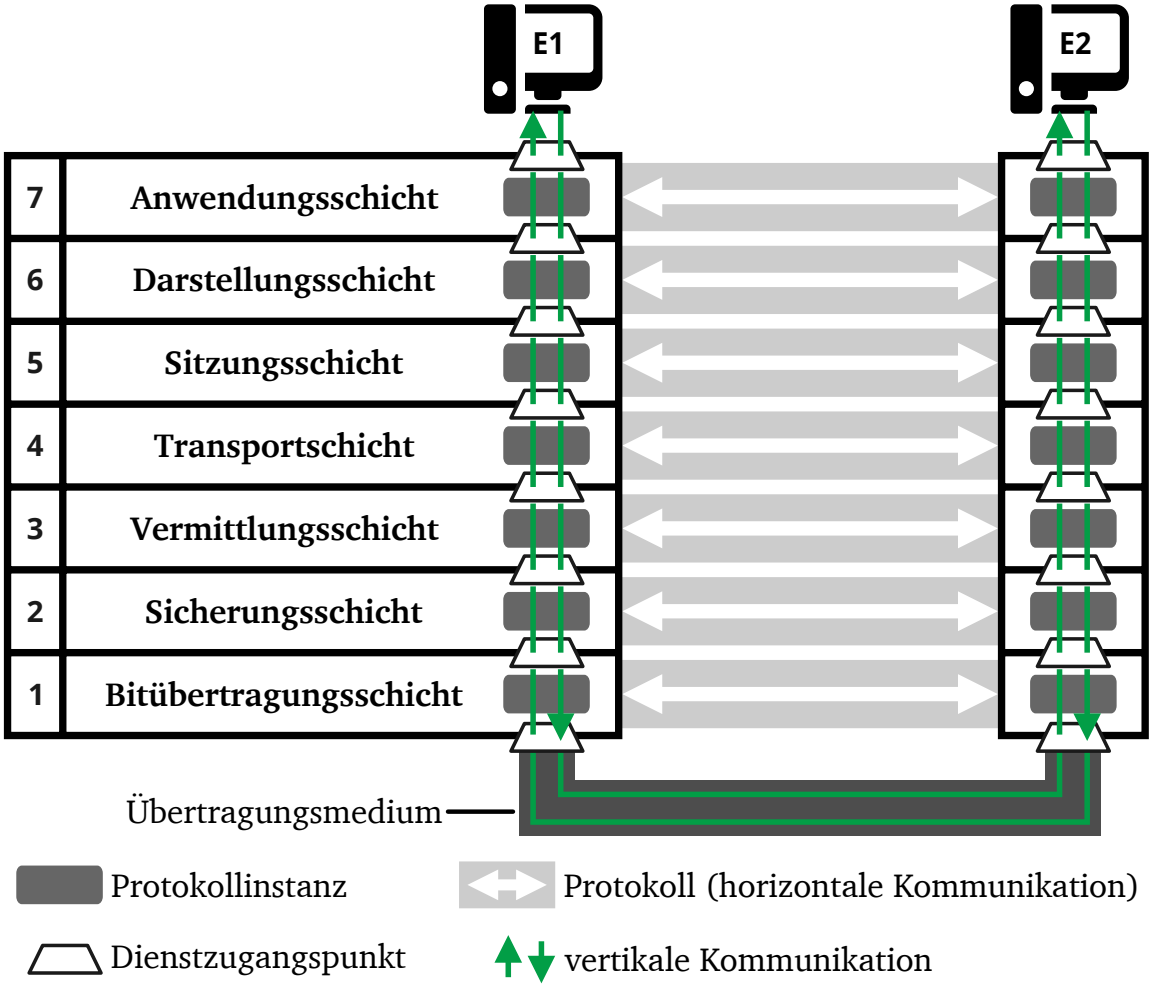


Abbildung 2.3: *Das ISO-OSI 7 Schichten Referenzmodell.*

folgende Schichten mit unterschiedlichen jeweils klar definierten Aufgaben. Diese Aufgaben werden auch Dienste genannt. Jede Schicht bietet ihren Dienst der darüber liegenden Schicht an einem Dienstzugangspunkt an. Innerhalb einer Schicht kommunizieren die einzelnen Systeme über ein definiertes Protokoll. Aufgrund der fest definierten Schnittstellen in Form der Dienstzugangspunkte sind die Protokolle einer Schicht grundsätzlich austauschbar. Auf den Systemen befindet sich jeweils pro Schicht eine Protokollinstanz.

Da im Internet Dienste unterschiedlichster Art bereitgestellt werden, ist die dazu erforderliche Kommunikation zwischen den einzelnen Systemen sehr komplex. Eine Vielzahl von Aufgaben muss bewältigt werden, um eine Kommunikationsbeziehung zwischen Anwendungsprozessen aufzubauen. Die meisten Anwendungen haben dabei an die Kommunikation zusätzliche Anforderungen wie Zuverlässigkeit oder Effizienz der Datenübertragung, die erfüllt werden müssen. Die zu lösenden Probleme reichen von der

elektronischen Übertragung der Signale auf der untersten Schicht, über das Finden einer passenden Route durchs Netz bis hin zur fehlerfreien Übertragung von Nachrichten.

Aufgrund der vielfältigen Aufgaben die erledigt werden müssen, hat jede der sieben im ISO/OSI-Modell definiert Schichten aus Abbildung 2.3 eine fest vorgegebene Aufgabe, welche durch die Protokollinstanzen umgesetzt wird.

Während die Protokolle eine Art horizontale (virtuelle) Verbindung zwischen den Schichten zweier Systeme darstellen, ist der eigentliche Datenfluss innerhalb der Systeme vertikal. Dateneinheiten werden durch die einzelnen Schichten vom Anwendungsprozess abwärts Richtung Übertragungsmedium gereicht. Dabei nehmen die Schichten jeweils den von der darunterliegenden Schicht angebotenen Dienst in Anspruch und reichen die Daten über die Dienstzugangspunkte von Protokollinstanz zu Protokollinstanz. Auf der Gegenseite werden die Daten wieder vom Übertragungsmedium zur Anwendung hinaufgereicht.

Bei den sieben durch das ISO/OSI Referenzmodell beschriebenen Schichten handelt es sich um Folgende:

- Schicht 1: Bitübertragungsschicht

  Die Aufgabe dieser Schicht ist die Bereitstellung physischer Verbindungen zur Übertragung von Bits. Je nach Übertragungsmedium beispielsweise über elektrische oder optische Signale oder über elektromagnetische Wellen.

  Die digitale Bitübertragung kann dabei sowohl leitungsgebunden als auch über eine drahtlose Übertragungsstrecke stattfinden. Die gemeinsame Nutzung eines geteilten Mediums mittels Multiplexing Verfahren gehört genauso zu den zu lösenden Problemen, wie die Codierung der einzelnen Bits auf dem Medium.

- Schicht 2: Sicherungsschicht

  Die Aufgabe dieser Schicht ist eine fehlerfreie Übertragung des Bitdatenstromes zwischen zwei Systemen. Die Schicht regelt den konkurrierenden Zugriff mehrerer Parteien auf ein geteiltes Übertragungsmedium. Dazu wird der Bitdatenstrom in Blöcke, welche auch als Rahmen bezeichnet werden, aufgeteilt. Diese enthalten Prüfsummen zur Fehlerüberprüfung. Der Empfänger kann mit ihrer Hilfe Fehler erkennen und teilweise korrigieren. Andernfalls werden fehlerhafte Rahmen verworfen.

  Das Ethernet Protokoll mit dem Zugriffskontrollverfahren CSMA/CD ist ein Beispiel für ein Protokoll welches sowohl auf Schicht 1 als auch 2 einzuordnen ist. Dagegen handelt es sich beim High-Level Data Link Control (HDLC) Protokoll um ein reines Schicht-2-Protokoll.

- Schicht 3: Vermittlungsschicht

  Diese Schicht sorgt bei leitungs- oder paketorientierten Diensten für die Etablierung eines dedizierten Übertragungskanals beziehungsweise die korrekte Weitervermittlung von Datenpaketen beziehungsweise Datagrammen. Statt zwischen zwei Systemen geht die Übertragung von Daten in diesem Fall jeweils über das gesamte Kommunikationsnetz hinweg. Dazu muss eine Wegesuche (Routing) zwischen den einzelnen Netzwerkknoten durchgeführt werden. Da nur selten eine direkte Kommunikation zwischen Absender und Empfänger möglich ist, müssen Datagramme von mehreren Systemen, die zwischen Absender und Empfänger liegen, weitergeleitet werden. Weitervermittelte Datagramme erhalten dabei auf jedem System ein neues Zwischenziel, welches den nächsten Knoten auf dem Weg angibt.

  Auf dieser Schicht werden neben der Wegewahl auch netzwerkübergreifende Adressen bereitgestellt. Dabei handelt es sich heute in der Regel um IPv4 oder IPv6 Adressen durch das Internet Protocol.

- Schicht 4: Transportschicht

  Die Aufgabe dieser Schicht ist die Segmentierung des Datenstroms und beispielsweise die Stauvermeidung. Ein Datensegment ist dabei ein Datagramm mit zusätzlichen für die Schicht 4 relevanten Protokollelementen. Zu der IP Adresse der Schicht 3 kommt als Schicht 4 Adresse der Port hinzu, über den mehrere Verbindungen zwischen den gleichen Systemen eindeutig identifiziert werden können.

  Den anwendungsorientierten Schichten wird ein einheitlicher Zugriff auf das Kommunikationsnetz geboten, der von den unterliegenden Schichten abstrahiert und die Datenübertragung Ende-zu-Ende zwischen Absender und Empfänger sichert.

  Je nach eingesetztem Transportprotokoll werden Dienstgüteeigenschaften wie die Reihenfolgetreue von Segmenten oder die Duplikatfreiheit umgesetzt.

  Die gängigsten Transportprotokolle sind das Transmission Control Protocol (TCP) und das User Datagram Protocol (UDP).

- Schicht 5: Sitzungsschicht

  Die Aufgabe dieser Schicht ist die Steuerung logischer Verbindungen. Die Funktionen der Sitzungsschicht erhalten Kommunikationsbeziehungen zwischen Sender und Empfänger aufrecht. Dazu werden Daten ausgetauscht die der Einleitung, Aufrechterhaltung und dem Neustart von Sitzungen, die unterbrochen oder längere Zeit nicht verwendet wurden, dienen.

  Zu diesem Zweck wird der Auf- und Abbau von Transportverbindung zwischen kommunizierenden Endgeräten angestoßen. Sitzung können sehr kurz, beispielsweise

für die Dauer der Übertragung einer Nachricht in eine Richtung, sein. Während andere Sitzungen länger dauern und in diesem Fall auch durch eine der kommunizierenden Parteien unterbrochen werden können.

Auf der Sitzungsschicht findet sich beispielsweise das RPC-Protokoll (Remote Procedure Call).

- Schicht 6: Darstellungsschicht

  Diese Schicht hat die Aufgabe eine systemabhängige Darstellung der übertragenen Daten (zum Beispiel ASCII, UTF8) in eine unabhängige Form umzuwandeln und damit einen syntaktisch korrekten Datenaustausch zwischen unterschiedlichen Systemen zu gewährleisten.

  Die Darstellungsschicht soll sicherstellen, dass Daten, die von einem Sender gesendet werden, auf der Anwendungsschicht des Empfängers gelesen werden können. Dazu gehören auch Aufgaben wie die Datenkompression und die Datenverschlüsselung. Falls erforderlich soll die Darstellungsschicht zwischen verschiedenen Datenformaten übersetzen, indem beispielsweise ein beidseitig verständliches Datenformat wie ASN.1 (Abstract Syntax Notation One) verwendet wird.

- Schicht 7: Anwendungsschicht

  Auf dieser Schicht finden sich die verteilten Anwendungen, welche unterschiedlichste Anwendungsdienste bereitstellen können.

  Typische Anwendungen sind *Webbrowser* und *Webserver*, *E-Mail* Clients oder *Filesharing* Dienste wie zum Beispiel *BitTorrent*.

2.2.2 Das DoD Schichtenmodell & das TCP/IP Referenzmodell

Vor der Entwicklung des ISO/OSI Referenzmodells existierte bereits ein einfacheres Modell: Das *DoD Internet Architecture Model*. Es ist ebenfalls eine Schichtenarchitektur, welche die einzelnen Aufgaben bei der Datenübertragung in aufeinander aufbauende Schichten einteilt. Das Modell hat seinen Ursprung in Arbeiten die Ende der 1960er Jahre von der DARPA für das Verteidigungsministerium der Vereinigten Staaten (*Department of Defense* oder kurz: *DoD*) durchgeführt wurden [CC83]. Es wurde darauf basierend weiterentwickelt und besteht aus vier Schichten. Die Architektur wurde dabei zunächst nur für rein militärische Anwendungen konzipiert. Deswegen war ein zentrales Designziel durch eine dezentrale Struktur ein robustes und vor Ausfällen geschütztes Netzwerk zu schaffen.

Das DoD Internet Architecture Model ist in Abbildung 2.4 neben dem ISO/OSI Referenzmodell dargestellt und zeigt in welchem Bezug die Schichten des jeweiligen Modells

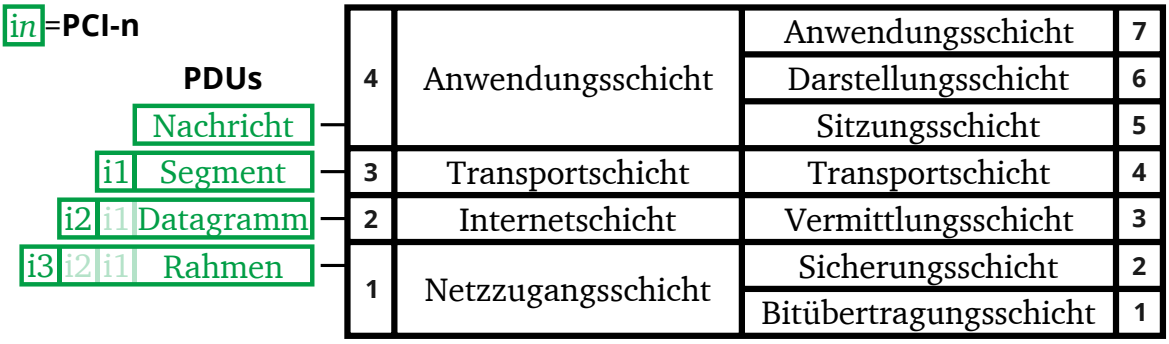


Abbildung 2.4: *Das DoD bzw. TCP/IP Referenzmodell im Vergleich zum ISO/OSI Modell.*

zueinander stehen. Für jede Schicht des DoD Modells existiert mindestens ein häufig sogar eine Reihe von Protokollen, die die Aufgaben der jeweiligen Schicht auf unterschiedliche Weise lösen. Diese Protokolle werden auch die Internetprotokollfamilie genannt und das DoD Modell wird auch als TCP/IP-Referenzmodell bezeichnet, da es sich zu einem großen Teil um Protokolle handelt, welche auch heute noch im Einsatz sind.

Da einzelne Protokolle wie beispielsweise TCP bereits vor dem ISO/OSI Referenzmodell spezifiziert wurden, entspricht das DoD beziehungsweise TCP/IP-Referenzmodell Modell eher der heute im Netz vorzufindenden Realität. Das ISO/OSI Referenzmodell hingegen stellt vielmehr ein wünschenswertes Idealbild da, dem die heutige Internetarchitektur nicht hundertprozentig gerecht wird. Es wird häufiger verwendet, um Probleme der Netzwerkkommunikation im Allgemeinen und abstrakt zu betrachten.

Die Protokolle der Internetprotokollfamilie sind darauf zugeschnitten, den Datenaustausch über die Grenzen lokaler Netzwerke hinweg zu ermöglichen. Dazu werden Datenpakete über mehrere Punkt-zu-Punkt Verbindungen (auch *Hops* genannt) weitervermittelt, um basierend darauf Verbindungen zwischen Netzwerkteilnehmern über mehrere *Hops* hinweg als einen vollständigen Pfad herzustellen.

Neben den Schichten zeigt Abbildung 2.4 auch die jeweiligen mit den Schichten assoziierten Dateneinheiten beziehungsweise *Protocol Data Units* (PDUs). Beim durchreichen der Daten fügt jede Schicht sogenannte *Protocol Control Information* (PCI) an die von der vorherigen Schicht erhaltene PDU an. Diese wird auf der Gegenseite von der Protokollinstanz der gleichen Schicht wieder entfernt und verarbeitet. Die einzelnen Schichten erfüllen folgende Funktionen:

- Die Protokolle der **Anwendungsschicht** bestimmen den Aufbau der eigentlich ausgetauschten **Nachrichten**, also den zwischen den Anwendungsprozessen beziehungsweise -instanzen ausgetauschten Nutzdaten. Für jede Information, die übertragen werden soll, muss deren exakte Position in der Nachricht und der genaue Aufbau der verwendeten Nachrichten festgelegt sein. Die Anwendungsschicht setzt direkt über der Transportschicht an, so dass Aufgaben der Sitzungs- und Darstel-

lungsschicht von den Anwendungen selbst umgesetzt werden müssen. Zwischen der Anwendungs- und der Transportschicht befindet sich die Anwendungsschnittstelle, über die Anwendungen Zugriff auf das Netzwerk erhalten.

- Die **Transportschicht** ermöglicht eine Ende-zu-Ende Kommunikation zwischen zwei Anwendungsprozessen. Jedes **Segment** enthält dabei zusätzlich Informationen, für welchen auf dem Zielrechner laufenden Prozess, der auf Daten aus dem Netz wartet, es bestimmt ist. Es wird zwischen verschiedenen grundlegenden Ansätzen zur Informationsübertragung unterschieden:

  Bei der verbindungslosen Übertragung ist nicht gewährleistet, dass ein Segment auch ankommt. Die Kontrolle auf verlorengegangene oder verfälschte Segmente muss auf der Anwendungsschicht erfolgen.

  Bei einer bestätigten, verbindungslosen Übertragung ist eine Bestätigungsmeldung vorgesehen, um den Empfang eines Segmentes dem Sender mitzuteilen.

  Bei einer verbindungsorientierten Übertragung wird vor der Übertragung eine logische Verbindung aufgebaut und aufrechterhalten. Diese ist zustandsbehaftet und zum Erhalt der Verbindung werden Kontrollinformationen zwischen den Systemen ausgetauscht, auch wenn gerade keine Daten übertragen werden. Dadurch wird es möglich die korrekte Reihenfolge der Segmente und eine Datenflusskontrolle zu gewährleisten. Über Sendewiederholungen können zudem auch verlorengegangene und fehlerhafte Segmente korrekt übertragen werden.

- Auf der **Internetschicht** erfolgt die Weitervermittlung von **Datagrammen** aufgrund von Sender und Empfänger Adressen, welche die Endsysteme auf denen die Anwendungsinstanzen laufen identifizieren und an die Segmente angefügt werden. Hier findet die Weiterleitung statt. Anhand von Direktverbindungen zwischen benachbarten Systemen im Netz, wird für empfangene Datagramme das jeweils nächste Zwischenziel ermittelt und das Datagramm dorthin weitergeleitet. Jedes System hat dazu ebenfalls eine eigene Netzwerkadresse, welche unabhängig vom Übertragungsmedium ist.

- Auf der **Netzzugangsschicht** werden **Rahmen** übertragen. Diese Kennzeichen eindeutig den Anfang und das Ende aufeinanderfolgender Dateneinheiten. Die Netzzugangsschicht enthält keine Protokolle der TCP/IP-Familie. Sie bietet vielmehr verschiedene Techniken zur Datenübertragung von Punkt-zu-Punkt an, wie beispielsweise Ethernet oder 802.11 (WLAN). Die Netzzugangsschicht entspricht der Sicherungs- und Bitübertragungsschicht des ISO/OSI-Referenzmodells. In ihr wird das Problem der Datenübertragung zwischen Rechnern und Prozessen auf der physikalischen Ebene gelöst. Dazu werden den Rahmen System-spezifische

Adressen hinzugefügt, welche angeben welches System als nächstes den Rahmen erhalten soll.

Abbildung 2.5 zeigt für zwei Endsysteme und ein Zwischensystem den tatsächlichen Pfad einiger Daten von Anwendungsinstanz zu Anwendungsinstanz. Dabei werden die Daten mindestens einmal durch sämtliche Schichten hinab und wieder hinaufgereicht. Insbesondere auf Zwischensystemen werden aber in der Regel nicht sämtliche Schichten durchlaufen. Hier wird auf dem Zwischensystem Beispielsweise nur die Netzzugangs- und Internetschicht durchlaufen. Die horizontale Kommunikation zwischen einzelnen Protokollinstanzen findet also nicht zwingend auf allen Schichten zwischen den gleichen Systemen statt. Insbesondere befindet sich nicht auf sämtlichen Zwischensystemen eine Protokollinstanz der Protokolle der Anwendungs- und Transportschicht.

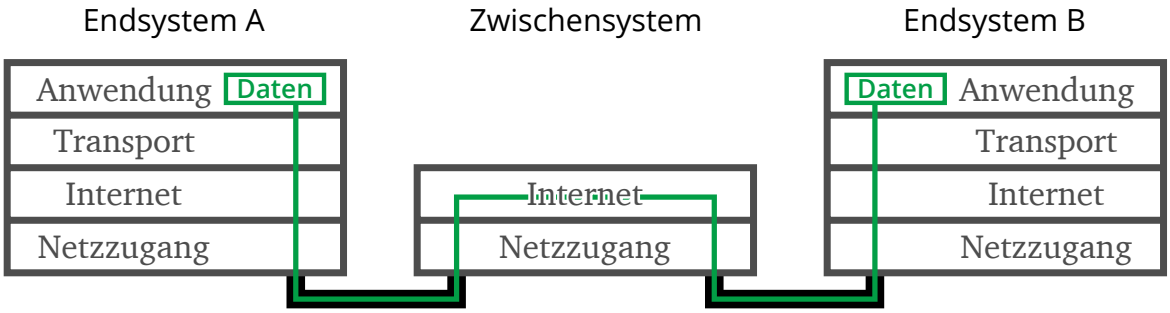


Abbildung 2.5: *Der Datenpfad über verschiedene Systeme und Übertragungsmedien hinweg.*

2.3 Begriffe und Definitionen

An dieser Stelle werden nochmal für die folgenden Kapitel relevante Begriffe definiert und voneinander abgegrenzt:

Kommunikationsbeziehung

Eine Kommunikationsbeziehung meint im Folgenden im weitesten Sinne den Vorgang des Datenaustausches zwischen zwei oder mehr beteiligten Parteien. Bei den beteiligten Parteien handelt es sich um konkrete Anwendungsinstanzen einer oder verschiedener Anwendungen. Bei den Anwendungsinstanzen handelt es sich letztendlich um auf Endsystemen ausgeführte Prozesse. Zwischen diesen findet der eigentliche Datenaustausch über mindestens eine auf dem jeweiligen Endsystem vorhandene Netzwerkschnittstelle statt.

Endsystem

Endsysteme sind alle netzwerkfähigen Geräte (Smartphones, Tablets, PCs, Laptops aber auch Server), welche über eine Kommunikationsbeziehung miteinander Daten austauschen. Dabei ist es unerheblich ob es sich dabei um physikalische oder virtuelle Geräte handelt oder ob diese sich am Netzrand oder im Kern des Netzes befinden. Dementsprechend sind verschiedene Systeme immer nur Endsysteme in Bezug auf eine oder mehrere Kommunikationsbeziehungen.

Zwischensystem

Zwischensysteme sind analog zu den Endsystemen alle Systeme die Teil eines Netzwerkes sind und Daten für eine Kommunikationsbeziehung transportieren, ohne ein Absender oder Empfänger der Daten zu sein. Dies schließt neben beispielsweise Routern und Brücken im Netzwerk auch sämtliche anderen netzwerkfähigen Geräte mit ein, da auch Smartphones, PCs oder ähnliche Geräte als Zwischensysteme für Kommunikationsbeziehungen fungieren können, an denen sie nicht als Endsystem beteiligt sind.

Kommunikationsassoziation

Kommunikationsassoziationen werden in Kapitel 3.2 eingeführt und in Kapitel 4.5 genauer beschrieben. Grundsätzlich handelt es sich dabei um eine Kommunikationsbeziehung zwischen genau zwei Anwendungsinstanzen. Diese befinden sich auf Endsystemen und tauschen über eine beliebige und nicht konstante Menge von Zwischensystemen Daten aus. Der Datenaustausch wird dabei über einen definierten von der Anwendung gewählten Transportdienst umgesetzt, der für die Dauer der Kommunikationsassoziation fest mit dieser verbunden ist.

Dabei existieren Kommunikationsassoziation unabhängig von konkreten Protokollen (egal ob verbindungsorientiert oder verbindungslos) und unabhängig davon, ob gerade aktiv Daten ausgetauscht werden oder nicht. Die Lebensdauer wird nur durch die Anwendungsinstanzen, welche die Kommunikationsassoziationen aufbauen und beenden, bestimmt.

Transportdienst

Transportdienste unterscheiden sich in der Art und Weise, wie Daten durchs Netz transportiert werden. Das heißt Transportdienste setzten verschiedene Transportdiensteigenschaften um, um eine bestimmte Dienstgüte für den Austausch von Daten zwischen Anwendungsinstanzen zu gewährleisten. Dies umfasst beispielsweise Qualitätsmerkmale

wie die Vollständigkeit oder Fehlerfreiheit der übertragenen Daten. Transportdienste sind letztendlich definiert durch eine Beschreibung ihrer Eigenschaften, welche von konkreten Transportprotokollen abstrahiert. Sie werden in Kapitel 3.5 ausführlicher beschrieben.

Im Rahmen dieser Arbeit meint Transportdienst alle unterhalb der Schnittstelle zwischen Netzwerk und Anwendung angebotenen Dienste, welche dem Datentransport dienen. Diese sind grundsätzlich unabhängig von der Art der übertragenen Daten und dem über dem Transportdienst angebotenen Anwendungsdienst. Transportdienste werden durch ein oder mehrere Transportprotokolle realisiert.

Anwendungsdienst

Anwendungsdienste sind alle durch eine Anwendung angebotenen Netzdienste. Sie befinden sich oberhalb der Schnittstelle zwischen Anwendung und Netzwerk und werden durch die konkreten Anwendungsinstanzen einer Anwendung realisiert. Auf der Anwendungsebene ist Anwendungsdienst-spezifisches Wissen verfügbar. Zum Austausch der zur Erbringung eines Anwendungsdienstes benötigten Daten, verwenden die Anwendungsinstanzen einen oder mehrere Transportdienste.

Anwendungsinstanz/-prozess

Anwendungsinstanzen sind die auf (verteilten) Systemen laufenden Komponenten, welche zusammen eine verteilte Anwendung bilden. Anwendungsinstanz und Anwendungsprozess werden im Laufe dieser Arbeit synonym verwendet und bezeichnen beide das Gleiche. Als Anwendung wird im Folgenden die Summe der Anwendungsinstanz aus denen die Anwendung besteht bezeichnet. Dies weicht etwas von der klassischen Definition, bei der häufig nur die Client oder die Server Seite gemeint ist ab. Statt also beispielsweise einen *Webbrowser* und einen *Webserver* jeweils als Anwendung zu bezeichnen, wären dies im Folgenden Anwendungsinstanzen, welche zusammen die Anwendung beziehungsweise den Anwendungsdienst *web browsing* realisieren.

Transportprotokoll

Transportprotokolle setzen die durch Transportdienste abstrakt definierten Dienste um. Dazu werden Transportdienstbeschreibungen auf passende Transportprotokolle abgebildet und diese entsprechend der gewünschten Transportdiensteigenschaften parametrisiert. Im Folgenden bezeichnen Transportprotokolle sämtliche Schicht 4 Protokolle, die dem Datentransport dienen. Implizit ist damit aber auch der gesamte Protokollstapel gemeint, der zur Nutzung eines Transportprotokolls benötigt wird.

Transportbeziehung/-verbindung

Eine Transportbeziehung bezeichnet einen bestehenden Kanal zwischen zwei Systemen zum Datenaustausch über ein Transportprotokoll. Transportbeziehung bezeichnet dabei korrekterweise sowohl einen Kanal über verbindungsorientierte als auch über verbindungslose Protokolle. Transportverbindung hingegen ist im eigentlichen Sinne des Wortes nur eine korrekte Bezeichnung für verbindungsorientierte Protokolle, wird im Folgenden aber Synonym zu Transportbeziehung benutzt, da in der Praxis für beide Protokollarten ein ähnlicher Zustand auf den Endsystemen angelegt, gehalten und am Ende des Datenaustausches wieder aufgeräumt werden muss. Zudem ist es aus Anwendungssicht im Allgemeinen unerheblich ob ein Protokoll verbindungsorientiert oder verbindungslos ist, da grundsätzlich aufbauend auf beiden Protokollarten die gleichen Diensteigenschaften erzielt werden können.

Protokollstapel

Als Protokollstapel wird eine Kombination von Protokollen der Schichten 1-4 bezeichnet, welche im Zusammenspiel einen bestimmten Transportdienst anbieten. So können verschiedene Transportschicht Protokolle heute über IPv4 oder IPv6 verwendet werden. Unterhalb der Vermittlungsschicht existieren abhängig vom Gerät verschiedene Netzwerkadapter, wie beispielsweise Ethernet und Wi-Fi in verschiedenen Varianten oder Mobilfunkstandards wie LTE, HSDPA, UMTS oder GSM. Bei der Abbildung von Transportdiensten auf ein konkretes Transportprotokoll muss demnach auch der zu verwendende Netzwerkadapter sowie die Adressfamilie unter der ein System erreichbar ist betrachtet und ausgewählt werden.

Kommunikationsdomäne

Innerhalb einer Kommunikationsdomäne sind alle End- und Zwischensysteme über einheitliche Adressen untereinander erreichbar. Jedes System kann Teil einer oder auch mehrere Kommunikationsdomänen sein. Heute existierende Kommunikationsdomänen sind beispielsweise die IPv4 oder IPv6 Domäne.

Kapitel 3

Die PASTE-API: Eine Anwendungsschnittstelle für zukünftige Netze

Moderne verteilte Anwendungen müssen stetig wachsende Erwartungen der Nutzer erfüllen. Nicht nur im Hinblick auf ihre Leistungsfähigkeit, sondern zunehmend auch in Form der von einer Anwendung gebotenen Funktionalität. Während die maximal verfügbare Datentransferrate im Laufe der letzten Jahre kontinuierlich gestiegen ist, beschränken sich die einer Anwendung zur Verfügung stehenden Netzwerkdienste auf einige wenige durch Standardprotokolle angebotene Dienste.

Theoretisch existiert neben TCP und UDP eine stetig wachsende Menge alternativer Transportprotokolle, wie beispielsweise *Multipath TCP* (MPTCP) [For+13; Rai+12], *Datagram Congestion Control Protocol* (DCCP) [KHF06] oder *Stream Control Transmission Protocol* (SCTP) [Ste07]. Diese bieten aktuell schon Dienste wie zum Beispiel mehrere Verbindungen zwischen einer Quelle und einem Ziel zur Erhöhung des Datendurchsatzes. Dabei werden zwischen den Verbindungen teilweise Staukontrollinformationen ausgetauscht, um weiterhin Fairness garantieren zu können. Während DCCP zum Beispiel die Wahl verschiedener Staukontrollmechanismen erlaubt.

In der Praxis entscheiden sich Anwendungsentwickler jedoch bereits zur Designzeit einer Anwendung in der Regel für TCP oder UDP als Transportprotokoll. Dies ist sowohl aus Sicht des Netzwerkbetreibers als auch des Anwendungsentwicklers unvorteilhaft. Dadurch ist es nicht möglich unterhalb der Anwendungsschicht zur Laufzeit verschiedene Protokolle auszuwählen. Abhängig vom Kontext in dem eine Anwendung ausgeführt wird, kann es aber sehr sinnvoll sein, zwischen unterschiedlichen Protokollen zu wählen. Es können beispielsweise mehrere Protokolle oder auch Protokollvarianten zur Verfügung stehen, welche einen gleichen oder ähnlichen Transportdienst erbringen. Diese können für drahtlose oder kabelgebundene Netze optimiert sein und verschiedene Eigenschaften

aufweisen, je nachdem ob sie beispielsweise im Internet-Backbone oder im Randbereich des Netzwerks eingesetzt werden. So existiert neben der klassischen Variante von TCP zum Beispiel auch eine *Datacenter* TCP [Ben+15] Version, welche auf die besonderen Rahmenbedingungen, wie hohe Datenraten und geringe Latenzen, im *Datacenter* optimiert ist. Nicht nur die initiale Wahl eines Protokolls in Abhängigkeit vom Kontext ist sinnvoll, auch ein Wechsel des eingesetzten Protokolls, sofern sich der Kontext während einer Datenübertragung ändert, kann vorteilhaft sein. Dies kann zum Beispiel aufgrund eines Standortwechsels geschehen. Auf mobilen Geräten wie Smartphones besteht beispielsweise die Möglichkeit mehrere Netzwerkadapter wie W-Lan und HSDPA parallel zu nutzen. Dies macht den Einsatz von MPTCP interessant, welches in der Praxis jedoch in vielen Domänen aufgrund restriktiv konfigurierter Middleboxes wie beispielsweise Firewalls nicht eingesetzt werden kann. Wechselt ein Gerät nun in eine Netzwerkdomäne in der MPTCP verfügbar ist, kann die TCP Variante gewechselt werden, um eine höhere Datentransferrate zu erzielen. Ebenso ist es denkbar das eingesetzte Protokoll aufgrund einer Störung beziehungsweise eines Fehlers im Netzwerk zu wechseln.

Neuere Protokolle sind für Anwendungsentwickler jedoch häufig unattraktiv. Diese können nicht automatisch genutzt werden, ohne einen Großteil der existierenden Anwendungen anpassen zu müssen. Diese Anpassungen werden aber im Gegenzug nicht oder nur sehr selten vorgenommen, da die Unterstützung neuerer Protokolle in Form von einer großflächigen Verfügbarkeit häufig nicht gegeben ist. Dies führt dazu, dass es Innovationen unterhalb der Anwendungsschicht im heutigen Internet sehr schwer haben sich durchzusetzen. Eine schrittweise Ausbringung neuer Protokolle ist zwar denkbar, sie scheitert aber daran, dass dem Anwendungsentwickler die zusätzliche Aufgabe zufällt eine Rückfalllösung auf TCP oder UDP zu implementieren. Der Aufwand dafür übersteigt den Nutzen, sofern der Mehrwert eines neuen Protokolls nur partiell verfügbar ist. Viele Protokolle bieten zudem aus Anwendungssicht ähnliche oder gar gleiche Transportdienste an, was die Festlegung auf ein konkretes Protokoll zur Entwicklungszeit erschwert. Optimaler Weise sollte das Protokoll also erst zur Laufzeit aus einer Menge von geeigneten Alternativen gewählt werden, da so bei der Wahl auch auf den Kontext, in dem eine Anwendung ausgeführt wird, eingegangen werden kann.

Dies führt im Gegenzug dazu, dass ISPs und vor allem die Netzbetreiber keinen Anreiz haben, neue Transportprotokolle auszubringen und anzubieten, solange keine breite Anwendungsunterstützung dafür existiert. Da kaum eine Anwendung Gebrauch von den neuen Protokollen machen würde, wird kein wirklicher Bedarf dafür gesehen. Die heutige Situation entspricht damit einem klassisches Henne-Ei-Problem, bei dem für beide Seiten die Anreize fehlen die Einführung und Unterstützung neuer Protokolle in Zukunft zu forcieren. Durch eine losere Kopplung von Anwendungen und unterliegendem Netzwerk lässt es sich jedoch auflösen.

Um Anwendungen und Netzwerk erfolgreich voneinander zu entkoppeln, bedarf es einer geeigneten Abstraktion an der Schnittstelle zwischen Anwendung und Netzwerk. Bevor in Kapitel 4 das Protocol Agnostic Services & Transport Enrichment (PASTE) Rahmenwerk vorgestellt wird, wird daher in diesem Kapitel zunächst die zugehörige PASTE-API eingeführt. Dabei handelt es sich um eine Anwendungsschnittstelle, welche mit dem Ziel entworfen wurde Netzwerk spezifisches Wissen aus der Anwendungsschicht in die unterliegenden Schichten zu verlagern und damit Innovationen im Netz unterhalb der Anwendungsschicht zu fördern. Dabei fordern Anwendungen einen Netzwerkdienst über eine generische Beschreibung an, anstatt ein konkretes Protokoll auszuwählen. Davon profitieren Anwendungen, die zukünftig ohne Änderungen neue Protokolle verwenden können, genauso wie Netzbetreiber, die von ihnen eingesetzte Protokolle auf spezifische Domänen zuschneiden können, solange die gleichen Transportdienste angeboten werden.

Zunächst wird im Folgenden der Ist-Zustand heutiger Anwendungsschnittstellen betrachtet und basierend darauf werden Anforderungen an eine neue Schnittstelle abgeleitet. Anschließend folgt eine Beschreibung der durch die neue Schnittstelle definierten Dienstprimitive und der Verwendung von Namensschemata innerhalb der PASTE-API. Zusätzlich werden Transportdienstbeschreibungen auf der Anwendungsebene eingeführt, welche von konkreten Transportprotokollen abstrahieren und damit die Protokollwahl aus dem Verantwortungsbereich der Anwendungen ins Netzwerk verlagern können. Abschließend wird anhand einiger exemplarischer Anwendungsfälle der Einsatz der PASTE-API veranschaulicht.

3.1 Die Anwendungsschnittstelle heute

Die Berkeley beziehungsweise die daraus hervorgegangene POSIX Socket-API kann heute als der De-Facto-Standard für Interprozess Kommunikation angesehen werden. Anwendungen greifen entweder direkt über die Schnittstelle auf die vom Betriebssystem bereitgestellten Protokollstapel zu, oder verwenden mit großer Wahrscheinlichkeit eine Bibliothek die auf der Socket-API aufsetzt. Die Grundidee der Berkeley Socket-API ist es Anwendungen einen oder mehrere Kommunikationsendpunkte zur Verfügung zu stellen, auf denen diese lesend und schreibend zugreifen können, um Daten zwischen jeweils zwei Endpunkten auszutauschen. Beim Erstellen des Endpunktes haben Anwendungen die Wahl über die gewünschte Kommunikationsdomäne, den Typ des Endpunktes und das konkret zu verwendende Transportprotokoll.

Die Wahl der Kommunikationsdomäne legt dabei implizit die Protokollfamilie fest, über die kommuniziert werden soll. Neben Varianten für lokale Kommunikation und älteren Standards wie zum Beispiel IPX, existieren im Wesentlichen zwei heute relevante Alternativen (AF_INET & AF_INET6) zur Wahl von IPv4 beziehungsweise IPv6.

Über den Endpunkt Typ wird die Kommunikationssemantik festgelegt. Folgende Typen stehen zur Auswahl:

(1) **SOCK_STREAM** stellt einen verbindungsorientierten, bidirektionalen Byte-Strom zur Verfügung, welcher die Auslieferung sämtlicher Dateneinheiten garantiert. Dabei wird nur sichergestellt, dass jede Dateneinheit mindestens einmal fehlerfrei empfangen wird. Duplikatfreiheit und Reihenfolgetreue hingegen sind optional, werden aber in der Regel ebenfalls geboten. Da beim Lesen sämtliche verfügbaren Daten an die Anwendung gereicht werden, müssen Dateneinheitsgrenzen auf der Anwendungsschicht selbständig wiederhergestellt werden.

(2) **SOCK_DGRAM** erlaubt den Versand von Datagrammen mit fester Maximalgröße. Datagramme werden dabei verbindungslos & unzuverlässig übertragen. Weder die Fehlerfreiheit der Datagramme noch deren Erhalt wird garantiert. Beim Lesen von Dateneinheiten, wird jeweils ein Datagramm erhalten.

(3) **SOCK_SEQPACKET** stellt ähnlich wie SOCK_STREAM eine verbindungsorientierte, bidirektionale Verbindung für Datagramme mit fester Maximalgröße zur Verfügung. Im Unterschied zu SOCK_STREAM werden jedoch die Dateneinheitsgrenzen eingehalten, so dass beim Lesen immer eine oder mehrere vollständige Datagramme gelesen werden. Zudem wird auch die Reihenfolgetreue der Datagramme gewährleistet.

(4) **SOCK_RDM** erlaubt den Versand von Datagrammen mit fester Maximalgröße. Datagramme werden dabei verbindungslos übertragen, aber die fehlerfreie Auslieferung ist garantiert. Die Reihenfolgetreue ist hingegen nicht gewährleistet.

(5) **SOCK_RAW** ermöglicht den Versand von Datagrammen ohne Transportschicht spezifische Informationen. Es werden also keine Transportprotokoll-spezifischen *Header* automatisch an die versendeten Dateneinheiten angefügt. Daher eignet sich SOCK_RAW zum Implementieren eigener Transportprotokolle auf der Anwendungsschicht.

Obwohl zusätzlich zum Endpunkt Typ ein konkretes Protokoll angegeben werden kann, ist in der Regel durch Protokollfamilie und Endpunkt Typ das Transportprotokoll bereits festgelegt und wird in der Praxis selten explizit angegeben. Vielmehr ist für jede Protokollfamilie pro Endpunkt Typ ein Standardprotokoll definiert, welches automatisch gewählt wird. Durch die Wahl von SOCK_STREAM oder SOCK_DGRAM wird in der Regel implizit TCP oder UDP als Transportprotokoll für die beiden Protokollfamilien AF_INET & AF_INET6 festgelegt. Während SOCK_SEQPACKET mitunter durch das *Stream Control Transmission Protocol* (SCTP) unterstützt wird, sind aktuell keine weitverbreiteten

Protokolle für die Internet Domäne zur Unterstützung von SOCK_SEQPACKET und SOCK_RDM standardisiert.

Je nach gewähltem Endpunkt Typ, unterscheidet sich der verfügbare Befehlssatz zum Senden und Empfangen von Daten. Grundsätzlich handelt es sich um eine funktionsorientierte Programmierschnittstelle. Dabei wird das Konzept der *Handles* verwendet. Man ruft eine Funktion auf, welche ein *Handle* zurückliefert. Auf diesem *Handle* lassen sich dann weitere Funktionen mit oder ohne Rückgabewert aufrufen, bis abschließend das *Handle* wieder geschlossen werden muss. SOCK_STREAM erwartet zum Beispiel das zunächst per *connect()* Befehl eine Verbindung zur Gegenstelle aufgebaut wird, wobei einmalig die Zieladresse angegeben wird. Danach können per *send()* Befehl Daten an das Ziel geschickt werden. Im Falle von SOCK_DGRAM entfällt der Verbindungsaufbau und Daten werden direkt per sendto() verschickt. Datagramme können dabei an beliebige Systeme versendet werden, dafür muss jedoch jedes Mal die entsprechende Zieladresse mit angegeben werden. In beiden Fällen werden neue Sockets mit der *socket()* Funktion erzeugt, welche den *Handle* liefert. Die Funktion *bind()* erzeugt serverseitig einen Endpunkt, zu dem Verbindungen aufgebaut oder an den Datagramme verschickt werden können, während *close()* einen *Handle* schließlich wieder schließt.

3.1.1 Probleme der Berkeley Socket-API

Die Berkeley Socket-API stammt aus der Zeit Mitte der 80er Anfang der 90er Jahre und hat aus heutiger Sicht einige konzeptionelle Probleme, da grundlegende Annahmen über das unterliegende Netzwerk heute nicht mehr zwingend gültig sind:

(1) **Kommunizierende Endgeräten befinden sich in der gleichen Kommunikationsdomäne.** Seit der Einführung von IPv6 existieren mit IPv4 und IPv6 zwei verschiedene Kommunikationsdomänen. Nicht alle Endgeräte sind immer über beide erreichbar, aber wollen dennoch untereinander kommunizieren.

(2) **Endgeräte sind stationär und ihre Adresse ändert sich nicht für die Dauer eines Datenaustausches.** Heute existiert eine Vielzahl mobiler Endgeräte, deren Adressen sich jederzeit ändern können. Durch die weite Verbreitung unterschiedlichster drahtloser Übertragungstechniken und die Mobilität der Endnutzer, wechseln Geräte heute häufig die Netzwerkdomäne und damit einhergehend auch häufig ihre IP Adresse.

(3) **Endgeräten werden eindeutig durch eine Adresse identifiziert und sind über diese erreichbar.** Aus Gründen der Redundanz und zur Erhöhung der Zuverlässigkeit, können Endgeräte parallel über mehrere Internetprovider mit dem Internet verbunden sein. Aufgrund von *multi-homing* beziehungsweise der Verfügbarkeit

mehrerer Netzwerkschnittstellen (wie z.B. ein drahtloser Zugang zu WLAN und UMTS Netzen bei Mobiltelefonen) können Endgeräte im Netzwerk demnach mehr als eine Adresse gleichzeitig besitzen, also sowohl über eine oder mehrere IPv4 oder IPv6 Adressen erreichbar sein. Dies ist ein Problem da IP Adressen damals wie heute nicht nur als Lokator sondern auch als Identifikator für Endgeräte fungieren.

(4) **Der Datenaustausch findet immer zwischen zwei Endgeräten Ende zu Ende statt.** Im heutigen Internet existiert eine große Anzahl von Zwischensystemen, welche in den Datenverkehr eingreifen können. Beispielsweise in Form von *Firewalls*. Zudem sind durch den Einsatz von Verfahren wie NAPT bzw. NAT (Network Address (Port) Translation) Geräte nicht mehr zwingend von jedem Punkt im Netz aus unter der gleichen Adresse erreichbar. Hinter einer Zieladresse können sich mehrere Endsysteme verbergen.

Ein grundlegendes Problem der Berkeley Socket-API ist die niedrige Abstraktionsebene auf der die API ansetzt. Dies führt dazu, dass Anwendungen und Anwendungsentwickler wissen müssen, welches Transportprotokoll in Verbindung mit welcher Protokollfamilie eingesetzt wird. Je nach eingesetztem Protokoll werden Anwendungsdateneinheiten zum Beispiel automatisch erhalten (UDP) oder müssen von der Anwendung selbständig aus einem Bytestrom wiederhergestellt werden (TCP). Auch Aufgaben wie die Namensauflösung, also zum Beispiel das Abbilden einer URI auf eine IP Adresse, und nicht zuletzt das Verwalten von über die Socket-API aufgebauten Verbindungen, finden innerhalb der Anwendungen also auf der Anwendungsschicht statt.

Im ISO/OSI Modell existiert oberhalb der Transportschicht die Sitzungsschicht. Ihre Aufgabe ist der Auf- und Abbau von Kommunikationsassoziationen zwischen Endgeräten. Eine Kommunikationsassoziation zwischen einem oder mehreren Endgeräten ist eine Art logische Verbindung zwischen diesen Geräten oberhalb der Transportschicht. Diese wird durch Transportverbindungen (z.B. via TCP) oder Transportassoziationen (z.B. via UDP) für die Dauer eines Datenaustausches aufrechterhalten. Man spricht auch von einer Sitzung, anstatt logischer Verbindung. Aufgrund der geänderten Anforderungen ist die Sitzungsverwaltung aber immer komplexer geworden. Anwendungen müssen heute häufig eigene Mechanismen implementieren um nach Transportverbindungsabbrüchen die Transportverbindung beziehungsweise -assoziation wiederherstellen zu können und auf Adressänderungen reagieren zu können. Dazu gehört ebenfalls die Verwaltung der dafür benötigten Transportverbindungszustandsdaten.

3.1.2 HTTP als neue Anwendungsschnittstelle

Neben der Berkeley Socket-API hat sich durch die weite Verbreitung des World Wide Web zunehmend auch das Anwendungsschichtprotokoll HTTP und sein *Object Model* als

Programmierparadigma für verteilte Anwendungen etabliert. Bereits in 2010 wurden ca. 50-60% des Endnutzer-generierten Datenverkehrs in Zugangsnetzen über HTTP ausgetauscht [Mai+09; Lab+10]. Ein Trend der durch [San12; Cis12a] bestätigt wird. Anteil daran haben neben reinen Web-Seiten vor allem auch Audio- und Video-(Streaming)-Daten, welche ca. 25-40% der HTTP-Daten ausmachen. Weitere 30% entfallen auf den Download von Dateien. Dies liegt nicht nur an der zunehmenden Popularität von Online-Video und Musik Diensten sondern auch darin begründet, dass sich neben einfachen Web-Diensten heute auch komplexe Web-Anwendungen über HTTP im Browser realisieren lassen. Internetbrowser wandeln sich damit von einer Software zum reinen Anzeigen von Web-Seiten, hin zu Ausführungsumgebungen für Plattform-übergreifende beziehungsweise -unabhängige Anwendungen. Das bedeutet Anwendungen müssen nur einmal erstellt werden und die gleiche Anwendung läuft unter verschiedensten Betriebssystemen und Hardwarearchitekturen, sofern ein HTTP Client in Form eines Browsers verfügbar ist.

Das Modell von HTTP wurde in [Fie00] als *Representational State Transfer* (REST) verallgemeinert und setzt auf einer höheren Abstraktionsebene als die Berkeley Socket-API an. Ein REST-konformer Dienst muss über eine eindeutige Adresse in Form eines *Uniform Resource Locator* (URL) erreichbar sein und ermöglicht es einer Anwendung direkt auf Ressourcen über einen *Uniform Resource Identifier* (URI) zuzugreifen. Dabei muss die unter einer URI verfügbare Ressource eindeutig sein, kann aber in verschiedenen Repräsentationsformen, wie zum Beispiel verschiedenen Sprachen oder Formaten (HTML, XML oder JSON), ausgeliefert werden. Wiederholte Anfragen müssen dabei jeweils das gleiche Ergebnis liefern.

REST-konforme Dienste bieten verschiedene Operationen zum Zugriff auf Ressourcen an, um diese beispielsweise auszuliefern oder zu verändern. Die Operationen selbst sind nicht fest definiert. In [PGS10] wird jedoch HTTP und seine definierten Methoden als De-Facto-Standard für REST-konforme Webdienste identifiziert. Wird über HTTP auf Ressourcen zugegriffen, so geben die verwendeten HTTP Methoden an, welche Operation eines Dienstes gewünscht ist. Mit *GET* kann zum Beispiel eine Ressource angefordert werden, während *PUT* eine neue Ressource erzeugt oder *POST* eine existierende Ressource aktualisiert beziehungsweise verändert. Der Fokus auf URIs und der direkte Zugriff auf Ressourcen über verschiedene Operationen, macht HTTP für viele Anwendungsfälle zu einer, im Vergleich mit der Berkeley Socket-API, intuitiveren Netzwerkabstraktion für Anwendungsentwickler. HTTP erfordert jedoch zwingend ein unterliegendes zuverlässiges Transportprotokoll, in der Regel handelt es sich dabei um TCP, zum Datenaustausch.

Besonders Google hat in diesem Bereich mit der Entwicklung von SPDY [BP12] und QUIC [Ham+16] die Entwicklung zweier Protokolle vorangetrieben, um die Dienstgüte beim Abrufen von Webseiten zu erhöhen. Bei SPDY handelt es sich um ein Protokoll zum effizienten Transport von Web Daten. Dabei werden die Daten komprimiert, aufgeteilt und parallel über mehrere Kanäle priorisiert übertragen. Auch die Erhöhung der Sicherheit ist

ein ausgeschriebenes Ziel von SPDY, welches inzwischen zu großen Teilen in den HTTP Nachfolger HTTP/2 [BPT15] eingeflossen ist und von Google selbst durch HTTP/2 ersetzt wurde. Damit einhergehend wurde auch ein Aushandlungsmechanismus eingeführt, welcher die Wahl der zu verwendenden HTTP Version aber auch potentielle nicht-HTTP Protokolle zum Datentransport zulässt.

Bei QUIC handelt es sich um ein experimentelles Transportprotokoll, welches für den Einsatz mit SPDY beziehungsweise HTTP/2 als Alternative zu TCP gedacht ist und unter anderem das bei TCP auftretende *Head-of-line blocking* Problem löst. Sowohl SPDY als auch QUIC zeigen, dass Bedarf an alternativen Transportdiensten mit unterschiedlichen Dienstgüteparametern besteht. Sie sind aber auf den Anwendungsfall Webdaten Transport zugeschnitten und hin optimiert. Daher haben Anwendungsentwickler, ähnlich wie bei der Berkeley Socket-API, wenig Spielraum bei der Wahl spezifischer Dienstgüteparameter für den Datentransport.

Durch den Einsatz alternativer Protokolle für den eigentlichen Datentransport könnten Anwendungen beispielsweise eine maximal tolerierbare Verzögerung oder eine minimal benötigte Datenrate anfordern, was im Web Kontext im Hinblick auf Technologien wie WebRTC [Alv16] interessant sein könnte, welches den direkten Datentransport zwischen Clients ohne den Umweg über den Webserver ermöglicht. Genauso können von SPDY angebotene Dienstgütemechanismen wie die Datenkompression oder -priorisierung im generischen Fall für nicht Web-basierte Anwendungen interessant sein.

3.2 Die PASTE-API

In Abschnitt 3.1 wurden zwei heute verwendete Anwendungsschnittstellen vorgestellt, welche sich auf verschiedenen Abstraktionsebene befinden und unterschiedliche Ansätze verfolgen. Während die Berkeley Socket API einer Anwendung Kommunikationsendpunkte zum Austausch von Daten bereit stellt, ermöglicht HTTP es einer Anwendung direkt auf Ressourcen zuzugreifen und bietet Anwendungsentwicklern damit eine intuitive Form des Datenaustausches an. Abbildung 3.1 zeigt die beiden Alternativen neben der im Folgenden beschriebenen PASTE-API. Diese wurde mit dem Ziel entworfen, sämtliche für den Datentransport und die Verwaltung von Kommunikationsbeziehungen spezifischen Protokolle und Mechanismen unterhalb der Anwendungsschnittstelle zu verlagern. Dazu gehört beispielsweise die Namensauflösung per DNS. Gleichzeitig soll die Unabhängigkeit von einem konkreten unterliegenden Adressraum gewährleistet sein.

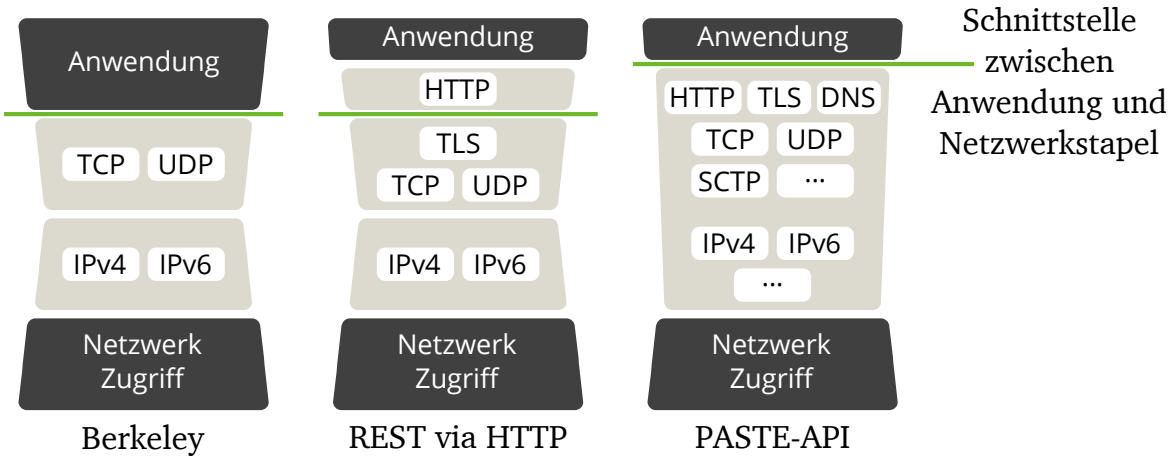


Abbildung 3.1: *Stundenglas Modell des Internets für die Berkeley Socket-API (links), HTTP (mitte) und PASTE-API (rechts).*

3.2.1 Anforderungen

Bereits 2010 wurde in „HTTP as the narrow waist of the future Internet“ [PGS10] ein erweitertes HTTP als Allzweckprotokoll für sämtliche Anwendungen vorgeschlagen. Die Autoren kamen zu dem Schluss, dass HTTP Methoden für jegliche Anwendungsfälle ausreichend sind, sofern diese um Datagramm-Transport Unterstützung ergänzt werden. Die PASTE-API greift daher Ansätze sowohl der Socket API als auch von HTTP auf. Im Folgenden werden weitere Ziele und sich daraus ergebende Anforderungen an die PASTE-API beschrieben. Um eine möglichst große Entkopplung zwischen Anwendung und Netzwerk zu erreichen, ist eine übergeordnete Anforderung an die Anwendungsschnittstelle die Reduzierung des auf der Anwendungsebene benötigten Wissens und gehaltenen Zustandes über das unterliegende Netzwerk.

Weitere Ziele sind:

- Eine einfache und intuitiv benutzbare Schnittstelle für den Anwendungsentwickler durch eine kleine Menge Dienstprimitive. Diese sollen unabhängig vom Kontext, in dem sie verwendet werden, eine konsistente Funktion erfüllen.

- Die ausschließliche Verwendung von eindeutigen Namen statt Adressen auf der Anwendungsschicht, um besser Unterstützung für Mobilität, Multi-homing und Multicast zu erreichen.

- Die Einführung Protokoll-agnostischer parametrierbarer Transportdienstbeschreibungen auf der Anwendungsschicht, zur Unterstützung automatischer Protokollselektion unterhalb der Anwendungsschicht.

3.3 Dienstprimitive der PASTE-API

Die durch HTTP definierten Methoden bzw. Dienstprimitive bilden eine gute Ausgangslage für eine Anwendungsschnittstelle, da diese Anwendungsentwicklern bekannt sind und sie somit das Kriterium erfüllen, dass eine gängige Erwartungshaltung an die von ihnen produzierte Antwort geknüpft ist.

Bereits im Rahmen des G-Lab Projektes und als Teil des NENA Rahmenwerkes [MVZ11], wurde in „A Future-Proof Application-to-Network Interface“ [MWB11] in Form der NENA-API eine Anwendungsschnittstelle auf dieser Basis veröffentlicht. Diese verwendet die HTTP entliehenen Dienstprimitive GET, PUT und CONNECT und erweiterte den Satz der Dienstprimitive um ein BIND und ein ACCEPT. Gewonnen wurde dieser Satz Dienstprimitive durch die Analyse verschiedener Netzwerkarchitekturen und ihrer Anwendungsschnittstellen. Berücksichtigt wurden dabei insbesondere das heutige Internet, Content Centric Networking (CCN) [Jac+09a], die Services Integration controL and Optimization (SILO) [Dut+07] Architektur sowie die Recursive Network Architecture (RNA) [TP08]. Des Weiteren wurde die Abbildbarkeit der Dienstprimitive auf verschiedene Kommunikationsmuster wie *publish/subscribe* oder *request/response* untersucht, welche beispielhaft in Abschnitt 3.6 umgesetzt sind.

Die Primitiven der NENA-API sind in allen Fällen ausreichend, in denen Client- und Serverseite einer verteilten Anwendung parallel (und häufig vom gleichen Entwickler) entwickelt werden. Für viele verteilte Anwendung, insbesondere proprietäre Software, ist dies der Regelfall. Es gibt jedoch auch eine weitere Klasse von verteilten Anwendungen, bei der verschiedene Clients und Server von mehreren Entwicklern unabhängig voneinander entwickelt werden. Beispielweise existieren verschiedenste Web- oder Email-Server und eine jeweils ungleich größere Anzahl an Web Browsern und Email-Clients. In diesen Fällen sind die Dienstprimitive der NENA-API nicht ausreichend, da keine Möglichkeit besteht vor einem Kommunikationsassoziationsaufbau Informationen über den entfernten Endpunkt abzufragen.

Die PASTE-API führt daher aufbauend auf der NENA-API weitere Dienstprimitive ein und stellt somit eine erweiterte Version der NENA-API dar. Tabelle 3.1 zeigt eine vollständige Übersicht aller Dienstprimitive der PASTE-API, ihrer wichtigsten Parameter und ihre Klasse. Die Dienstprimitive lassen sich dabei in drei Klassen einteilen:

- Assoziationsaufbau: Dies beinhaltet das Erzeugen eines Kommunikationsendpunktes sowie den Aufbau einer Kommunikationsassoziation zwischen Anwendungsinstanzen einer oder verschiedener Anwendungen. Dabei werden für alle beteiligten Anwendungsinstanzen Kommunikationsendpunkte in Form von HANDLES bereitgestellt, auf denen weitere Operationen ausgeführt werden können.

Tabelle 3.1: *Überblick über sämtliche Dienstprimitive der PASTE API.*

Primitive	**Parameter**	**Rückgabewert**	**Klasse**
GET	URI, Transportdienst	HANDLE	Assoziationsaufbau
PUT	URI, Transportdienst	HANDLE	Assoziationsaufbau
CONNECT	URI, Transportdienst	HANDLE	Assoziationsaufbau
BIND	URI, DienstprimitiveA, Transportdienst(e), ..., DienstprimitiveB, Transportdienst(e), ..., ...	HANDLE	Assoziationsaufbau
ACCEPT	Ereignis	HANDLE	Assoziationsaufbau
RESUME	AssoziationsID	HANDLE	Assoziationsaufbau
OPTIONS	URI	Dienstprimitive- und Transportdienstliste	Dienstinformationen abfragen
HEAD	URI, Dienstprimitive, Transportdienst	Anwendungsdienst spezifische Namensparameter	Dienstinformationen abfragen
TRACE	URI, Dienstprimitive, Transportdienst *oder* AssoziationsID	Transportprotokoll- und Pfadinformationen	Dienstinformationen abfragen
QUERY	SitzungsID	AssoziationsID Liste	Sitzungsverwaltung
RESUME	SitzungsID		Sitzungsverwaltung

- Dienstinformation: Informationen über angebotene Dienstprimitive und darüber verfügbare Transportdienste von entfernten Kommunikationsendpunkten können abgefragt werden. Dies kann sowohl vor dem Aufbau einer Kommunikationsassoziation geschehen als auch für existierende Kommunikationsassoziationen durchgeführt werden, um beispielsweise Änderungen im Netz festzustellen.

- Sitzungsverwaltung: Ermöglicht die Abfrage von zu einer Anwendungsinstanz gehörenden offenen Kommunikationsassoziationen aus einer vorangegangenen Sitzung. Beispielsweise bei der Wiederaufnahme einer zuvor pausierten Kommunikationsassoziation.

Abbildung 3.2 zeigt einen Überblick über die einen HANDLE erzeugenden Dienstprimitive. Diese Dienstprimitive lassen sich wie in Abbildung 3.2 dargestellt in *Client* und

Server seitige Rollen unterteilen. Dabei nimmt eine von zwei kommunizierenden Parteien in der Regel die Seite des initialen Dienstanbieters ein. Während die andere Seite einen Dienst konsumieren möchte und aktiv den Aufbau einer Kommunikationsbeziehung mit dem Dienstanbieter anstößt.

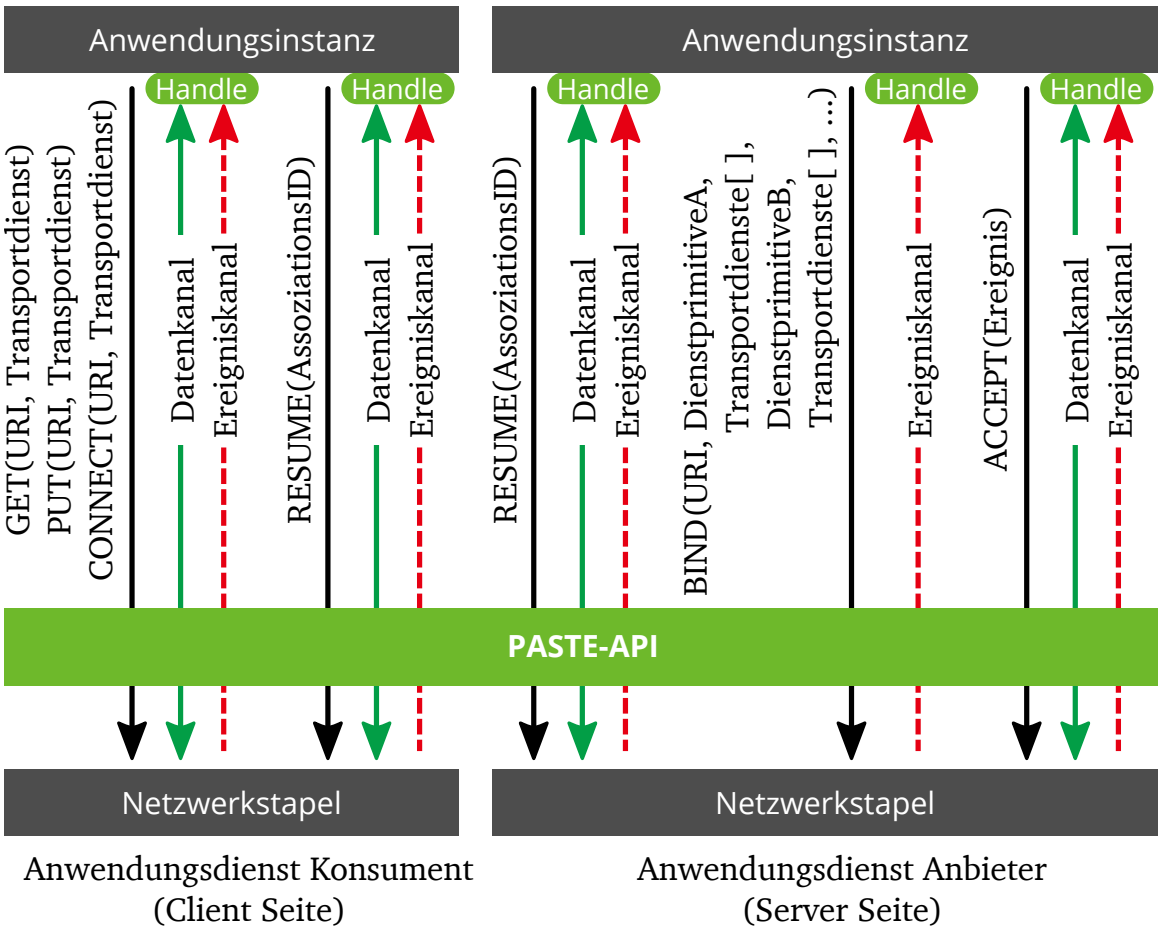


Abbildung 3.2: *Dienstprimitive der PASTE-API zur HANDLE Erzeugung. Aufgeteilt in Client- und Server-seitige Befehle.*

Per GET, PUT oder CONNECT erzeugen Anwendungen einen Endpunkt beziehungsweise HANDLE und bauen eine Kommunikationsassoziation zwischen zwei Anwendungsinstanzen auf. Alle drei Dienstprimitive werden dabei mit einem *Uniform Resource Identifier* und dem gewünschten Transportdienst als Parameter aufgerufen. *Uniform Resource Identifiers* (URIs) wurden dabei als Namen gewählt, da diese im *Word Wide Web* bereits etabliert und geläufig sind. Sie erfüllen das Kriterium der Eindeutigkeit und lassen sich leicht erweitern. Ihre genaue Verwendung wird in Abschnitt 3.4 erläutert. Der gewünschte Transportdienst wird in Form von einer Transportdienstbeschreibung spezifiziert. Auf diese wird in Abschnitt 3.5 näher eingegangen.

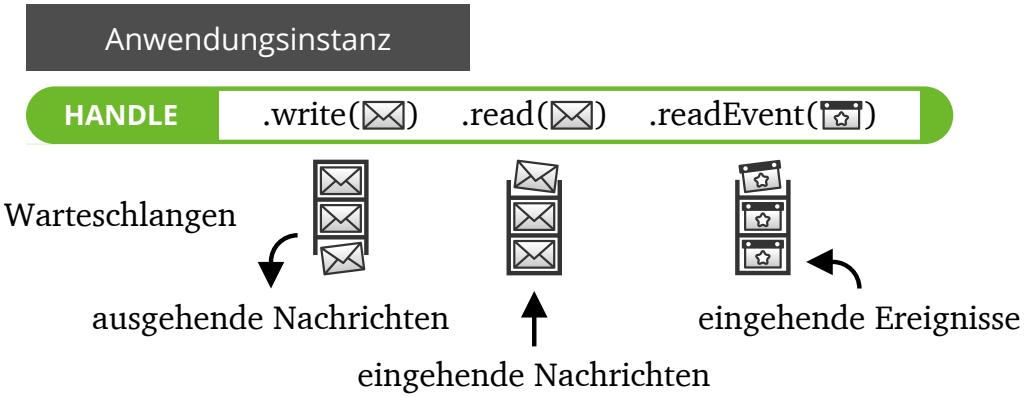


Abbildung 3.3: *Mit einem HANDLE assoziierte Warteschlangen und die darauf lesend und schreibend operierenden HANDLE-Methoden.*

Abhängig von der verwendeten Dienstprimitive und dem gewählten Transportdienst ermöglicht der erzeugte Endpunkt den Zugriff auf unterschiedliche Ressourcen. Dabei kann es sich um direkt zurückgelieferte konkrete Inhalte handeln, oder aber auch ein durch eine Anwendung bereitgestellter Dienst genutzt oder ein bidirektionaler (interaktiver) Datenaustausch angestoßen werden. In jedem Fall wird ein HANDLE, welcher in Abbildung 3.3 dargestellt ist, an die aufrufende Anwendungsinstanz zurückgeliefert. Mit diesem ist jeweils eine Ereigniswarteschlange für eingehende Benachrichtigungen assoziiert, sowie mindestens eine Warteschlange für den Erhalt oder das Versenden von Nachrichten. Die dadurch erreichte Trennung zwischen Daten- und Ereigniskanal ist notwendig, da Ereignisse und darüber ausgetauschte Signalisierungsnachrichten zwischen Anwendungsinstanzen grundsätzlich zuverlässig übertragen werden. Während der Datenkanal je nach angefordertem Transportdienst unterschiedliche Dienstgüteeigenschaften aufweisen kann.

Bevor eine Kommunikationsassoziation aufgebaut werden kann, wird zunächst ein Endpunkt benötigt unter dem ein Dienst angeboten wird. Im Falle von HTTP steht in der Regel ein Webserver auf der Gegenseite zur Verfügung, um eingehende Anfragen entgegenzunehmen und zu bearbeiten. Dieser ist dafür verantwortlich, dass unter einer bestimmten URI ein Webdienst oder ein Inhalt verfügbar gemacht wird. Da die PASTE-API keine spezifischen Transportprotokolle oder Netzwerkarchitekturen voraussetzt, können beliebige Anwendungsinstanzen beliebige Ressourcen anbieten. Per BIND bindet eine Anwendungsinstanz die von ihr angebotenen Inhalte oder Anwendungsdienste an eine URI und erstellt so einen Endpunkt, unter dem sie erreichbar ist. Zusätzlich zur URI werden als Parameter eine Liste der unterstützten Dienstprimitive übergeben. Pro unterstützter Dienstprimitive wird zudem eine Liste aller Transportdienste in Form von einer oder mehreren Transportdienstbeschreibungen übergeben, welche über diese Dienstprimitive verfügbar sind (vgl. Abbildung 3.2). Ähnlich der Berkeley Socket-API wird von BIND nur

ein HANDLE mit einem Ereigniskanal zurückgeliefert, über den die Anwendungsinstanz über eingehende Verbindungsanfragen informiert wird.

Per ACCEPT Aufruf kann eine Anwendungsinstanz für eine eingehende Kommunikationsassoziationsanfrage ein dediziertes Daten- und Ereigniskanal Paar erstellen. Dazu wird als Parameter das Ereignis übergeben, welches über einen per BIND gebundenen HANDLE erhalten wurde, sofern die Anwendungsinstanz eine Kommunikationsassoziation annehmen möchte.

Die letzte in Abbildung 3.2 dargestellte Dienstprimitive ist RESUME. Die PASTE-API unterstützt das Pausieren und spätere Wiederaufnehmen von Kommunikationsassoziationen. Mit RESUME kann eine zuvor pausierte Kommunikationsassoziation unter Angabe eines zugehörigen Assoziationsidentifikators wiederaufgenommen werden. Diesen erhält die Anwendungsinstanz vom PASTE Rahmenwerk, welches auch den damit verbundenen Assoziationszustand auf allen beteiligten Endsystemen verwaltet. Auf Assoziationsidentifikatoren und -zustände wird in Kapitel 4.5 genauer eingegangen. Der RESUME Dienstprimitive kann keine eindeutige Client- oder Serverrolle zugeordnet werden, da alle an einer Kommunikationsassoziation beteiligten Anwendungsinstanzen das Pausieren und demnach auch Wiederaufnehmen einer Kommunikationsassoziation anstoßen können.

Abbildung 3.4 zeigt die Dienstprimitive der PASTE-API zur Client-seitigen Abfrage von Dienstinformationen. Nicht jede Anwendungsinstanz, und damit jeder Endpunkt, unterstützt zwingend alle Dienstprimitive. GET sollte ausschließlich für einen lesenden Zugriff auf Ressourcen verwendet werden, während PUT impliziert, dass eine Ressource angelegt oder modifiziert werden soll. Beide Dienstprimitive erfordern im Gegensatz zu CONNECT nur einen unidirektionalen Datenkanal. Da nicht für alle Anwendungsfälle beziehungsweise -dienste alle Dienstprimitive sinnvoll verwendbar sind, können mit OPTIONS an einem entfernten Endpunkt verfügbare Dienstprimitive abgefragt werden, sofern sie der anfragenden Anwendung nicht bereits bekannt sind. Dazu wir die URI als Parameter übergeben, unter der der entfernte Endpunkt erreichbar ist. Als Antwort auf eine OPTIONS Anfrage erhält eine Anwendungsinstanz sämtliche zuvor per BIND von der Gegenseite angebotenen Dienstprimitive und die jeweils für eine Dienstprimitive verfügbaren Transportdienste zurück.

Mit Hilfe von HEAD lassen sich weitere Informationen zu einem angebotenen Anwendungsdienst abfragen, in Form von Anwendungsdienst bezogenen Parametern wie beispielsweise die Auswahl verfügbarer Formate für angebotene Inhalte. Ein Dokument kann zum Beispiel im Klartext, XML codiert oder als PDF vorliegen. Eine Anwendung kann daraufhin das gewünschte Format als Teil der URI anfordern. Diese Informationen sind spezifisch für den konkreten von einer Anwendung angebotenen Anwendungsdienst. Anwendungsinstanzen, welche einen Dienst anbieten, müssen daher per URI verfügbare Parameter registrieren, damit eingehende HEAD Anfragen beantworten werden können. Parameter können für eine URI jeweils pro Dienstprimitive, welche per BIND von einer

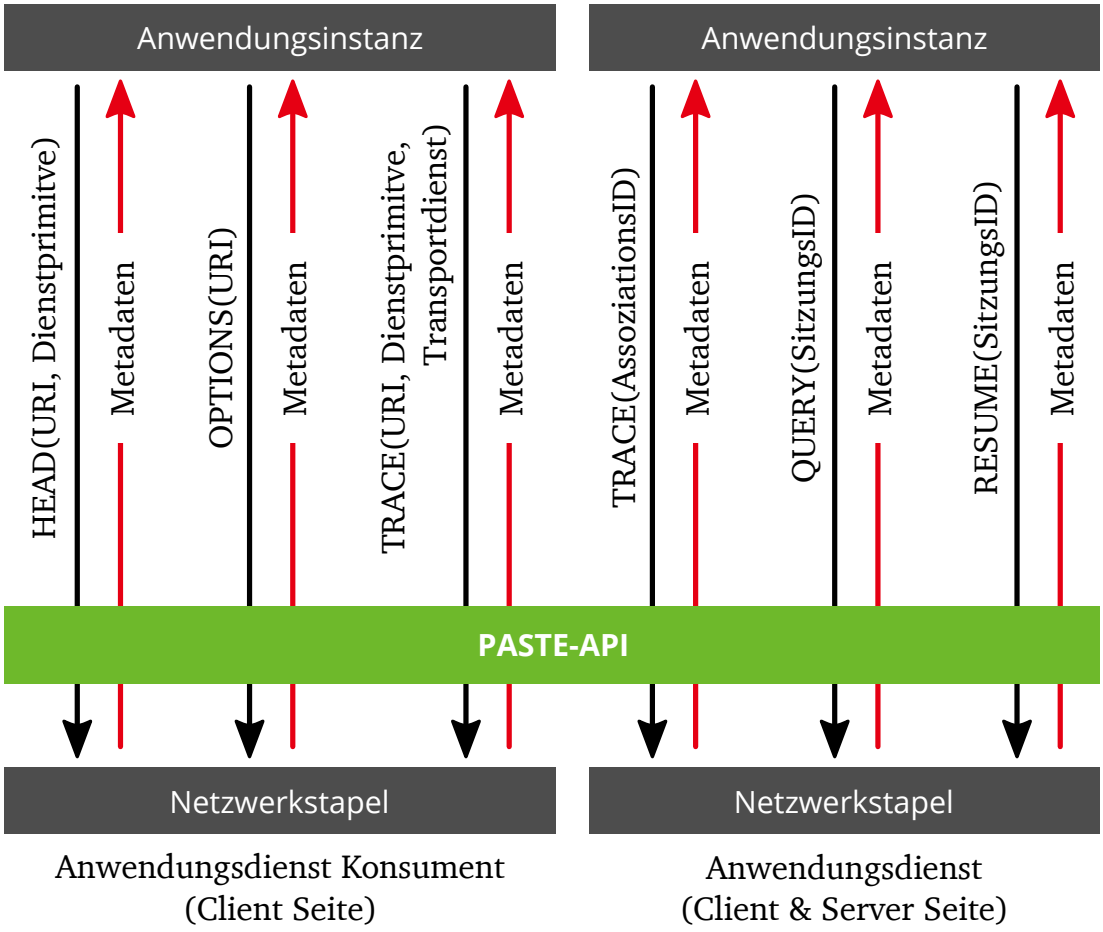


Abbildung 3.4: *Dienstprimitive der PASTE-API zur Client-seitigen Abfrage von Dienstinformationen, sowie der Abfrage von Sitzungsinformationen.*

Anwendungsinstanz angeboten wird, hinterlegt werden. Dies wird in Abschnitt 3.5 näher erläutert.

Um Anwendungen die Möglichkeit zu geben ihren gewünschten Transportdienst aufgrund von Netz-spezifischen Bedingungen anzupassen, können diese per TRACE für eine bestimmte URI, einen gegeben Transportdienst und eine konkrete Dienstprimitive Informationen abrufen. Im einfachsten Fall wird das für die Kombination aus URI, Transportdienst und Dienstprimitive vom unterliegenden Netz aktuell gewählte Transportprotokoll zurückgeliefert. Zusätzlich können im Netz zum Beispiel Informationen über auf dem Datenpfad befindliche *Middleboxes* gesammelt werden, welche die ursprüngliche Anfrage modifizieren. Hierfür existieren verschiedene Lösungen wie beispielsweise Tracebox [Det+13; Til] oder das HICCUPS Protokoll [CBA14], deren Information an die Anwendung weitergereicht werden können. Dazu wird allerdings Unterstützung im Netzwerk benötigt. Das in Kapitel 4 vorgestellte PASTE Rahmenwerk sammelt entspre-

chende Statusinformationen für einzelne Kommunikationsassoziationen, um diese einer Anwendung zur Verfügung zu stellen.

Die TRACE Primitive bildet also das Gegenstück zur HEAD Primitive. Beide Dienstprimitive liefern Informationen zu einem über eine URI erreichbaren Dienst bei der Verwendung einer bestimmte Dienstprimitive, also GET, PUT oder CONNECT. Per HEAD Primitive werden grundsätzlich Informationen oberhalb der Transportschicht zurückgeliefert, welche abhängig vom angebotenen Anwendungsdienst sind. Per TRACE Primitive kann eine Anwendungsinstanz im Gegenzug Informationen der Transportschicht und unterliegender Schichten direkt abfragen. Grundsätzlich soll die PASTE-API genau diese Informationen für die Anwendung transparent behandeln. Jedoch sind Fälle denkbar, in denen Anwendungen auf diese Information angewiesen sind. So kann es beispielsweise rechtliche Vorgaben geben, die festlegen welche Protokolle für eine Klasse sicherheitskritischer Anwendungen als robust und verlässlich genug angesehen werden.

Statt für eine URI, eine Dienstprimitive und einen Transportdienst kann TRACE auch für eine bereits existierende Kommunikationsassoziation mit dem zugehörigen Assoziationsidentifikator aufgerufen werden, um die gleichen Informationen zu erhalten. Dies gibt Anwendungen die Möglichkeit auch laufende Kommunikationsassoziationen zu beobachten. Anwendungen können dann bei Bedarf eine neue Kommunikationsassoziation mit geänderten Diensteigenschaften anbieten oder aufbauen. Zum Beispiel, wenn sich der Kontext eines an der Kommunikationsassoziation beteiligten Endsystems ändert oder die ursprünglich gewünschten Dienstgüteeigenschaften nicht mehr vollständig erbracht werden. Das ist beispielsweise beim Wiederaufnahmen einer Kommunikationsassoziation nach einer längeren Zeit sinnvoll, wenn inzwischen ein anderer Transportdienst als der zuvor gewählte besser geeignet ist, da eines oder beide der beteiligten Endsysteme sich inzwischen in einer anderen Netzdomäne befinden.

Die letzten, ebenfalls in Abbildung 3.4 dargestellten, Dienstprimitive der PASTE-API sind QUERY und erneut RESUME. Mit Hilfe von QUERY können Anwendungsinstanzen eine Liste aller ihnen zugehörigen Kommunikationsassoziationen von ihrem lokalen PASTE Rahmenwerk erhalten. Diese erhalten sie in Form einer Liste von Assoziationsidentifikatoren. Dazu muss ein Sitzungsidentifikator (oder auch SitzungsID) übergeben werden, der lokal gültig ist und vom PASTE Rahmenwerk für jede Anwendungsinstanz vergeben wird. Abbildung 3.5 zeigt den Zusammenhang zwischen Sitzungsidentifikatoren und Assoziationsidentifikatoren (oder auch AssoziationsIDs). Sämtliche Assoziationen einer Anwendungsinstanz werden von PASTE intern unter einem lokal gültigem Sitzungsidentifikator zusammengefasst und verwaltet. Dieser wird initial beim ersten Erzeugen eines Kommunikationsendpunktes beziehungsweise HANDLES vom PASTE Rahmenwerk generiert und kann über jeden HANDLE abgefragt werden. Anwendungsinstanzen werden dabei anhand ihres Prozesses im Betriebssystem identifiziert. Möchte eine Anwendungsinstanz zu einem späteren Zeitpunkt eine vorherige Sitzung wiederaufnehmen, so kann

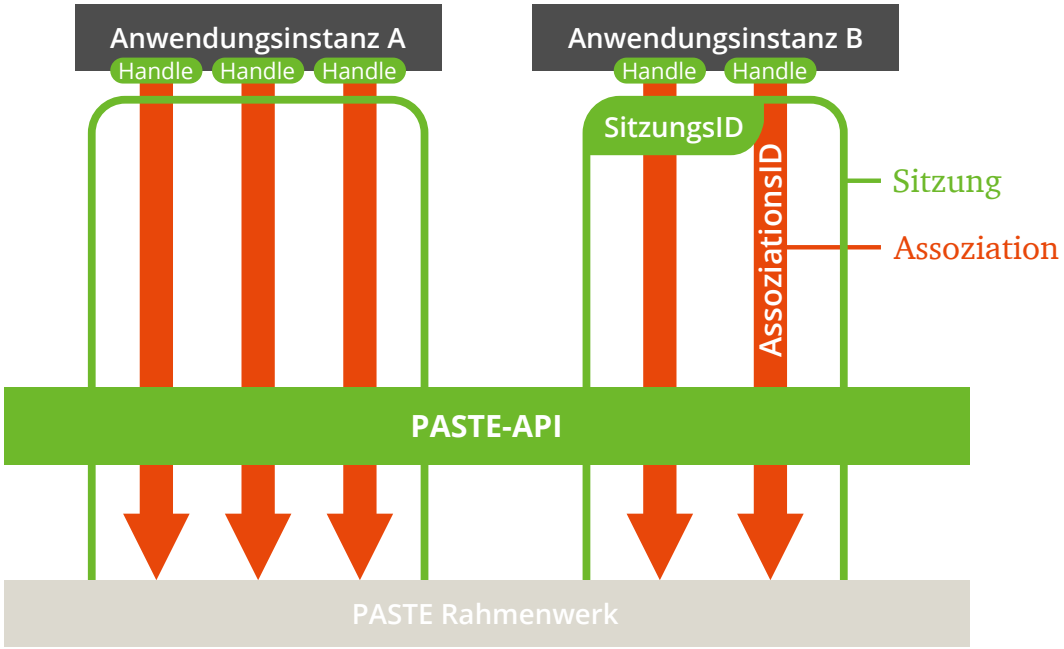


Abbildung 3.5: *Sitzungsidentifikatoren und Assoziationsidentifikatoren der PASTE-API beziehungsweise des PASTE Rahmenwerkes.*

diese RESUME mit dem zuvor erhaltenen Sitzungsidentifikator aufrufen. Damit wird dem PASTE Rahmenwerk signalisiert, dass kein neuer Sitzungsidentifikator generiert werden soll. Neue erzeugte HANDLES dieser Anwendungsinstanz werden dann, genau wie bereits bestehende HANDLES, dem existierenden Sitzungsidentifikator zugeordnet. Jede Kommunikationsassoziation besitzt einen global gültigen Assoziationsidentifikator und ist auf jedem System zu jedem Zeitpunkt genau einer Sitzung zugeordnet. Auf Sitzungsidentifikatoren wird genau wie auf Assoziationsidentifikatoren in Kapitel 4.5 detaillierter eingegangen.

3.3.1 HANDLE Methoden

Wie bei der Berkeley Socket-API handelt es sich bei der PASTE-API um eine funktionsorientierte Programmierschnittstelle. GET, PUT, CONNECT, RESUME und ACCEPT erzeugen wie bereits erwähnt HANDLES. Über diese findet der eigentliche Datenaustausch statt. Abbildung 3.3 zeigt bereits die zum Lesen und schreiben von Daten und Ereignissen definierten HANDLE Methoden. Eine vollständige Liste aller Methoden, ihrer Parameter und ihrer Aufgabe ist in Tabelle 3.2 zu sehen.

Die Methoden `read(...)` und `write(...)` lesen am HANDLE verfügbare Nachrichten beziehungsweise übergeben zu sendende Nachrichten an das PASTE Rahmenwerk. Diese werden über den logischen Datenkanal, also die Empfangs- oder Sendewarteschlange zwi-

Tabelle 3.2: *Überblick sämtlicher HANDLE Methoden.*

Methode	**Parameter**	**Aufgabe**
read	Application Data Unit	Empfangene Daten werden in Form von Nachrichten an die Anwendung gereicht.
write	Application Data Unit	Versenden von Daten in Form von Nachrichten.
mayRead		Überprüfen ob über ein HANDLE (noch) Daten empfangen werden können.
mayWrite		Überprüfen ob über ein HANDLE (noch) Daten gesendet werden können.
readEvent	Event Data Unit	Es wird das nächste Ereignis aus der Ereigniswarteschlange des HANDLES gelesen.
hasPendingEvents		Überprüfen ob ein HANDLE ausstehende Ereignisse hat.
close		Schließen einer Assoziation (inklusive sauberem Assoziationsabbau).
terminate		Beenden einer Assoziation ohne Bestätigung von Zwischensystemen und anderem Endsystem.
suspend		(Temporäres) Schlafenlegen einer Assoziation.
link	HANDLE	Zum gleichen Datenstrom gehörende Assoziationen miteinander verknüpfen.
unlink	HANDLE	Verknüpfte Assoziationen wieder trennen.
getAssociationID		Abfragen des Assoziationsidentifikators.
getSessionID		Abfragen des Sitzungsidentifikators.
addParameters	Namens Parameter Beschreibung	Für einen mit BIND registrierten Namen verfügbare Parameter angeben.

schen Anwendungsinstanz und PASTE Rahmenwerk ausgetauscht. Der Datenkanal dient also nur dem Austausch der reinen Nutzdaten mit den zuvor angeforderten Diensteigenschaften des beim Erstellen des HANDLES gewählten Transportdienstes. Der Ereigniskanal hingegen ist beispielsweise für Fehlermeldungen oder eingehende Anfragen zum Aufbau einer Kommunikationsassoziation reserviert. Die Methoden `mayRead()` und `mayWrite()` werden respektive verwendet, um festzustellen ob aktuell auf einem HANDLE lesend oder schreibend zugegriffen werden kann.

Für das Lesen von Ereignissen vom Ereigniskanal ist analog zu `read(...)` zusätzlich die Methode `readEvent(...)` spezifiziert. Mit dieser werden eingehende Fehlermeldungen, Statusinformationen oder Verbindungsanfragen abgerufen um anschließend von der Anwendungsinstanz bearbeitet zu werden. Analog zu `mayRead()` wird die Methode `hasPendingEvents()` verwendet um zu überprüfen, ob am HANDLE noch nicht abgearbeitete Ereignisse vorliegen.

Zum endgültigen Beenden einer Kommunikationsassoziation, wird auf einem HANDLE die Methode `close()` aufgerufen. Neben der `close()` Methode lassen sich Kommunikationsassoziation auch per `terminate()` beenden. Der Unterschied besteht darin, dass `close()` versucht Kommunikationsassoziationen ordnungsgemäß abzubauen, während `terminate()` Kommunikationsassoziationen direkt beendet, ohne beispielsweise auf noch ausstehende Daten zu warten. Dies kann sinnvoll sein, um andere Endsysteme schneller zu benachrichtigen, wenn eine Anwendungsinstanz beziehungsweise ein System gezwungen ist eine Kommunikationsassoziation abzubrechen. Beispielsweise kurz vor dem Verlassen des Empfangsbereichs bei einem Smartphone. In diesem Fall brauchen die anderen Endsysteme nicht auf den Ablauf eines *Timeouts* warten und können das Ereignis direkt an ihre Anwendungsinstanzen weiterleiten, die entsprechend darauf reagieren können.

Über die Methode `suspend()` kann dem entfernten Endsystem signalisiert werden, dass eine Anwendungsinstanz eine Kommunikationsassoziation pausieren möchte. Die Methode bildet das Gegenstück zur RESUME Dienstprimitive, mit der eine auf diese Weise schlafen gelegte Kommunikationsassoziation später wiederaufgenommen werden kann. Wie dies vom PASTE Rahmenwerk realisiert wird, ist in Kapitel 4.5 beschrieben.

Mit der Methode `link(HANDLE)` können zwei oder mehrere Kommunikationsassoziationen gekoppelt werden. Das heißt dem PASTE Rahmenwerk wird mitgeteilt, dass mehrere Kommunikationsassoziationen Teil des gleichen Datenaustausches sind. Anwendungen machen damit kenntlich, dass verschiedene Datenströme gleiche oder ähnliche *Quality of Service* Bedingungen im Netz erfahren sollen. Beispielsweise bei getrennten Datenströmen für Bild und Ton des gleichen Videos ist es sinnvoll, eine Kommunikationsassoziation abzubrechen, wenn die Andere aufgrund eines Fehlers beendet wird. Gleichzeitig kann im Netzwerk beispielsweise auf Protokolle wie SCTP mit *Multistreaming* Unterstützung zurückgegriffen werden, um diese Kommunikationsassoziationen umzusetzen. Selbst bei der Übertragung beider Ströme über getrennte Transportverbindungen, kann das Netzwerk besser auf Verbindungsabbrüche oder Fehler in einem der beiden Datenströme reagieren. HANDLES gekoppelter Kommunikationsassoziationen erhalten vom PASTE Rahmenwerk Ereignisse über Fehler, welche bei gekoppelten Kommunikationsassoziationen aufgetreten sind. Mit der Methode `unlink(HANDLE)` lassen sich zuvor gekoppelte Kommunikationsassoziationen wieder entkoppeln.

Über die Methoden `getAssociationID()` und `getSessionID()` fragen Anwendungsinstanzen die zu einem HANDLE gehörenden Identifikatoren ab. Jede Kommunikationsassoziation und damit jeder HANDLE hat einen eindeutigen Assoziationsidentifikator, während alle HANDLES für die Laufzeit einer Anwendungsinstanz den gleichen Sitzungsidentifikator zugewiesen bekommen.

Zuletzt existiert die `addParameters(...)` Methode, mit der Anwendungsinstanzen am HANDLE Informationen über unterstützte URI Parameter hinlegen. Dazu wird eine Beschreibung der Parameter übergeben und im PASTE Rahmenwerk hinterlegt, welche mittels HEAD Dienstprimitive später von anderen Anwendungsinstanzen abgerufen werden kann. Die übergebene Parameterbeschreibung besteht aus einem *JSON Array* von Parameterbeschreibungsobjekten. Auflistung 1 zeigt den Aufbau eines einzelnen Parameterbeschreibungsobjektes.

Auflistung 1: *Aufbaue eines Parameterbeschreibungsobjektes in JSON Notation.*

```
{
  "name"        : "(parameter name)",
  "type"        : "(parameter type)",
  "description" : "(human readable description")
}
```

Es besteht aus dem Parameternamen (*name*), dem Typ des Parameters (*type*) und einer von Menschen lesbaren Beschreibung des Parameters (*description*). Liste 3.3 zeigt eine Übersicht über alle definierten Parametertypen, welche den trivialen Datentypen der Programmiersprache Java entsprechen und zusätzlich um den Typ *String* erweitert wurden.

Da viele Programmiersprachen unterschiedliche Datenstromobjekte zum Beispiel in Form von Dateideskriptoren oder Byteströmen anbieten, ist es sinnvoll die HANDLE Methoden individuell durch Sprach spezifische Befehle zu ersetzen. Grundsätzlich wird von der Anwendungsschnittstelle nur jeweils die Methode und ihre Aufgabe vorgegeben. Dadurch lassen sich individuelle Implementationen der Schnittstelle leichter sprachkonform gestalten.

3.3.2 Umsetzung von Kommunikationsmustern

Namen dienen in der PASTE-API zum Identifizieren und Lokalisieren von Ressourcen und Anwendungsdiensten. Die Transportdienstbeschreibungen geben dagegen an, welche Merkmale der Datentransport zwischen Anwendungsinstanzen aufweisen soll, also wie Ressourcen ausgetauscht oder Anwendungsdienstdaten übertragen werden sollen.

Tabelle 3.3: *Überblick sämtlicher Parameter Typen.*

Typ	**Beschreibung**
byte	Eine 8-bit große Ganzzahl.
short	Eine 16-bit große Ganzzahl.
int	Eine 32-bit große Ganzzahl.
long	Eine 64-bit große Ganzzahl.
float	Eine 32-bit große Fließkommazahl.
double	Eine 64-bit große Fließkommazahl.
char	Ein 16-bit Unicode Zeichen.
boolean	Entweder *true* oder *false*.
string	Eine beliebige Folge von Zeichen.

Die Aufgabe der Dienstprimitive in der PASTE-API ist es, Anwendungen die Umsetzung verschiedener Kommunikationsmuster zu ermöglichen. Die Dienstprimitive OPTIONS, HEAD und TRACE werden dabei als *Remote Procedure Call* umgesetzt und müssen von jedem per BIND erstellten Endpunkt unterstützt werden. Die Verfügbarkeit von GET, PUT und CONNECT hingegen ist abhängig vom angebotenen Anwendungsdienst. Die genaue Funktion von GET, PUT und CONNECT kann dabei abhängig vom Namensraum oder einem bestimmten Anwendungsdienst variieren. Grundsätzlich spezifizieren sie immer, ob es sich um einen uni- oder bidirektionalen Datenaustausch handelt und in welche Richtung die Daten einer Kommunikationsassoziation fließen: Vom Anwendungsdienstanbieter zum Anwendungsdienstnehmer oder umgekehrt. Anwendungsentwickler haben die Möglichkeit, für verschiedene Dienstprimitive verschiedene Transportdienste festzulegen, welche unter dem gleichen Namen verfügbar sind. Dadurch können unterschiedliche Abläufe wie beispielsweise das Verteilen von Daten oder das Abonnieren eines Anwendungsdienstes auf die unterschiedlichen Primitiven aufgeteilt werden. In Abschnitt 3.6 wird die Abbildung der Dienstprimitive auf unterschiedliche Kommunikationsmuster anhand einiger Beispiele veranschaulicht.

Sollten zukünftige Netzwerkarchitekturen gänzlich neue Kommunikationsmuster und -paradigmen anbieten, für die keine sinnvolle Abbildung auf die existierenden Dienstprimitive besteht, ist auch die Einführung neuer Dienstprimitive denkbar. Sollte eine neuere Version der PASTE-API irgendwann zusätzliche Dienstprimitive unterstützen, können Anwendungen diese per BIND anbieten und alternative Anwendungsdienste über die neuen Dienstprimitive verfügbar machen. Zu dem Zeitpunkt bereits existierende Anwendungen verwenden dann weiterhin GET, PUT und CONNECT, während neue An-

wendungen mit Hilfe von OPTIONS die Unterstützung neuerer Dienstprimitive bei einem Anwendungsdienst abfragen und diese alternativ verwenden können.

3.4 Namensschemata

Zur eindeutigen Identifizierung von Ressourcen verwendet die PASTE-API auf der Anwendungsebene ausschließlich *Uniform Resource Identifier* (URI) [BFM05]. Diese haben ihren Ursprung im *World Wide Web* und werden heute bereits vielfältig verwendet. Als eindeutiger Name für einen bestimmten Inhalt, als Lokator von Anwendungsdiensten aber auch für Daten wie Telefonnummern, *E-Mail* Adressen oder Sozialversicherungsnummern. Selbst physische Objekte wie gedruckte Bücher oder abstrakte Konzepte wie mathematische Operatoren oder numerische Werte sind Ressourcen im Sinne von URIs. Neben ihrer universellen Einsetzbarkeit ist ein weiterer Vorteil, das mit dem *Domain Name System* bereits ein Namensdienst zur Auflösung von URIs auf IP Adressen existiert. Auch andere zukünftige Netzwerkarchitekturen wie beispielsweise *Content Centric Networking* verwenden URIs zur eindeutigen Identifizierung und zum Zugriff auf Ressourcen im Internet.

3.4.1 Aufbau

Der grundlegende Aufbau einer URI ist wie folgt definiert:

```
URI = Schema ':' [Anbieter] Pfad ['?' Anfrage] ['#' Fragment]
```

Nur das *Schema* und der *Pfad* müssen zwingend vorhanden sein, die restlichen Elemente sind optional. Durch das *Schema* wird häufig ein bestimmtes Protokoll impliziert (z.B. *http* oder *ftp*) oder der Typ festgelegt (z.B. *E-Mail* Adresse *mailto* oder eine Datei *file*). Damit lassen sich auch *Schemas* für bestimmte Typen von Ressourcen definieren wie zum Beispiel:

```
video://netzfliks.de/serien/karlsruhe_tag_und_nacht
audio://www.radioeins.de/podcast/sanft_und_sorgfältig/sus_20150209.mp3
services://wetter/deutschland?time=next_three_days
```

Durch die Nutzung von URI Schemas wissen sowohl Anwendung als auch Netzwerk anhand des Namens, welche Art von Daten übertragen werden sollen, ohne auf der Anwendungsebene ein konkretes Protokoll vorzugeben. Ebenso sind *Schemas* für einzelne Anwendungen oder Anwendungsdienste eines bestimmten Anbieters denkbar. Eine zusätzlich bei der PASTE-API mit übergebene Transportdienstbeschreibung, welche an die unterliegenden Schichten übergeben wird, dient zusammen mit dem Ressourcen Typ

der automatischen Wahl geeigneter Protokolle. Darauf wird in Kapitel 4 näher eingegangen, in dem die Verarbeitung der Transportdienstbeschreibungen durch das PASTE Rahmenwerk beschrieben wird.

Auf das *Schema* folgen ein optionaler *Anbieter* und der *Pfad* der Ressource. Beim *Anbieter* handelt es sich beispielsweise um den Server, der die Ressource bereithält. Damit dient der *Anbieter* als Lokator eines Systems im Netz, während der *Pfad* den genauen Speicherort einer Ressource auf einem System spezifiziert. Darauf folgt optional, abgetrennt durch ein Fragezeichen, der *Anfrageteil* in Form von einem oder mehreren Schlüssel Wert Paaren. Damit lassen sich Ressourcenanfragen weiter konkretisieren, indem zum Beispiel eine bestimmte Repräsentation gewählt werden kann. Per *Fragment*, abgetrennt durch ein Doppelkreuz, kann letztendlich ein bestimmter Teil der angeforderten Ressource gewählt werden, beispielsweise ein Anker in einer HTML Seite oder ein bestimmtes Kapitel in einem langen Dokument:

```
http://www.w3.org/TR/html4/struct/links.html#anchors
magazin://titanic/2015/01?typ=klartext
book://isdn_heute?auflage=3&sprache=deutsch#kapitel5
```

3.4.2 Verwendung innerhalb der PASTE-API

Bei der Verwendung der PASTE-API müssen der Anwendung *Schema* und *Pfad* der angeforderten Ressource bekannt sein. Der Anbieter kann abhängig vom verwendeten *Schema* Teil der übergebenen URI sein, aber für generische Namensräume auch entfallen. Ein Video-on-demand Anbieter wie *Netflix* oder *Amazon Prime* wird alle seine Filme unter dem eigenen Namensraum veröffentlichen, der auf die eigenen Server verweist:

```
video:// netflix.com /de/serien/Unbreakable_Kimmy_Schmidt
video:// amazon.de   /video/serien/The_Man_in_the_High_Castle
Schema   Anbieter    Pfad
```

Alternativ kann ein Name auch nur aus *Schema* und *Pfad* bestehen. In Fällen wo die gleiche Ressource oder der gleiche Anwendungsdienst auf mehreren Systemen und möglicherweise von mehreren Anbietern bereitgestellt wird, kann ein Namensraum mit eigenem *Schema* dafür definiert werden, da das PASTE Rahmenwerk die parallele Nutzung verschiedener Netzwerkarchitekturen unterstützt. Bei der Auflösung einer URI, werden alle bekannten Namensdienste aller lokal verfügbaren Netzwerkarchitekturen befragt:

```
wiki:// /Informatik/Telematik/Integrated_Services_Digital_Network
book:// /Pynchon/Thomas/Gravity\%27s_Rainbow.pdf
ccnx:// /8096170905/backup/dokumente/meine_passwoerter.txt
Schema  Pfad
```

Ist ein System beispielsweise per CCNx (also *Content Centric Networking*) ans Netzwerk angebunden, wird der entsprechende Namensdienst für das ccnx Schema eine Antwort liefern, während eine DNS Anfrage ergebnislos bleiben wird. Da Anwendungen, die Nutzern Ressourcen in Form von Inhalten oder Anwendungsdiensten bereitstellen, sich per BIND Primitive und einer URI an einen eindeutigen Namen binden, kann dieser per *longest prefix matching* aufgelöst werden. Dabei können sich Anwendungsdienstanbieter neben Schema- und Anbieterteil auch an Teile des Pfades oder einen kompletten Pfad binden. Anfragen an Unterpfade einer vom Anwendungsdienstanbieter gewählten URI, werden von PASTE an die per *longest prefix matching* lokalisierte Anwendungsinstanz weitergereicht. Dadurch können Anwendungsinstanzen eine Menge von Ressourcen oder Anwendungsdiensten unter einem einzelnen Namen verfügbar machen und die gewünschte Ressource beziehungsweise den gewünschten Anwendungsdienst auf der Anwendungsebene bestimmen.

Der Anfrage- und Fragmentteil ist im Gegensatz zum Schema, einem möglichen Anbieter und dem Pfad nicht festgelegt und nie Teil der an BIND übergebenen URI. Anfrage- und Fragmentteil werden entweder direkt vom Anwendungsentwickler festgelegt oder sind Schema-spezifisch. Beispielsweise Parameter für Auflösung, Seitenverhältnis und zu verwendendes Codec Format für ein generisches *video://* Namensschema. Anwendungen welche ihren Anwendungsdienst in einem solchen Namensraum zur Verfügung stellen, müssen in dem Fall die Namensraum spezifischen Parameter unterstützen. Des Weiteren können Anwendungen Anwendungsdienst-spezifische Parameter festlegen oder komplett auf Anfrage- und Fragmentteil der URI verzichten. Anfrage- und Fragmentteil werden von PASTE grundsätzlich an eine lokalisierte Anwendungsinstanz weitergereicht. Daher müssen Anfragen stellenden Anwendungsinstanzen in der Regel vorab die Anwendungsdienst- oder Schema-spezifischen Parameter bekannt sein.

Um auf der Anwendungsebene Informationen über Anwendungsdienst-spezifische Parameter zur Verfügung zu stellen, können Anwendungen diese per HEAD Primitive abfragbar machen. Damit können durch eine Anwendungsinstanz, welche einen Dienst nutzen will, für eine gegebene URI zunächst die verfügbaren Parameter erfragt werden; Sowie den Typ eines Parameters und eine Beschreibung desselben. Da der Anfrageteil der URI nur auf der Anwendungsebene relevant ist, dient er primär der Konkretisierung der Ausprägung von angeforderten Ressourcen beziehungsweise Diensten; Also zum genaueren Beschreiben des gewünschten Inhaltes oder des angeforderten Anwendungsdienstes. Datentransport-spezifische Parameter müssen hingegen Teil der Beschreibung des gewünschten beziehungsweise des angebotenen Transportdienstes sein und nicht Teil der URI, da nur diese Beschreibung unterhalb der Anwendungsschicht verarbeitet wird.

3.5 Transportdiensteigenschaften und -beschreibungen

Transportdienstbeschreibungen ermöglichen es einer Anwendungsinstanz bei Ressourcen- beziehungsweise Dienstanfragen die gewünschten Transportdiensteigenschaften anzufordern, ohne dass eine Anwendung sich dabei vorher auf ein bestimmtes Transportprotokoll festlegen muss. Damit erlaubt die PASTE-API unterhalb der Anwendungsschicht liegenden Netzwerkarchitekturen selbstständig ein geeignetes Protokoll anhand der Transportdienstbeschreibung zu wählen.

Es existiert bereits eine Vielzahl an unterschiedlichen Beschreibungssprachen im Web Kontext, beispielsweise die *Web-service Description Language* (WSDL) [W3C01] oder die *Web Ontology Language* (OWL) [W3C04a], welche auf dem *Resource Description Framework* (RDF) [W3C04b] basiert. Vielen ist gemein, dass sie sehr mächtig dadurch aber gleichzeitig auch sehr komplex und aufwendig zu benutzen sind. Im Rahmen dieser Arbeit wurde keine vollständige neue Beschreibungssprache entwickelt. Stattdessen werden im Folgenden Konzepte beschrieben, welche zum Beschreiben von Transportdiensten benötigte Elemente identifizieren. Insbesondere im Hinblick auf die Funktionalität, welche durch das PASTE Rahmenwerk bereitgestellt wird. Dazu wird eine einfache auf JSON basierende Notation verwendet, welche sich in existierende Beschreibungssprachen überführen lässt.

Transportdienste werden anhand von einer Menge von Transportdiensteigenschaften beschrieben. Es existieren drei Klassen, in die sich Transportdiensteigenschaften im Rahmen einer Transportdienstbeschreibung einordnen lassen:

- verpflichtende Transportdiensteigenschaften
- optionale Transportdiensteigenschaften
- informierende Transportdiensteigenschaften

Die verpflichtenden Transportdiensteigenschaften definieren dabei den von einer Anwendung benötigten Transportdienst und müssen auf den beteiligten Endsystemen übereinstimmen. Darüber kann beispielsweise ein zuverlässiger Transportdienst angefordert werden.

Optionale Anforderungen können zwischen den an einer Kommunikationsassoziation beteiligten Endsystemen ausgehandelt werden. Zusätzlich zu einem zuverlässigen Transportdienst, könnte ein Endsystem beispielsweise die Kompression der Daten wünschen, um die benötigte Datenübertragungsrate zu senken.

Informierende Eigenschaften werden nicht ausgehandelt, sondern dienen nur dazu zusätzliche aber nicht zwingend benötigte Informationen zu übermitteln. Das kann beispielsweise die Menge der zu übertragenden Daten sein, soweit diese bekannt ist. Aber auch ob eine Anwendung beispielsweise einen kontinuierlichen Datenstrom oder viele

kleine Datagramme versenden möchte. In Fällen wo mehrere gleichwertige Protokolle für einen Transportdienst zur Verfügung stehen, können diese zusätzlichen Informationen bei der Protokollwahl herangezogen werden oder auch an das Netzwerk- oder Ressourcenmanagement weitergereicht werden. Wird beispielsweise die zu übertragende Datenmenge bei einem Kommunikationsaufbau angegeben, kann, sofern kein zuverlässiger Dienst benötigt wird, ein verbindungsorientiertes Protokoll für große Datenmengen sinnvoller sein, während ein verbindungsloses für sehr kleine Datenmengen besser geeignet ist.

Einige Transportdiensteigenschaften welche beispielsweise von TCP angeboten werden sind:

- Reihenfolgetreue
- Duplikatfreiheit
- Vollständigkeit
- Fehlerfreiheit
- Fairness gegenüber anderen TCP-Datenströmen

Weitere Transportdiensteigenschaften lassen sich mit anderen Protokollen umsetzten (beispielsweise SCTP und RTP) oder sind heute häufig durch Mechanismen auf der Anwendungsschicht realisiert:

- Priorisierung von Datenströmen (beispielsweise durch SCTP)
- zeitlich begrenzte (partielle) Vollständigkeit (auch partielle Zuverlässigkeit) durch Sendewiederholungen nur bis zu einer maximalen Lebenszeit pro Paket
- tolerierbare fehlerhafte Daten (beispielsweise durch UDP-Lite)
- maximal verfügbare Datenübertragungsrate (durch automatische Datenkompression erreichbar)
- Jitter Kompensation (durch Puffern der Daten im Zielsystem)

3.5.1 Transportdiensteigenschaften

Sämtliche spezifizierten Transportdiensteigenschaften sind in Form eines Transportdiensteigenschaften Katalogs auf den Endgeräten verfügbar, welcher in Auflistung 2 dargestellt ist. Dieser enthält Einträge bestehend aus einem Namen, dem Typ und in der Regel einem Vergleichsoperator. Der Typ gibt an ob es sich beispielsweise um eine numerische oder boolesche Eigenschaft handelt. Der Vergleichsoperator definiert wie zwei konkrete Werte

dieser Eigenschaft miteinander verglichen werden können, wenn zwischen mehreren Teilnehmern ein zu verwendender Transportdienst ausgehandelt werden soll. Abhängig vom Typ existieren weitere Felder, welche im Folgenden erläutert werden.

Auflistung 2: *Aufbau des Transportdiensteigenschaften Katalogs.*

```
{
  "features" : [
    {
      "name"       : "(feature name)",
      "type"       : "(feature type)",
      "comparator" : "(comperator type)"
    },
                {
      "name"       : "(feature name)",
      "type"       : "(feature type)",
      "comparator" : "(comperator type)"
    },
                ...
  ]
}
```

Der Eigenschaftentyp kann dabei folgende Werte annehmen:

- **boolean** - Ein Transportdienst kann diese Transportdiensteigenschaft entweder erfüllen oder weist diese Transportdiensteigenschaft nicht auf. Beispielsweise *Flusskontrolle*.
- **time** - Für zeitabhängige Transportdiensteigenschaft, wie beispielsweise *Jitter* oder *Latenz*.
- **size** - Für größenabhängige Transportdiensteigenschaft, wie beispielsweise die maximal erlaubte *Nachrichtengröße* in *kByte*.
- **size-per-time** - Für ratenabhängige Transportdiensteigenschaft, wie beispielsweise die maximal erlaubte *Senderate* in Form von *Mbit/s*.
- **count** - Für größenabhängige Transportdiensteigenschaft, wie beispielsweise die maximale Anzahl an *Sendewiederholungen*. Im Gegensatz zum **size** Typ eignet sich dieser Typ für beliebige generische Größen.

- **count-per-time** - Für ratenabhängige Transportdiensteigenschaft, wie beispielsweise die maximal erlaubte *Token Rate* eines *Traffic Shapers*. Im Gegensatz zum **size-per-time** Typ eignet sich dieser Typ für beliebige generische zeitabhängige Größen.
- **set** - Definiert eine Menge von Werten, von denen die Transportdiensteigenschaft **einen** oder **mehrere** annehmen kann.
- **list** - Definiert eine Liste von Werten, von denen die Transportdiensteigenschaft **genau einen** annehmen kann.
- **container** - Definiert eine Gruppe von Transportdiensteigenschaften des gleichen Typs als eigenständige Transportdiensteigenschaft. Beispielsweise *Zuverlässigkeit*, welche sich aus den Transportdiensteigenschaften *Reihenfolgetreue*, *Duplikatfreiheit*, *Vollständigkeit* und *Fehlerfreiheit*, alle vom Typ *boolean*, zusammensetzt.

Im Falle der Typen *set*, *list* und *container* weist die Transportdiensteigenschaftendefinition im Transportdiensteigenschaftenkatalog ein weiteres Feld mit dem Namen *content* auf, welches eine Menge der definierten Werte enthält. Dies ist in Auflistung 3 für die Typen *set* und *list* beziehungsweise in Auflistung 4 für den Typ *container* aufgeführt. Im Falle des Typs *container* enthält das *content* Feld die Menge der Transportdiensteigenschaften, aus denen sich die spezifizierte Transportdiensteigenschaft zusammensetzt.

Auflistung 3: *Transportdiensteigenschaft vom Typ* `set` *bzw.* `list`.

```
{
  "name"    : "(feature name)",
  "type"    : "set|list",
  "content" : [
    "item a",
    "item b",
    ...
    "item z"
  ]
}
```

Auflistung 5 zeigt einen Auszug des Transportdiensteigenschaften Katalogs mit einigen der oben genannten Transportdiensteigenschaften.

Bei sämtlichen Transportdiensteigenschaften handelt es sich hier um boolesche Typen. Als Vergleichsoperator ist im Katalogauszug aus Auflistung 5 ein *logisches Und* angegeben. Bei einem Wertekonflikt, wenn eine Eigenschaft beispielsweise nur von einer Seite

Auflistung 4: *Transportdiensteigenschaft vom Typ* `container`.

```
{
  "name"    : "(feature group name)",
  "type"    : "container",
  "content" : [
    "feature name a",
    "feature name b",
    ...
    "feature name z"
  ]
}
```

einer Kommunikationsassoziation unterstützt wird, muss also ein Transportdienst ohne diese Eigenschaft gewählt werden. Nur wenn alle an einer Kommunikationsassoziation beteiligten Systeme eine Eigenschaft unterstützen, ist sie nach Anwendung des *logisches Und* Operators verfügbar.

Boolesche Transportdiensteigenschaften können in einem Container zusammengefasst werden, um häufig vorkommende Transportdiensteigenschaftenkombinationen zu einer Transportdiensteigenschaft zusammenzufassen. So kann beispielsweise *Zuverlässigkeit* über das Vorhandensein der in Auflistung 6 aufgezählten Eigenschaften definiert werden.

Wird in einer Transportdienstbeschreibung *Zuverlässigkeit* gefordert, wird diese Eigenschaft bei der Transportdienstaushandlung einfach durch die im Container angegebenen ersetzt, welche alle auf den gleichen Wert (*true* oder *false*) wie die Container Eigenschaft gesetzt werden. Auf das Format der Transportdienstbeschreibungen wird in Abschnitt 3.5.2 eingegangen. Einige Beispiele für zeitabhängige Eigenschaften finden sich in Auflistung 7.

Mit Hilfe der *qos://transport/data/lifetime* Eigenschaft lässt sich partielle Vollständigkeit beschreiben. Also die vollständige Zustellung einzelner Pakete, solange eine maximale Lebensdauer der Pakete nicht überschritten wird, in welchem Fall sie verworfen werden können.

Die maximal tolerierbare Ende-zu-Ende Latenz *qos://transport/delay/end-to-end* einer Kommunikationsassoziation ist eine weitere numerische Eigenschaft vom Typ *time*. Beide Eigenschaften haben als Vergleichsoperator das Minimum aus miteinander in Konflikt stehenden Werten definiert.

Die Eigenschaft *qos://transport/delay/tolerance* ist ähnlich definiert wie die Eigenschaft *qos://transport/data/lifetime*. Statt aber die maximale Lebensdauer eines Paketes

Auflistung 5: *Auszug des Transportdiensteigenschaften Katalogs.*

```
{
  "features" : [
    {
      "name"       : "qos://transport/reordering",
      "type"       : "boolean",
      "comparator" : "&"
    },
    {
      "name"       : "qos://transport/noduplicates",
      "type"       : "boolean",
      "comparator" : "&"
    },
    {
      "name"       : "qos://transport/completeness",
      "type"       : "boolean",
      "comparator" : "&"
    },
    {
      "name"       : "qos://transport/error/correction",
      "type"       : "boolean",
      "comparator" : "&"
    },
    {
      "name"       : "qos://transport/control/flow",
      "type"       : "boolean",
      "comparator" : "&"
    }
  ]
}
```

anzugeben, wird mit ihr die gewünschte minimale Lebensdauer einer Kommunikationsassoziation festgelegt. Als Vergleichsoperator ist das Maximum definiert; Es wird also die größte von einer Anwendungsinstanz gewünschte Mindestlebensdauer gewählt, was für langlebige Kommunikationsassoziation beispielsweise bei Delay Tolerant Networking Anwendungen sinnvoll sein kann. Hiermit wird festgelegt, wie lange versucht werden soll anfallende Daten auszuliefern, bis eine Kommunikationsassoziation beendet wird.

Eigenschaften vom Typ *size-per-time* können die gewünschte Datenrate spezifizieren, wie beispielsweise eine garantierte Mindestbitrate. Genauso kann eine Anwendung die

Auflistung 6: *Beispiel `container` für die Transportdiensteigenschaft Zuverlässigkeit.*

```
{
  "name"    : "qos://transport/reliability",
  "type"    : "container",
  "content" : [
    "qos://transport/reordering",
    "qos://transport/noduplicates",
    "qos://transport/completeness",
    "qos://transport/error/correction"
  ]
}
```

Auflistung 7: *Beispiele für zeitabhängige Transportdiensteigenschaften.*

```
{
  "name"       : "qos://transport/data/lifetime",
  "type"       : "time",
  "comperator" : "minimum"
},
{
  "name"       : "qos://transport/delay/end-to-end",
  "type"       : "time",
  "comperator" : "minimum"
},
{
  "name"       : "qos://transport/delay/tolerance",
  "type"       : "time",
  "comperator" : "maximum"
}
```

maximale Datenrate, mit der sie Daten empfangen möchte, begrenzen. Auflistung 8 zeigt beispielhaft die Eigenschaft *qos://transport/bitrate/guaranteed* und die Eigenschaft *qos://transport/bitrate/maximum*.

Ein Beispiel für eine Eigenschaft vom Typ *size* ist in Auflistung 9 zu sehen. Mit *qos://transport/data/expected-size* kann beispielsweise die eingangs erwähnte Datenmenge, die eine Anwendungsinstanz versenden möchte, angegeben werden.

Mit Hilfe des Typ *set* können Eigenschaften spezifiziert werden, bei denen ein Wert für eine Eigenschaft aus einer Liste von vorgegebenen Parametern gewählt werden kann.

Auflistung 8: *Beispiele für Datenraten abhängige Transportdiensteigenschaften.*

```
{
  "name"       : "qos://transport/bitrate/guaranteed",
  "type"       : "size-per-time",
  "comperator" : "maximum"
},
{
  "name"       : "qos://transport/bitrate/maximum",
  "type"       : "size-per-time",
  "comperator" : "minimum"
}
```

Auflistung 9: *Beispiele für Größen abhängige Transportdiensteigenschaften.*

```
{
  "name"       : "qos://transport/data/expected-size",
  "type"       : "size",
  "comperator" : "maximum"
}
```

Beispielsweise kann zwischen einem Raten- oder Fenster-basierten Staukontrollverfahren gewählt werden, wie in Auflistung 10 dargestellt. Als Vergleichsoperator wird die Schnittmenge aller Transportdienstbeschreibungen gebildet. Wird mehr als ein Parameter angegeben und unterstützt, so wird die Reihenfolge der Parameter als Präferenz interpretiert und der erste verfügbare Parameter nach der Schnittmengenbildung gewählt.

Auflistung 10: *Beispiele für eine Transportdiensteigenschaft vom Typ set.*

```
{
  "name"       : "qos://transport/control/congestion/class",
  "type"       : "set",
  "comparator" : "intersection",
  "set"        : ["rate-based", "window-based", "mixed"]
}
```

Weitere Beispiele sind die Wahl einer *Traffic*-Klasse oder die Angabe des Medientyps der zu übertragenden Daten, wie in Auflistung 11 gezeigt. Gleichzeitig sind dies Beispiele für informierende Eigenschaften, weshalb kein Vergleichsoperator definiert sein muss. Diese werden nicht ausgehandelt sondern lediglich als Teil der Transportdienstbeschreibung ans PASTE Rahmenwerk gereicht. Die unterliegende Netzwerkarchitektur kann diese Eigenschaften vollständig ignorieren. Optional können aber auch Netz-spezifische Optimierungen anhand dieser Zusatzinformationen getroffen werden, sofern die unterliegende Netzwerkarchitektur solche unterstützt.

Auflistung 11: *Transportdiensteigenschaften vom Typ set und list.*

```
{
  "name" : "qos://transport/umts-like/class",
  "type" : "set",
  "list" : ["conversational", "streaming",
            "interactive", "background"]
}

{
  "name" : "qos://media/type",
  "type" : "list",
  "list" : [
    "application-specific",
    "text",
    "audio",
    "video",
    "image",
    "message",
    ...
  ]
}
```

3.5.2 Transportdienstbeschreibung

Transportdienstbeschreibungen bestehen aus einer Liste von Transportdiensteigenschaften, welche über ihren Namen im Transportdiensteigenschaftenkatalog referenziert werden. Den Transportdiensteigenschaften werden in der Transportdienstbeschreibung konkrete Werte zugewiesen. Aufgeteilt sind die Transportdiensteigenschaften in jeweils eine Gruppe von verpflichtenden, optionalen und informierenden Transportdiensteigenschaf-

ten. Dabei kann grundsätzlich jede Transportdiensteigenschaft in jeder Gruppe verwendet werden, je nachdem ob eine Anwendung sie zwingend benötigt oder auch optional darauf verzichten kann. Die einzige Ausnahme bilden Transportdiensteigenschaften, für die kein Vergleichsoperator definiert ist. Diese können nur im informierenden Teil der Transportdienstbeschreibung stehen, da keine Möglichkeit der Aushandlung besteht.

Dies ist auch der wesentliche Unterschied zwischen den verpflichtenden und optionalen Transportdiensteigenschaften: Für verpflichtende Transportdiensteigenschaften wird später lediglich die Erfüllbarkeit überprüft. Kann eine oder mehrere der verpflichtenden Transportdiensteigenschaften nicht erfüllt werden, lehnt die Gegenseite eine Kommunikationsassoziationsanfrage unter Angabe dieser Transportdiensteigenschaften ab. Ist eine Transportdiensteigenschaft hingegen als optional klassifiziert, wird diese von der Gegenseite mit Hilfe des Vergleichsoperators angepasst und an die anfragende Anwendungsinstanz übermittelt, welche dann entscheiden muss ob die angepassten Werte akzeptabel sind. Auflistung 12 zeigt den Aufbau einer Transportdienstbeschreibung.

Beim Anbieten eines Anwendungsdienstes mittels BIND legt die Anwendung einen oder mehrere unterstützte Transportdienste fest und gibt zusätzlich an über welche Dienstprimitive diese jeweils verfügbar sind. Anfragen über GET, PUT und CONNECT wählen ebenfalls einen zu verwendenden Transportdienst über die Angabe einer Transportdienstbeschreibung aus. Mit Hilfe der für jede Eigenschaft definierten Vergleichsoperatoren kann ein Satz an gemeinsamen Transportdiensteigenschaften bestimmt werden, welcher die zu verwendende Transportdienstbeschreibung bildet. Der eigentliche Aushandlungsprozess ist in Kapitel 4 genauer beschrieben.

Die in die drei Klassen unterteilten Transportdiensteigenschaftenanforderungen bestehen jeweils aus dem Namen der referenzierten Transportdiensteigenschaft aus dem Transportdiensteigenschaftenkatalog *name*, dem gewünschten Wert der Transportdiensteigenschaft *value* und der Einheit dieses Wertes *unit*. Die Felder *serviceID* und *templateID* werden bei einer vollständigen Transportdienstbeschreibung zunächst nicht verwendet. Ihre Verwendung wird in Abschnitt 3.5.3 erläutert.

Eine Transportdienstbeschreibung für einen partiell beziehungsweise zeitlich beschränkten zuverlässigen Dienst, wie er von einem Mehrspieler-Online-Spiel genutzt werden könnte, sieht beispielsweise wie in Auflistung 13 gezeigt aus.

Die Schlüssel für *serviceID* und *templateID* sind erneut nur der Vollständigkeit halber aufgeführt aber in diesem Beispiel ungenutzt. Darauf folgen drei Listen mit zu verwendenden Transportdiensteigenschaften. Dabei werden definierte Transportdiensteigenschaften aus dem Transportdiensteigenschaftenkatalog über ihren Namen referenziert. Gefolgt vom gewünschten Wert und der Einheit dieses Wertes. Erlaubte Einheiten hängen vom Typ der definierten Transportdiensteigenschaft ab. Ebenfalls spielt die Reihenfolge eine Rolle. So wird in diesem Beispiel zunächst überprüft, ob eine Lebensdauer von 30 Millisekunden eingehalten werden kann, ansonsten dürfen Daten verworfen werden. Erst

Auflistung 12: *Aufbau einer Transportdienstbeschreibung.*

```
{
  "description" : {
    "serviceID"  : "nul",
    "templateID" : "nul",
    "mandatory" : [
                        {
        "name"  : "(refrenced feature name)",
        "value" : "(feature value)",
        "unit"  : "(feature unit)"
      }
    ],
    "optional" : [
      {
        "name"  : "(refrenced feature name)",
        "value" : "(feature value)",
        "unit"  : "(feature unit)"
      }
    ],
    "informational" : [
      {
        "name"  : "(refrenced feature name)",
        "value" : "(feature value)",
        "unit"  : "(feature unit)"
      }
    ]
  }
}
```

danach wird gefordert, dass sämtliche Daten innerhalb der Lebensdauergrenze zuverlässig übertragen werden, im Unterschied zu einer zuverlässigen Übertragung, bei der alle Daten übertragen werden müssen. Ein solcher Dienst wird unter anderem von SCTP angeboten [Ste+04] und vermeidet das *Head-of-line blocking* Problem von TCP sowie das unnötige erneute Übertragen von Dateneinheiten, welche den Empfänger zu spät erreichen würden.

Im Falle von Online-Spielen ist häufig die Aktualisierungsrate und die Größe der ausgetauschten Zustandsinformationen bekannt. Bei 60 Aktualisierungen pro Sekunde und einer Größe von 128 Byte wird eine Datenrate von 7,5 kByte/s benötigt. Diese ist im Beispiel als `qos://transport/bitrate/maximum` angegeben. Da es für die Anwendung

Auflistung 13: *Eine Transportdienstbeschreibung für einen partiell zuverlässigen Dienst.*

```
{
  "description" : {
    "serviceID"  : "nul",
    "templateID" : "nul",
    "mandatory" : [
      {
        "name"  : "qos://transport/data/lifetime",
        "value" : "30",
        "unit"  : "ms"
      },
      {
        "name"  : "qos://transport/reliability",
        "value" : "true",
        "unit"  : "boolean"
      }
    ],
    "optional" : [
      {
        "name"  : "qos://transport/bitrate/maximum",
        "value" : "7.5",
        "unit"  : "kbyte/s"
      }
    ],
    "informational" : [
      {
        "name"  : "qos://transport/umts-like/class",
        "value" : ["interactive"],
        "unit"  : "nul"
      }
    ]
  }
}
```

unerheblich ist, ob die verfügbare Datenrate größer als 7,5 kByte/s ist, wird es unter den optionalen Transportdiensteigenschaften aufgeführt. Damit kann die Gegenseite den Wert zur besseren Ressourcenverwaltung heranziehen, einfach ignorieren oder aber im Rahmen der Aushandlung nach unten korrigieren, falls keine 7,5 kByte/s zur Verfügung stehen.

Zuletzt ist noch die Verkehrsklasse der zu übertragenden Daten angegeben, welche ebenfalls zusätzlich bei der Wahl oder Parametrisierung eines Transportprotokolls verwendet werden kann aber nicht muss.

3.5.3 Hierarchische Transportdienstwahl & -beschreibung

Das Festlegen von Transportdiensten welche von einer Anwendung genutzt werden sollen muss vom Anwendungsentwickler vorgenommen werden. Da dazu mindestens eine Transportdienstbeschreibung angelegt werden muss, kann das einen erheblichen Mehraufwand gegenüber heutigen Anwendungsschnittstellen bedeuten, wo der Entwickler häufig nur die Wahl zwischen einem von einer Hand voll Standardprotokolle hat. Dafür benötigt der Anwendungsentwickler im Gegenzug kein Wissen über die Funktionalität konkreter Protokolle mehr. Stattdessen reicht die Kenntnis generischer vom Netz angebotener Dienste. Durch diese Entkopplung können Protokolle leicht ausgetauscht oder neue Protokolle schneller ausgebracht werden, ohne dass sich für den Entwickler etwas ändert. Diesem stehen langfristig lediglich vielfältigere Dienste zur Verfügung, während Anwendungen von neuen Protokollen profitieren können, ohne angepasst werden zu müssen.

Um den Aufwand der mit der Erstellung einer Transportdienstbeschreibung einhergeht zu reduzieren, wurde zudem eine drei stufige Hierarchie von Transportdienstbeschreibungen unterschiedlicher Granularität eingeführt. Dabei ist die grundsätzliche Idee, dass häufig verwendete Transportdienste über verkürzte Transportdienstbeschreibungen angefordert werden können.

Abbildung 3.6 zeigt die unterschiedlichen Hierarchiestufen. Dabei handelt es sich auf der linken Seite um auf den Endgeräten verfügbare Informationen. Diese sind für alle Endgeräte identisch und können von einer globalen Quelle, wie einem Dienstverzeichnis von Endgeräten bezogen und dann lokal abgelegt werden. Auf der ersten Stufe ist dies ein Katalog mit sämtlichen standardisierten Transportdiensteigenschaften. Auf der rechten Seite sind die konkreten parametrisierten Beschreibungen abgebildet, welche jeweils über das Netzwerk ausgehandelt werden müssen. Auf der ersten Stufe stehen hier die vollständigen Transportdienstbeschreibungen, welche Transportdiensteigenschaften aus dem lokal verfügbaren Transportdiensteigenschaften Katalog referenzieren. Diese Stufe entspricht dem vorangegangenen Beispiel. Dabei handelt es sich um den allgemeinsten Fall einer Transportdienstbeschreibung, bei dem der gewünschte Transportdienst durch eine beliebige Kombination von Transportdiensteigenschaften angegeben werden kann. Häufig verwendete Kombinationen von Transportdiensteigenschaften können alternativ lokal in Form von Dienstschablonen abgelegt werden. Wird eine bestimmte Art von Dienst also häufig verwendet, beispielsweise der heutige zuverlässige *Best-Effort* Transport wie ihn TCP anbietet, dann macht es Sinn diesen bereits vorgefertigt anzubieten.

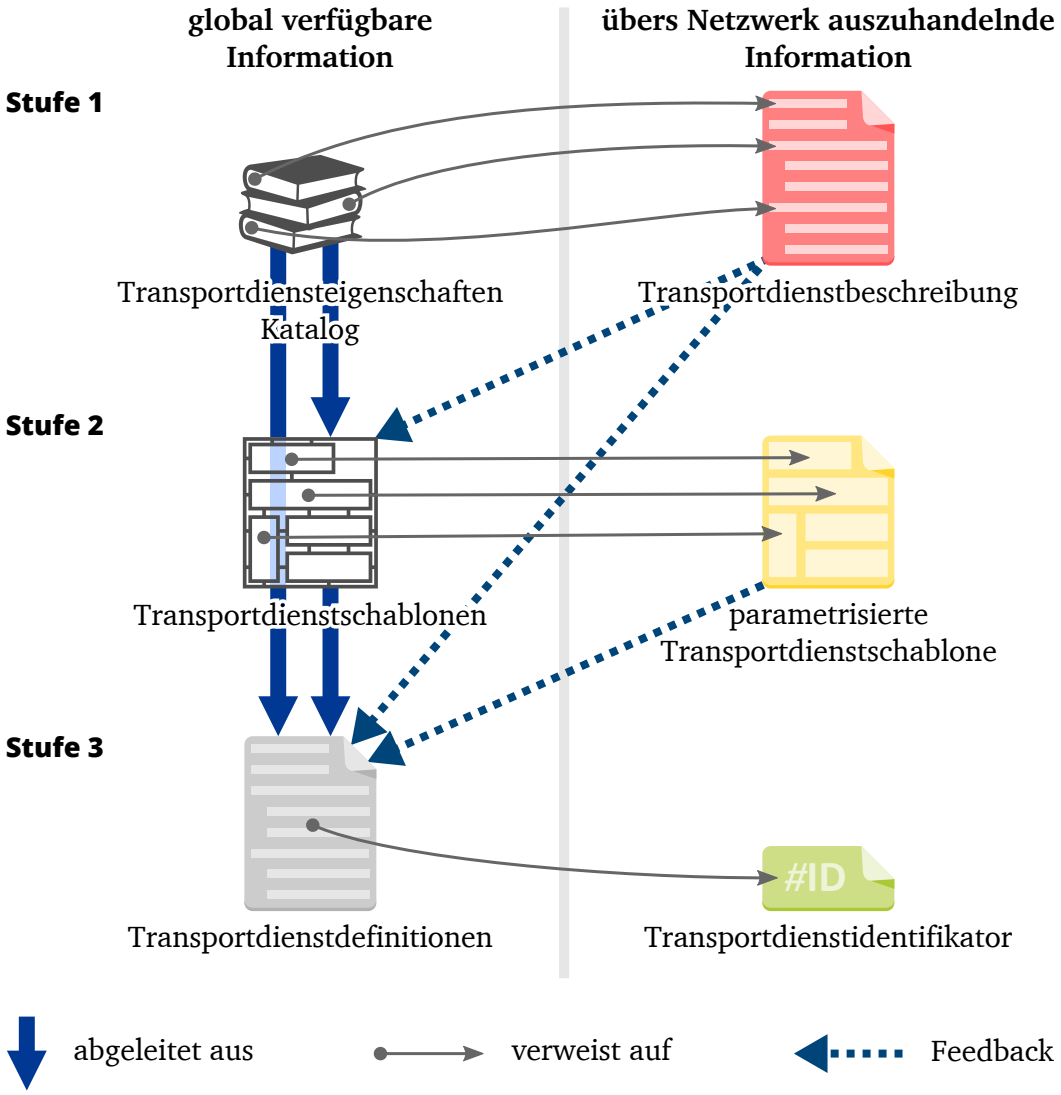


Abbildung 3.6: *Transportdiensteigenschaften und -beschreibungen unterschiedlicher Granularität auf verschiedenen Stufen.*

Diese verkürzten Beschreibungen existieren in zwei Ausprägungen. Eine Transportdienstschablone, welche als Grundlage der Transportdienstbeschreibung verwendet werden kann, und Transportdienstidentifikatoren, über die vollständige Transportdienstbeschreibungen referenziert werden können. Eine Transportdienstschablone ist im Allgemeinen wie in Auflistung 14 abgebildet definiert.

Die Transportdienstschablone legt die Klassen, Menge und Reihenfolge der enthaltenen Transportdiensteigenschaften fest. Dadurch können diese bei der Transportdienstbeschreibung weggelassen werden, wie in Auflistung 15 zu sehen ist, werden nur noch die Werte

Auflistung 14: *Aufbau einer Transportdienstschablone.*

```
{
  "template" : {
    "ID" : "(template name)",
    "mandatory" : [
      "(refrenced feature name)",
      "(refrenced feature name)"
    ],
    "optional" : [
      "(refrenced feature name)",
      "(refrenced feature name)"
    ],
    "informational" : [
      "(refrenced feature name)",
      "(refrenced feature name)"
    ]
  }
}
```

der einzelnen Transportdiensteigenschaften aufgelistet. Die zum vorherigen Beispiel gehörige Schablone sieht wie in Auflistung 16 dargestellt aus.

Schablonen definieren also die Struktur gängiger Transportdienstbeschreibungen und bilden die zweite Stufe der lokal auf jedem System verfügbaren Information. Neben einem Namen für die Schablone, enthält diese die verwendeten Transportdiensteigenschaften in einer definierten Reihenfolge und aufgeteilt in die entsprechenden Klassen; also verpflichtende, optionale und informierende Transportdiensteigenschaften. Schablonen können in einer Transportdienstbeschreibung über den *templateID* Schlüssel referenziert werden, so dass nur noch die fehlenden Parameter angegeben werden müssen. Eine passende Transportdienstbeschreibung zur vorherigen Schablone enthält lediglich die Werte, der in der Schablone vorgegebenen Transportdiensteigenschaften und ist in Auflistung 17 zu sehen.

Anwendungsentwicklern wird damit die Möglichkeit gegeben, für eine bestimmte Klasse von Anwendungen eine passende Schablone zu wählen und nur noch die konkreten Werte auf ihre Bedürfnisse anzupassen.

Auf der untersten und dritten Stufe in Abbildung 3.6 stehen vollständige Transportdienstdefinitionen. Analog zur zweiten Stufe, können gängige Parametrisierungen häufig verwendeter Transportdienstschablonen als vollständige Transportdienstdefinition unter einem eindeutigem Transportdienstidentifikator abgelegt werden. Transportdienstidenti-

Auflistung 15: *Eine Transportdienstbeschreibung mit Transportdienstschablone.*

```
{
  "description" : {
    "templateID" : "(template name)",
    "values" : [
      {
        "value" : "(value)",
        "unit"  : "(unit)"
      },
      {
        "value" : "(value)",
        "unit"  : "(unit)"
      },
                        {
        "value" : "(value)",
        "unit"  : "(unit)"
      },
                        {
        "value" : "(value)",
        "unit"  : "(unit)"
      },
    ]
  }
}
```

fikatoren enthalten im allgemeinen nur noch den eindeutigen Namen des gewünschten Transportdienstes und sehen wie in Auflistung 18 aus.

Dieser wird in Form eines *ID* Schlüssels festgelegt. Statt aus einer parametrisierten Schablone kann eine Transportdienstdefinition auch aus einer vollständigen Transportdienstbeschreibung, wie sie auf der ersten Stufe verwendet wird, bestehen. Eine beispielhafte Transportdienstdefinition für den Mehrspielerdienst sieht demnach ähnlich aus, wie eine parametrisierte Transportdienstschablone und ist in Auflistung 19 abgebildet.

Spezifische Transportdienste können dann vom Anwendungsentwickler anhand des Transportdienstidentifikators direkt ausgewählt werden. Auf jeder Stufe sinkt somit die Menge der Daten, welche zum Aushandeln des zu verwendenden Transportdienstes übertragen werden müssen, was ebenfalls den Mehraufwand beim Kommunikationsassoziationsaufbau reduziert. Wird ein Transportdienstidentifikator verwendet, reduziert sich

Auflistung 16: *Die Transportdienstschablone für partielle Zuverlässigkeit.*

```
{
  "template" : {
    "ID" : "gaming",
    "mandatory" : [
      "qos://transport/data/lifetime",
      "qos://transport/reliability"
    ],
    "optional" : [
      "qos://transport/bitrate/maximum"
    ],
    "informational" : [
      "qos://transport/umts-like/class"
    ]
  }
}
```

die zu übertragende Transportdienstbeschreibung zu einer Referenz auf den gewünschten Dienst wie in Auflistung 20.

In diesem Fall müssen gar keine Transportdiensteigenschaften mehr ausgehandelt werden, was zu einer Verkürzung des Verbindungsaufbaus führt, wie in Abschnitt 4.5.4 später gezeigt wird. Während Transportdienstschablonen also primär eine Erleichterung für den Anwendungsentwickler darstellen, können Transportdienstidentifikator auch die Zeit beim Kommunikationsassoziationsaufbau deutlich verkürzen. Daher sind sie besonders sinnvoll, wenn auf eine flexible anwendungsangepasste Dienstbeschreibung zugunsten einer schnelleren Antwortzeit verzichtet werden kann.

3.6 Anwendungsfälle

Im Folgenden wird anhand von vier Anwendungsfällen der Einsatz der PASTE-API aus der Sicht der beteiligten Anwendungsinstanzen dargestellt:

- Ein Chat-Server mit zwei Chat-Clients
- Ein Online Daten Backupdienst
- Ein Video-On-Demand Streamingdienst
- Ein Publish-Subscribe Dienst

Auflistung 17: *Die Transportdienstbeschreibung für die partielle Zuverlässigkeit Transportdienstschablone.*

```
{
  "description" : {
    "templateID" : "gaming",
    "values" : [
                  {
       "value" : "30",
       "unit"  : "ms"
      },
      {
        "value" : "true",
        "unit"  : "nul"
      },
      {
        "value" : "7.5",
        "unit"  : "kbyte/s"
      },
      {
        "value" : ["interactive"],
        "unit"  : "nul"
      }
    ]
  }
}
```

Auflistung 18: *Aufbau eines Transportdienstidentifikators.*

```
{
  "description" : {
    "serviceID" : "(service name)"
  }
}
```

Bei den Anwendungsfällen handelt es sich um gängige heute existierende Anwendungen und Dienste. Diese wurden so ausgewählt, dass der Einsatz der grundlegenden Dienstprimitive der PASTE-API gezeigt wird. Insbesondere die Umsetzung verschiedener Kommunikationsmuster durch den Einsatz von `GET`, `PUT` und `CONNECT`.

Auflistung 19: *Beispiel für eine Transportdienstdefinition.*

```
{
  "service" : {
    "ID" : "realtime_gaming",
    "templateID" : "gaming",
    "values" : [
      {
        "value" : "30",
        "unit"  : "ms"
      },
                        {
        "value" : "true",
        "unit"  : "nul"
      },
      {
        "value" : "7.5",
        "unit"  : "kbyte/s"
      },
      {
        "value" : ["interactive"],
        "unit"  : "nul"
      }
    ]
  }
}
```

Auflistung 20: *Eine Transportdienstbeschreibung über einen Transportdienstidentifikator.*

```
{
  "description" : {
    "serviceID" : "realtime_gaming"
  }
}
```

3.6.1 Chat-Server mit zwei Chat-Clients

Im ersten Anwendungsfall tauschen zwei Chat-Clients über bidirektionale Verbindungen mit einem Chat-Server Nachrichten untereinander aus. Abbildung 3.7 zeigt drei Anwendungsinstanzen. Einen Chat-Server S und zwei Chat-Clients A und B. Zunächst wird

vom Server S mit Hilfe von `BIND` ein Endpunkt in Form eines *HANDLES* erzeugt. Dazu registriert S die URI `instant://msg.me/room` für die CONNECT Dienstprimitive und einen zuverlässigen Datagramm Dienst. Die Unterstützung verschiedener Chat Räume wird in diesem Fall auf der Anwendungsebene anhand des letzten Pfadteils der URI umgesetzt. Ein Chat-Client bestimmt den gewünschten Chat Raum also, indem er den Raumnamen an die vom Server S registrierte URI beim Assoziationsaufbau anhängt. Der Aufbau des Names ist hier also spezifisch für die konkrete Anwendung. Sofern Client und Server vom selben Entwickler stammen, wird im Allgemeinen angenommen, dass der Aufbau allen Anwendungsinstanzen bekannt ist.

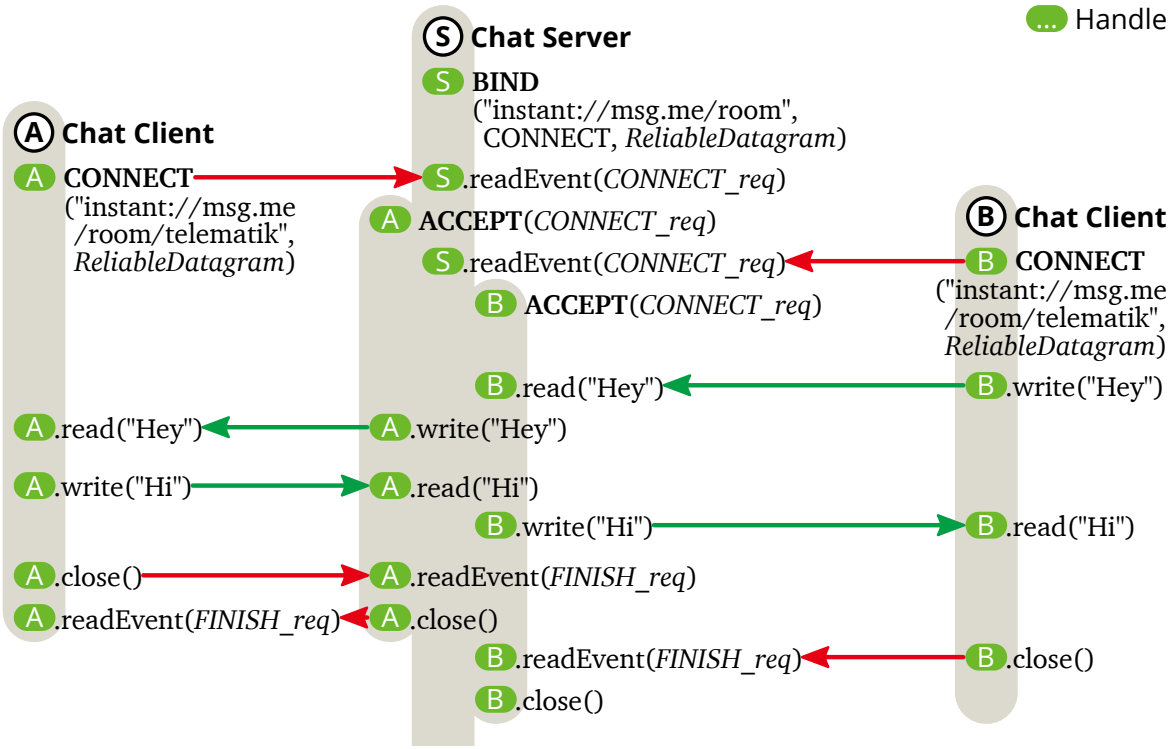


Abbildung 3.7: *Chat-Server S mit zwei Chat-Clients A & B.*

Der Übersichtlichkeit halber ist in Abbildung 3.7 die verwendete URI, Dienstprimitive und ein Dienstname angegeben, welcher auf einen entsprechenden Dienstidentifikator abgebildet werden kann. Der vollständige Aufruf könnte stattdessen auch eine oder mehrere vollständige Dienstbeschreibungen verwenden:

```
BIND("instant://msg.me/room", CONNECT, ReliableDatagram)
```

Die CONNECT Dienstprimitive wird verwendet, da Nutzer des Anwendungsdienstes Nachrichten sowohl verschicken als auch empfangen können. Ein zuverlässiger Trans-

portdienst garantiert sämtlichen beteiligten Anwendungsinstanzen dabei, dass keine Nachrichten verloren gehen.

Anschließend erzeugt Chat Client A einen Endpunkt durch folgenden Aufruf:

```
CONNECT("instant://msg.me/romm/telematik", ReliableDatagram)}
```

Der Server S wird über die Verbindungsanfrage über den Ereigniskanal per `CONNECT_req` Ereignis informiert und kann diese per `ACCEPT` annehmen. Dadurch wird eine Kommunikationsassoziation zwischen Anwendungsinstanz A und S aufgebaut. Analog dazu baut Chat Client B anschließend ebenfalls eine Kommunikationsassoziation zu S auf. Nun können A und B über S Nachrichten austauschen indem sie `read()` und `write()` auf dem entsprechenden HANDLE aufrufen. Anhand der beim `CONNECT` verwendeten URI kann S alle zum Telematik Chatraum gehörenden Endpunkte identifizieren. Bei Erhalt einer Nachricht wird diese von S an alle Endpunkte außer dem von dem die Nachricht empfangen wurde weitergeleitet. In Abbildung 3.7 sendet zunächst B eine Nachricht an A und anschließend antwortet A mit einer Nachricht an B. Danach schließen beide die Kommunikationsassoziation durch den Aufruf von `close()`. Der Server S wird darüber jeweils per `FINISH_req` Ereignis informiert und kann die Kommunikationsassoziationen ebenfalls schließen. Auch darüber werden die Chat Clients wie in Abbildung 3.7 dargestellt per `FINISH_req` Ereignis informiert. Dies kann aber auch wie im Falle von Chat Client B von der Anwendung ignoriert werden, falls diese beispielsweise weder weitere Daten senden noch empfangen möchte und ihre Kommunikationsassoziation bereits abgebaut hat.

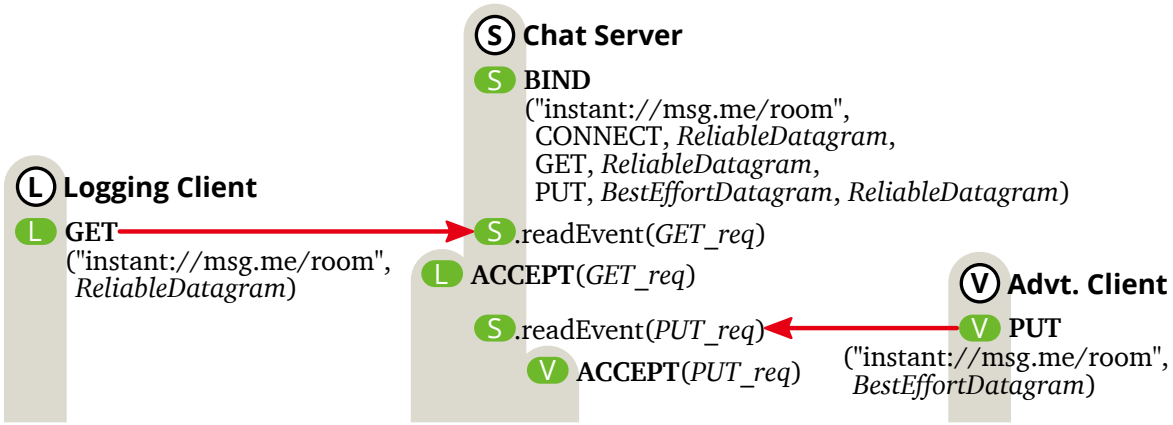


Abbildung 3.8: *Chat-Server S mit Logging Client L & Advertisement Client V.*

Zusätzlich zu `CONNECT` kann der Chat Server S auch die Dienstprimitive `GET` und `PUT` unterstützen wie in Abbildung 3.8 gezeigt:

```
BIND("instant://msg.me/room", CONNECT, ReliableDatagram,
     GET, ReliableDatagram,
     PUT, ReliableDatagram, BestEffortDatagram)
```

Da Anwendungsinstanzen neben der URI auch die Dienstprimitive die zum Aufbauen einer Kommunikationsassoziation verwendet wird bekannt ist, kann der Chat Server S diese nutzen um verschiedene Arten von Chat-Clients zu unterstützen und zu unterscheiden. Wie in Abbildung 3.8 zu sehen, kann ein Logging Client L sich per `GET` Primitive für sämtliche Chat Räume registrieren und ein Nachrichtenlog erstellen. Da dieser Client niemals Nachrichten verschicken wird, braucht er in diesem Fall nur eine unidirektionale Kommunikationsassoziation auf der Anwendungsebene. Der Server S braucht zudem nur lesend auf einen Chat Raum zugreifende Clients nicht in einer Teilnehmerliste aufführen, da diese sich nie aktiv am Chat beteiligen.

Ein *Advertisement* Client V hingegen kann die `PUT` Dienstprimitive verwenden, um dem Server S zu signalisieren, dass er für keinen Chat Raum Nachrichten empfangen möchte sondern nur Nachrichten sendet. Wird beispielsweise in regelmäßigen Abständen Werbung in die Chat Räume geschickt, ist auch ein zuverlässiger Dienst nicht zwingend nötig. So dass für die `PUT` Primitive von Server S auch ein *BestEffortDatagram* Dienst angeboten wird, den Clients alternativ nutzen können.

Über die verwendeten Dienstprimitive können Anwendungen also einfach verschiedene Kommunikationsmuster umsetzen, da S eingehende Nachrichten beispielsweise nur an mit `GET` und `CONNECT` verbundene Chat-Clients weiterleiten muss.

3.6.2 Online Daten Backupdienst

Abbildung 3.9 zeigt einen Client (Anwendungsinstanz C) und einen Backup Server (Anwendungsinstanz S) eines Online Daten Backupdienstes, welcher zunächst wieder von S durch Erzeugen eines Endpunktes angeboten wird.

```
BIND("backmeup://cloud.storage.de",
     GET, "ReliableTransport", "ReliableBackgroundTransport",
     PUT, "ReliableTransport", "ReliableBackgroundTransport")
```

Durch den Aufruf von `addParameters()` auf dem zuvor per BIND Primitive erstellten Endpunkt, hinterlegt der Backup Server S direkt im Anschluss die für die URI `backmeup://cloud.storage.de` verfügbaren Parameter.

In diesem Anwendungsfall wird davon ausgegangen, dass der Client C verschiedene Backupdienste aus dem *Namensraum* `backmeup` unterstützt. Im Falle vom *Anbieter* `cloud.storage.de` sind ihm aber die unterstützten Transportdienste nicht bekannt, da verschiedene Anbieter im Namensraum sowohl `PUT` als auch `CONNECT` zum hochladen

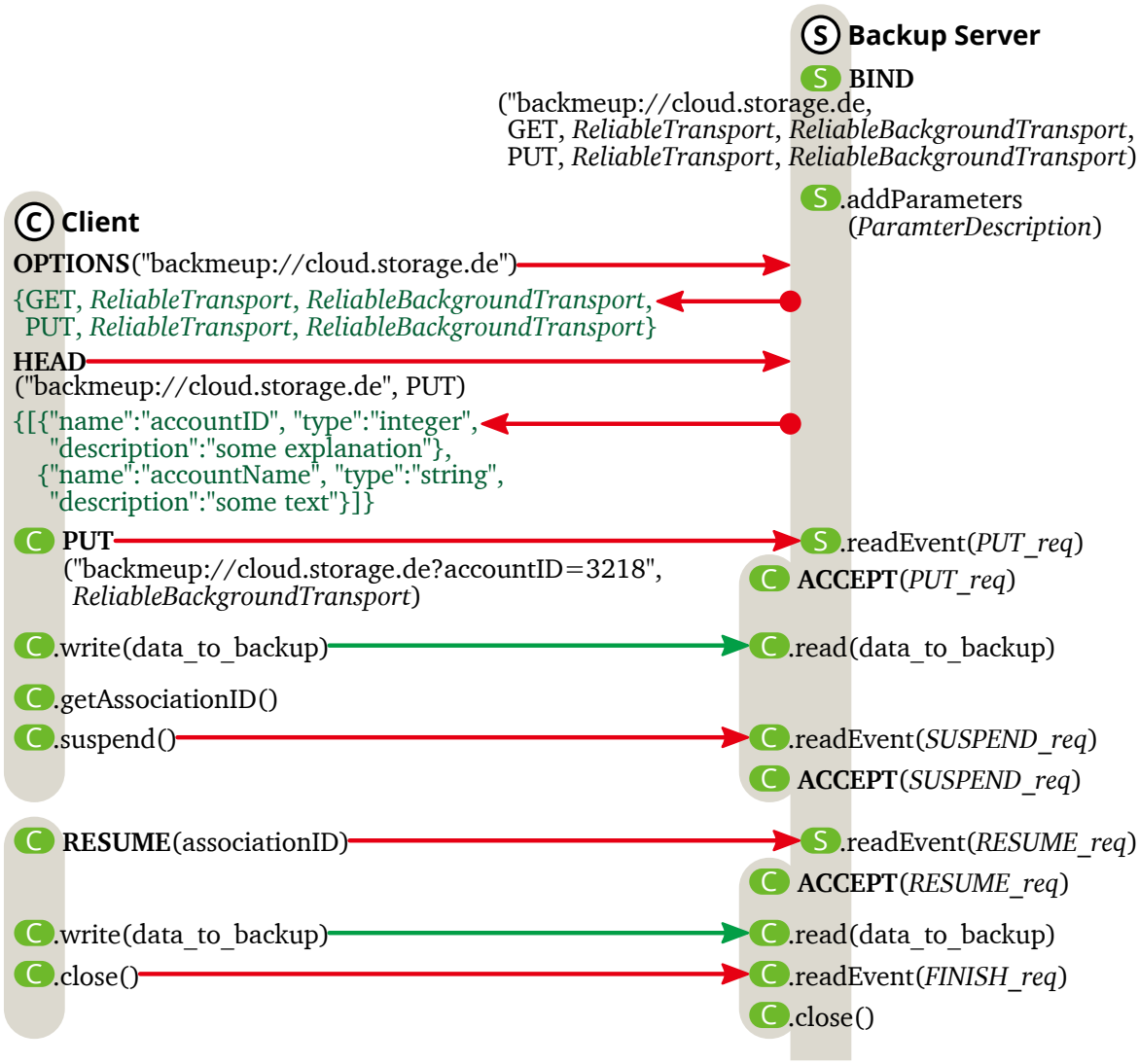


Abbildung 3.9: *Online Daten Backupdienst.*

von Daten auf den eigenen Server unterstützen. Als erstes fragt C daher per `OPTIONS` die unterstützten Dienstprimitive ab. Der *Anbieter* `cloud.storage.de` unterstützt nur lesenden oder schreibenden Zugriff per `GET` und `PUT` auf den Online Speicher und keinen bidirektionalen Zugriff per `CONNECT`, welcher im Namensraum `backmeup` beispielsweise zum Synchronisieren von Daten zwischen Client und Server von anderen *Anbietern* verwendet wird.

Anschließend ruft Anwendungsinstanz C `HEAD` auf um Informationen über den angebotenen Anwendungsdienst in Form der verfügbaren Parameter abzurufen. Mit dieser Information kann C nun per `PUT` eine Kommunikationsassoziation zu S aufbauen und seine Daten über den gewählten *ReliableBackgroundTransport* Dienst an den Online Speicher

senden. Da Client C wenig später das Backup pausieren möchte, holt sich die Anwendungsinstanz zunächst durch den Aufruf von `getAssociationID()` auf dem Endpunkt die zugehörige AssoziationsID und teilt dem Server S anschließend durch den Aufruf von `suspend()` mit, dass die Kommunikationsassoziation pausiert werden soll.

Zu einem späteren Zeitpunkt kann Client C durch den Aufruf der Dienstprimitive `RESUME` mit der zuvor per `getAssociationID()` erhaltenen AssoziationsID die ursprüngliche Kommunikationsassoziation wiederaufnehmen. Nachdem der Server S dies als erneute Verbindungsanfrage erhalten und die entsprechende Assoziation wiederaufgenommen hat, kann Client C mit dem Senden der Daten fortfahren. Abschließend wird die Kommunikationsassoziation von C durch den Aufruf von `close()` auf dem Endpunkt beendet.

3.6.3 Video-On-Demand Streamingdienst

Abbildung 3.10 zeigt einen Server des Video-On-Demand Anbieters Netzfliks (Anwendungsinstanz S) und einen Mediaplayer, der ein Video abrufen möchte (Anwendungsinstanz M).

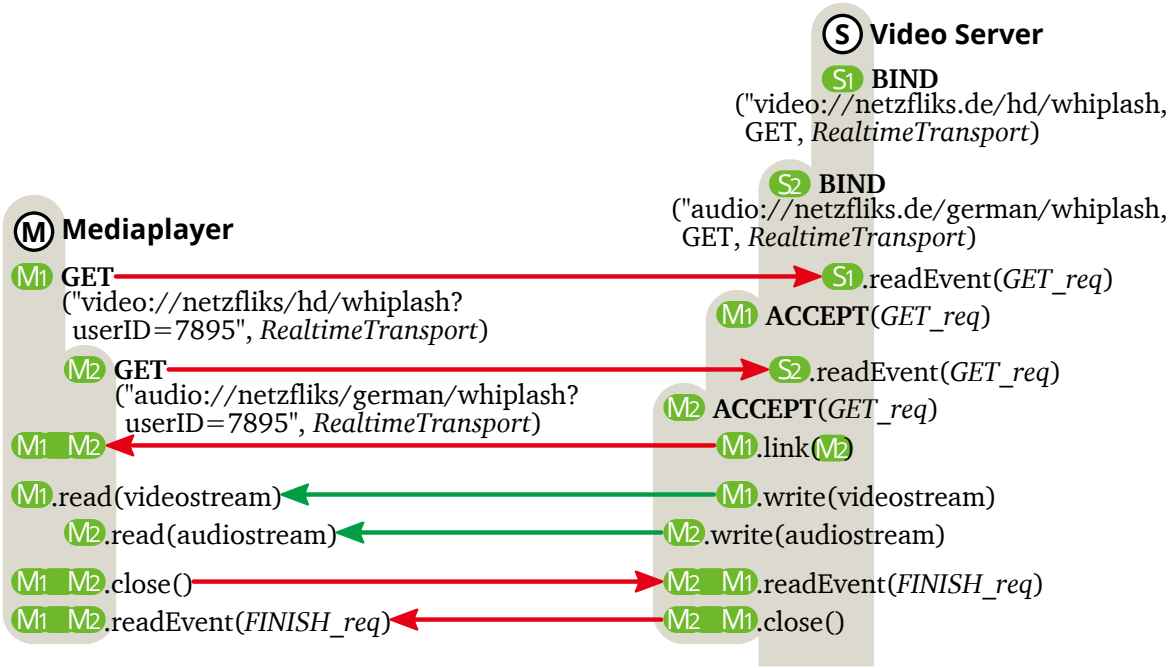


Abbildung 3.10: *Video-on-Demand Streamingdienst.*

Server B erzeugt Endpunkte für verschiedene Versionen einzelner Videos. Dabei befinden sich Bild- und Tonspuren jeweils in ihrem eigenen Namensraum (*video* respektive *audio*). In Abbildung 3.10 ist zu sehen wie S einen Endpunkt für den Videostrom der

HD-Version des Filmes *whiplash* erzeugt. Analog dazu wird im *audio* Namensraum eine deutsche Tonspur des Filmes angeboten.

Es wird angenommen das Mediaplayer M zuvor bereits den *Anbieter* Netzfliks kontaktiert hat, um den gewünschten Film in der gewünschten Qualität anzufordern. Auf diese Weise hat M die URIs der entsprechenden Audio- und Videoströme erhalten. Dabei hat der *Anbieter* Netzfliks auch die Zugriffsrechte von M auf seinen Dienst überprüft und M hat in diesem Beispiel eine eindeutige ID zur Identifikation für die folgenden Transaktionen erhalten, welche Netzfliks im Anfrageteil der URI erwartet.

Durch den Aufruf von `GET` einmal für den Video- und einmal für den Audiostrom (jeweils unter Angabe der eindeutigen ID) können diese vom Server S erhalten werden. Eine solche Aufteilung ist sinnvoll, da Video- und Audioströme über unterschiedliche Transportdienste bereitgestellt werden können. Die Quellen müssen auch nicht notwendigerweise auf dem gleichen Server liegen. Audio- und Videodaten können genauso gut von zwei verschiedenen Servern stammen. Während die Tonspuren auf einem regionalen Server liegen können, ist der Videostrom beispielsweise global identisch. Das kann in der Praxis dazu führen, dass die Daten der beiden Kommunikationsassoziationen über unterschiedliche Transportdienste übertragen werden und sehr unterschiedliche Wege durchs Netz nehmen.

Um dem Netzwerk beziehungsweise dem PASTE Rahmenwerk mitzuteilen, dass die beiden Datenströme dennoch zusammen gehören, verbindet der Server S den Endpunkt M2 mit dem Endpunkt M1 nach deren Erzeugung durch den Aufruf von `link()`. Diese Information kann im PASTE Rahmenwerk beispielsweise dazu verwendet werden die Kommunikationsassoziation M2 zu schließen für den Fall, dass die Kommunikationsassoziation M1 durch einen Fehler beendet werden muss. Zudem erreichen Fehler beider Datenströme über den Ereigniskanal auch den jeweils anderen Datenstrom. Tritt bei einem der Datenströme beispielsweise eine Verzögerung ein, kann bereits unterhalb der Anwendungsschicht darauf reagiert werden, indem der andere Datenstrom zunächst vor der Auslieferung gepuffert wird.

Im dargestellten Beispiel, bei dem beide Ströme vom gleichen Endsystem stammen und über den gleichen Transportdienst übertragen werden, bringt die `Link` Information weitere Vorteile. Beispielsweise kann der Transportdienst im Netzwerk über ein Substromfähiges Protokoll realisiert werden und eine Transportverbindung für beide Datenströme verwendet werden.

3.6.4 Publish-Subscribe Dienst

Abbildung 3.11 zeigt wie sich mit Hilfe der PASTE-API auch komplexere Kommunikationsmuster wie beispielsweise eine *Publish/Subscribe* basierte Kommunikation umsetzten lassen.

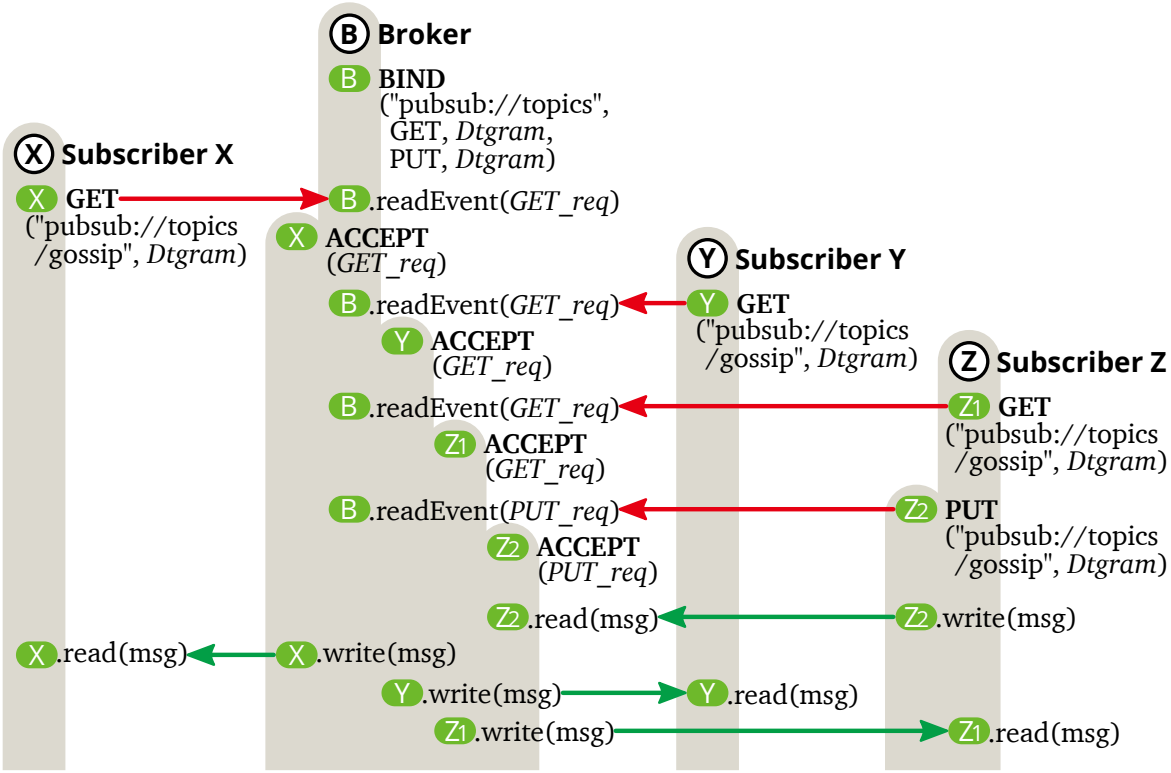


Abbildung 3.11: *Publish-Subscribe Dienst mit Broker B und Subscribern X, Y & Z.*

Anwendungsinstanz B fungiert in diesem Fall als Broker und macht unterschiedliche Themen unter verschiedenen Endpunkten verfügbar. Per `BIND` wird ein Endpunkt für die URI `pupsub://topics` erzeugt. Die Anwendungsinstanzen X, Y und Z abonnieren das Thema *gossip* indem sie durch den Aufruf von `GET` eine Kommunikationsassoziation zum Broker B aufbauen.

Anwendungsinstanz Z baut anschließend eine zweite Kommunikationsassoziation zum Broker B auf, verwendet dafür aber die `PUT` Dienstprimitive, um ein Datum zu publizieren. Wie im Instant Messaging Szenario (Abschnitt 3.6.1) kann der Broker B alle zu einem bestimmten Thema gehörigen Kommunikationsassoziationen anhand der URI `pupsub://topics` zuordnen. Über die für eine Kommunikationsassoziation verwendete Dienstprimitive (also `GET` oder `PUT`) kann zwischen *Publishern* und *Subscribern* unterschieden werden. Über `PUT` eingehende Daten werden mit Hilfe von `GET` Assoziationen weiterverteilt.

3.7 Zusammenfassung

In diesem Kapitel wurde der heutige Stand der Schnittstellen zwischen Anwendungen und dem Netzwerk und die damit verbundenen Probleme näher betrachtet. Als Lösung wurde die PASTE-API entworfen und ausführlich vorgestellt. Sie wurde mit dem Ziel entworfen, Anwendungen stärker vom Netzwerk zu entkoppeln, indem sie Anwendungen eine abstraktere Sicht auf das Netzwerk bietet. Damit können sowohl auf der Anwendungsschicht als auch im Netzwerk Anreize geschaffen werden, zukünftig neue Protokolle schneller zu unterstützen.

Anwendungsentwickler wird die Möglichkeit gegeben ihre Anforderungen an das Netzwerk beziehungsweise an den Datentransportdienst Protokoll-agnostisch zu formulieren, also ohne bereits zur Entwicklungszeit ein dediziertes Protokoll wählen zu müssen, über das später kommuniziert werden soll. Dazu wurde konzeptuell eine einfache Methode beschrieben, Transportdienstbeschreibungen anzufertigen. Netzwerkbetreiber erhalten so die Möglichkeit in Teilen ihres Netzwerks unterschiedlichste Protokolle einzusetzen und beim verfügbar werden besserer Varianten gegen diese auszutauschen.

Abschließend wurde die Anwendung der PASTE-API anhand einiger Anwendungsfälle aufgezeigt. Zwar erfordert der Umstieg von der Socket- auf die PASTE-API zunächst einen Mehraufwand durch die benötigte Anpassung der Anwendungen, welche aber im Gegenzug an Flexibilität gewinnen und an Komplexität, welche ins Netzwerk verlagert wird, verlieren.

Kapitel 4

Das PASTE Rahmenwerk

In Kapitel 3.1 wurden Probleme der heutigen Anwendungsschnittstelle zwischen Netzwerk und Anwendungen in Form der Berkeley Socket-API herausgestellt. Des Weiteren wurde HTTP als weit verbreitete Alternative zum Datenaustausch zwischen Anwendungen betrachtet.

Beiden Ansätzen ist gemein, dass Anwendungen bereits zur Designzeit auf eine starre Auswahl an Protokollen festgelegt sind. Weder die Berkeley Socket-API noch HTTP erlauben die Wahl verschiedener Transportprotokolle zur Laufzeit. Abhängig vom Kontext in dem eine Anwendung ausgeführt wird, kann es aber sehr sinnvoll sein, zwischen unterschiedlichen Protokollen zu wählen. Sowohl aus Sicht des Netzwerkbetreibers als auch des Anwendungsentwicklers ist dieser Umstand problematisch:

- Anwendungsentwickler müssen Anwendungen aufwendig anpassen um Gebrauch von zukünftigen Protokollen machen zu können. Wünschenswert wäre es hingegen, wenn Anwendungen zukünftig automatisch verschiedene Dienste abhängig vom Ausführungskontext nutzen könnten, ohne angepasst werden zu müssen.

- Netzbetreiber hingegen hätten einen Vorteil, wenn sie zwischen Protokollen, welche den gleichen Dienst erbringen, wählen könnten. Abhängig von Netz-spezifischen Parametern, wie beispielsweise der Auslastung des Netzes in verschiedenen Regionen und zu verschiedenen Tageszeiten, könnte das passendere Protokoll gewählt werden. Auch beim Verfügbar werden effizienterer Protokolle für einen bestimmten Dienst, sollte das Protokoll im Optimalfall einfach gewechselt werden können, ohne Auswirkungen auf existierende Anwendungen zu haben.

Ein sich daraus ergebender Nachteil der heute gängigen Anwendungsschnittstellen ist, dass die Einführung von neuen Protokollen behindert wird. Diese können nicht automatisch genutzt werden, ohne existierenden Anwendungen anpassen zu müssen. Diese

Anpassungen werden aber im Gegenzug nicht oder nur sehr selten vorgenommen, da die Unterstützung neuerer Protokolle in Form von einer großflächigen Verfügbarkeit häufig nicht gegeben ist. Dies führt dazu, dass es Innovationen unterhalb der Anwendungsschicht im heutigen Internet sehr schwer haben sich durchzusetzen.

Ein weiteres Problem der heutigen Anwendungsschnittstellen ist, dass Anwendungen sich selbst um die Verbindungsverwaltung kümmern müssen. Sollen verschiedene Transportprotokolle verwendet werden, müssen Anwendungen diese jeweils explizit unterstützen und zur Laufzeit selbstständig entscheiden, welches eingesetzt werden soll. Verbindungsabbrüche, welche beispielsweise durch den Wechsel von Zugangsnetzen oder des verwendeten Netzwerkinterfaces eines Gerätes auftreten, müssen ebenso behandelt werden, indem Anwendungen selbstständig Verbindungen wiederherstellen beziehungsweise neue Verbindungen aufbauen. Damit erledigen Anwendungen heute Aufgaben, die klassischer Weise auf der Sitzungsschicht verortet sind. Diese ist zwar im ISO/OSI Schichtenmodell oberhalb der Transportschicht vorgesehen, existiert aber im heutigen von der TCP/IP-Protokoll-Familie dominierten Internet de-facto nicht.

Die in Kapitel 3 vorgestellte PASTE-API wurde entworfen, um Anwendungen stärker vom Netzwerk zu entkoppeln, indem sie Anwendungen eine abstraktere Sicht auf das Netzwerk bietet. Damit können sowohl auf der Anwendungsschicht als auch im Netzwerk Anreize geschaffen werden, zukünftig neue Protokolle schneller zu unterstützen. Anwendungsentwickler wird die Möglichkeit gegeben ihre Anforderungen an das Netzwerk beziehungsweise an den Transportdienst generisch zu formulieren, ohne direkt ein dediziertes Protokoll wählen zu müssen. Netzwerkbetreiber erhalten so die Möglichkeit in Teilen ihres Netzwerks unterschiedlichste Protokolle einzusetzen und beim Verfügbar werden besserer Varianten diese auszutauschen. Beispielsweise könnte der gleiche Transportdienst durch ein alternatives Protokoll mit einer wesentlichen geringeren Auslastung der Infrastruktur erbracht werden. Sofern sich am erbrachten Dienst nichts ändert, nutzen existierende Anwendungen das neue Protokoll in diesem Fall automatisch. Einige der in Kapitel 3.3 vorgestellten Dienstprimitive benötigen jedoch Unterstützung vom Netzwerk, um umgesetzt werden zu können. Im Folgenden soll daher mit dem PASTE Rahmenwerk eine Lösung vorgestellt werden, welche genau diese Unterstützung bietet. Das PASTE Rahmenwerk übernimmt dabei Aufgaben der Sitzungsschicht aber auch der Transport- und Vermittlungsschicht und verlagert heute häufig auf der Anwendungsschicht erledigte Aufgaben wieder ins Netzwerk und damit unterhalb der Anwendungsschicht. Dies stellt einen weiteren Vorteil für den Anwendungsentwickler dar, da dieser sich um weniger Netzwerk spezifische Details kümmern muss. Zwar erfordert der Umstieg von der Socket- auf die PASTE-API zunächst einen Mehraufwand durch die Anpassung der Anwendung, welche aber im Gegenzug an Flexibilität gewinnt und Komplexität im Netzwerk spezifischen Teil verliert.

Folgende Anforderungen an das PASTE Rahmenwerk ergeben sich aus den obigen Feststellungen und implizit auch aus der in Kapitel 3 vorgestellten PASTE-API:

(1) Anwendungsinstanzen sollten einander über einen eindeutigen Namen beziehungsweise Identifikator ansprechen können. Dieser sollte unabhängig vom eingesetzten Adressschema sein, so dass dieses für die Anwendungen transparent ist. Zusätzlich sollte der Identifikator unabhängig von Adressänderungen über die gesamte Dauer eines Datenaustausches hinweg konstant sein.

(2) Die Namensauflösung, also die Abbildung eines Identifikators auf einen Lokator beziehungsweise eine Adresse, soll unterhalb der Anwendungsschicht passieren.

(3) Anwendungen sollten einen Protokoll-agnostischen Transportdienst statt konkreter Protokolle beim Öffnen eines Netzwerkendpunktes anfordern.

(4) Eine Verbindung beziehungsweise eine Kommunikationsassoziation zwischen zwei (oder mehr) Anwendungsinstanzen sollte unabhängig sein von:

- der Laufzeit einzelner an der Assoziation beteiligter Anwendungsinstanzen
- konkreten zum Datenaustausch verwendeten Protokollinstanzen im Netzwerk
- dem verwendeten Netzwerkinterface

(5) Kommunikationsassoziationen werden unterhalb der Anwendungsschicht verwaltet, um eine bessere Unterstützung für Mobilität, Multi-homing und Techniken wie beispielsweise Multicast oder Geocast zu erreichen.

(6) Anwendungen sollten optimaler Weise davon ausgehen können, grundsätzlich und dauerhaft mit einem Netzwerk verbunden zu sein, sofern über mindestens ein Netzwerkinterface Konnektivität hergestellt werden kann. Dabei sollen auftretende Fehler im Netzwerk soweit wie möglich unterhalb der Anwendungsschicht für die Anwendung transparent behandelt werden.

Wie lassen sich diese Anforderungen umsetzen?

(1) Durch die Verlagerung eines Namensauflösungsdienstes aus der Anwendungsschicht in die darunterliegenden Schichten, können Adressen beziehungsweise Lokatoren erfolgreich vor Anwendungen verborgen werden. Das PASTE Rahmenwerk kümmert sich in diesem Fall um die Abbildung eines Namens beziehungsweise eines Identifikators auf eine entsprechende Adresse des verwendeten Adressschemas. Gleichzeitig übernimmt es die Aufgabe im Falle einer Adressänderung etwaige Einträge bei allen verwendeten Namensdiensten zu aktualisieren. Dadurch wird auch die Unterstützung von Mobilität (also mobiler Endgeräte) erleichtert.

(2) Beim Öffnen eines Netzwerkendpunktes übergeben Anwendungen zusätzlich zum Namen (beziehungsweise Identifikator) den vom Netzwerk gewünschten Transportdienst. Dieser kann, wie in Kapitel 3.5 beschrieben, aus einer (erweiterbaren) Liste von vordefinierten und verfügbaren Diensten per Transportdienstidentifikator gewählt werden; Als parametrierbare Transportdienstschablone übergeben werden, um einen Dienst an die Bedürfnisse der kommunizierenden Anwendung anzupassen; Oder es kann eine vollständige individuelle Transportdienstbeschreibung des gewünschten Dienstes übergeben werden. Transportdienstschablonen und -beschreibungen können dabei auch Parameter enthalten, welche potentiell mit der Gegenseite ausgehandelt werden müssen. Das PASTE Rahmenwerk kann dann ein geeignetes Protokoll wählen, welches in der Lage ist den beschriebenen Dienst mit den gewünschten Parametern zu erbringen.

(3) Kommunikationsassoziationen (im Folgenden auch kurz: Assoziationen) zwischen Anwendungsinstanzen kapseln die Information, welche Anwendungsinstanz Daten über einen bestimmten Transportdienst mit einer anderen Anwendungsinstanz austauscht. Dabei abstrahieren Kommunikationsassoziationen von konkreten Adressen beziehungsweise Adressschemata und dedizierten Protokollen. Die Details darüber, wie ein Datenaustausch zu bewerkstelligen ist obliegen dem PASTE Rahmenwerk. Durch die Einführung eines Assoziationszustands sowie der Verwaltung der mit einer Assoziation verbundenen Transportverbindungen unterhalb der Anwendungsschicht, wird eine Entkopplung vom Zustand einer konkreten Anwendungsinstanz erreicht. Assoziationen können zum Beispiel pausiert oder für eine Weile schlafen gelegt und später wiederaufgenommen werden, wenn die zugehörige Anwendungsinstanz zwischenzeitlich pausiert oder vorübergehend beendet wird.

(4) Damit eine Anwendung im Normalfall davon ausgehen kann, konstant mit dem Netzwerk verbunden zu sein, sollten so viele Netzwerkprobleme und Protokoll bedingte Fehler wie möglich unterhalb der Anwendungsschicht behandelt werden. In dem Fall, dass ein Datenaustausch gerade nicht möglich ist, können weiter Daten entgegengenommen und lokal gepuffert werden, sofern genug Speicherplatz dafür zur Verfügung steht. Parallel können alternative Verbindungen geöffnet werden, um fehlerhafte zu ersetzen. Sofern der Fehler bei der Gegenseite liegt, können Daten nicht nur lokal, sondern zum Beispiel auch zeitweise im Netzwerk gepuffert werden, bis die Gegenseite wieder verfügbar ist. Dazu kann das Netzwerk verschiedene Dienste und damit Ressourcen im Netz anbieten, um Endsysteme bei der Realisierung von Assoziationen zu unterstützen. Kann ein Fehler nicht ohne Anwendungsunterstützung behandelt werden und muss an die Anwendungsinstanz hoch gereicht werden, so kann das PASTE Rahmenwerk dieser optional vorhandene

Kontextinformationen geben, um die Anwendungsinstanz bei der Fehlerbehandlung zu unterstützen.

4.1 Protocol Agnostic Services & Transport Enrichment

Im Folgenden wird das Protocol Agnostic Services & Transport Enrichment (PASTE) Rahmenwerk vorgestellt, welches häufig zu lösende Aufgaben der Anwendungsschicht übernimmt und damit Anwendungen eine einfachere Sicht auf das Netzwerk ermöglicht, indem von unterliegenden Mechanismen und Problemen weitgehend abstrahiert wird. Gleichzeitig erhalten Anwendungen Zugriff auf eine Vielzahl unterschiedlicher Datentransportdienste, deren Umsetzung für die Anwendungen transparent unterhalb der Anwendungsschicht auf der Transportschicht geschieht. Die Datentransportdienste werden dabei durch verschiedene Transportprotokolle der Transportschicht erbracht. PASTE kümmert sich dabei nicht nur um den reinen Datentransport, sondern auch um die Verbindungs- beziehungsweise Assoziationsverwaltung und übernimmt damit klassische Aufgaben der Sitzungsschicht.

Zusätzlich kann PASTE Dienst-unterstützende Komponenten im Netz bereitstellen. Diese *Companiondienste* werden von PASTE Instanzen bereitgestellt, welche zum Beispiel auf Routern oder allgemein im Netz befindlichen “Middleboxen” betrieben werden können. Als *Companion* werden im Folgenden Systeme bezeichnet, auf denen eine PASTE Instanz betrieben wird, welche einen oder mehrere *Companiondienste* bereitstellt.

In Abschnitt 4.9 wird detailliert anhand eines Anwendungsfalls der Einsatz der *Companions* vorgestellt und erläutert. Diese können zum Beispiel zum Zwischenspeichern von im Transit befindlichen Daten verwendet werden, um kurze Ausfälle von Endgeräten zu kompensieren. Die bereitgestellten Dienste bieten generische Lösungen für Datentransportprobleme und benötigen dementsprechend kein spezifisches Wissen über den Inhalt der zu transportierenden Daten auf der Anwendungsebene. Dadurch können Anwendungsentwickler beziehungsweise Anwendungen über die PASTE-API transparent auf diese Dienste zugreifen, unabhängig von den zu transportierenden Daten.

Abbildung 4.1 zeigt ein Netzwerk mit vier aktiven Endgeräten (*E1 - E4*) und PASTE spezifischen Systemen im Netzwerk. Auf sämtlichen Endgeräten läuft eine PASTE Instanz und stellt verschiedenen Anwendungsinstanzen Transportdienste über die in Kapitel 3 vorgestellte PASTE-API zur Verfügung. Das PASTE Rahmenwerk befindet sich dabei zwischen der Transport- und Anwendungsschicht vergleichbar mit Schicht 5 des ISO/OSI Modells. Wie im heutigen Internet befindet sich im Netzwerk ein Namensdienst, bei dem es sich im Falle eines IP Netzes um den üblichen DNS Dienst handeln kann. Für andere Netze, wie zum Beispiel “Named Data Networking” basierte Netze, sind aber auch alternative Namensdienste denkbar. Die Namensauflösung wird PASTE intern daher

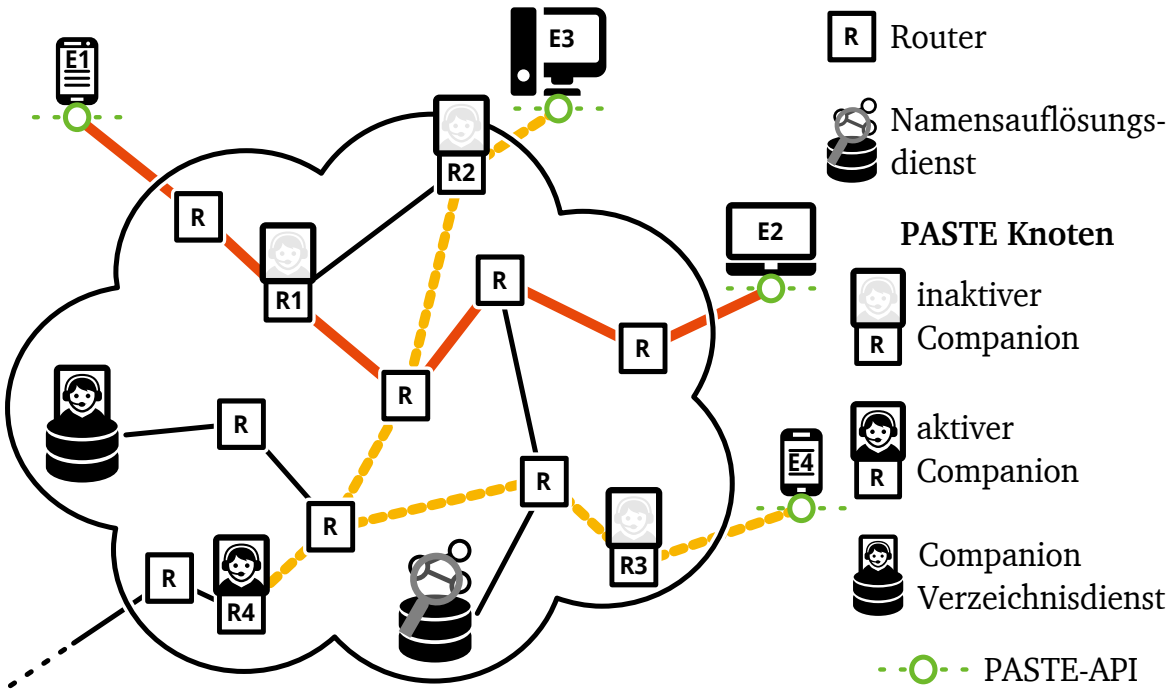


Abbildung 4.1: *PASTE Komponenten im Netzwerk.*

in Namensdienst spezifischen Komponenten gekapselt, welche über eine einheitliche Schnittstelle angesprochen werden können.

Zum Bereitstellen der Companiondienste in einem Netzwerk wird ein weiterer Verzeichnisdienst benötigt, über den bei Bedarf *Companions* gesucht werden können. Auf Zwischensystemen laufende PASTE Instanzen im Netzwerk registrieren dazu von ihnen unterstützte Companiondienste an diesem Companionverzeichnisdienst. Andere PASTE Instanzen können dann später gezielt mit Hilfe verschiedener Parameter, wie zum Beispiel verfügbarem Speicherplatz oder verfügbarer Rechenleistung, nach geeigneten *Companions* für einen bestimmten Dienst suchen. Dazu ist es notwendig, dass in regelmäßigen Abständen Ressourcen- und Zustandsinformationen zwischen einzelnen Companiondienste bereitstellenden PASTE Instanzen und dem Companionverzeichnis ausgetauscht werden.

Es ist denkbar den Verzeichnisdienst mit dem Namensdienst zu kombinieren. In IP-basierten Netzen zum Beispiel, könnte diese Information ebenfalls im DNS abgelegt werden. Grundsätzlich kann dabei jede PASTE Instanz Companiondienste anbieten, unabhängig davon ob sie auf Endsystemen am Netzrand oder Zwischensystemen wie Routern und *Middleboxes* im Netz eingesetzt wird. Im Folgenden wird jedoch davon ausgegangen, dass primär stationäre Systeme mit einer hohen Verfügbarkeit PASTE Instanzen betreiben, welche Companiondienste anbieten, da es ihre Hauptaufgabe ist schwächere Endsysteme am Netzrand zu entlasten und beim Datentransport zu unterstützen.

In Abbildung 4.1 sind sowohl Router mit (*R1 - R4*) als auch ohne PASTE Unterstützung zu sehen. Da PASTE für einzelne Anwendungsinstanzen Transportverbindungen aufbaut, muss nicht auf jedem System im Netz eine PASTE Instanz laufen. Wird von einer PASTE Instanz gerade kein bereitgestellter Companiondienst in Anspruch genommen, fungiert ein PASTE-unterstütztes System als "normaler" Router. Der Datenaustausch zwischen dem Smartphone *E1* und dem Notebook *E2* verwendet zum Beispiel eine direkte, über PASTE aufgebaute, Transportverbindung. Diese wird lediglich von den beiden PASTE Instanzen auf den Endsystemen selbst verwaltet, ohne dass die PASTE Instanz auf dem Router *R1* aktiv am Datentransport beteiligt ist. Bei der Kommunikation zwischen dem PC *E3* und dem zweiten Smartphone *E4* hingegen kommt ein *Companion* zum Einsatz. Dieser könnte zum Beispiel die Daten des breitbandiger angebunden PCs *E3* zwischenspeichern und verzögert an das Smartphone ausliefern, damit der PC seine zur Verfügung stehende Bandbreite besser ausnutzen kann. In diesem Fall sind drei PASTE Instanzen an der Datenübertragung beteiligt. Jeweils eine auf den Endsystemen *E3* und *E4*, sowie eine dritte auf dem Router *R4*, welche den Companiondienst bereitstellt. Je nach eingesetztem Companiondienst kann dieser alleine von *R4* oder gemeinsam mit einem der Endsysteme *E3* oder *E4* erbracht werden. Im hier abgebildeten Fall existiert zum Beispiel eine Transportverbindung zwischen *E3* und *R4* und eine weitere zwischen *R4* und *E4*.

PASTE kapselt vom Betriebssystem bereitgestellte Transportprotokolle wie zum Beispiel TCP, UDP oder SCTP in Assoziationen. Diese halten auf den End- und Companionsystemen sämtliche für den Datenaustausch relevanten Zustände. Zudem abstrahieren sie von den konkret verwendeten Transportprotokollen und kümmern sich um den Verbindungsaufbau und -abbau verschiedener Transportverbindungen über unterschiedliche Transportprotokolle. Um über PASTE kommunizieren zu können und dabei Gebrauch von den durch PASTE bereitgestellten Diensten zu machen, müssen daher alle am Datenaustausch direkt beteiligten Systeme ebenfalls PASTE verwenden.

Bei den *Companions* handelt es sich, wie bereits erwähnt, um Zwischensysteme, die auf dem Datenpfad zwischen zwei oder mehreren Endsystemen liegen und diese beim Datentransport unterstützen. Eine auf einem Zwischensystem betriebene PASTE Instanz stellt verschiedene Companiondienste zur Verfügung. Unter dem Datenpfad ist beziehungsweise sind dabei der oder die Pfade zu verstehen, die die einzelnen Datenpakete durchs Netz nehmen. Dabei müssen nicht sämtliche Daten notwendigerweise über dieselben Knoten im Netzwerk laufen. Lediglich am Dienst beteiligte PASTE Instanzen, welche Companiondienste bereitstellen, sowie natürlich auf Sender- und Empfängerseite, werden von jedem Paket passiert.

Jedes System, auf dem PASTE verfügbar ist, kann prinzipiell als *Companion* fungieren. Um nützlich zu sein, sollten diese jedoch nicht nur eine hohe Verfügbarkeit aufweisen, sondern je nach angebotenem Dienst auch in der Nähe der Endnutzer oder des Netzkerns platziert sein und über ausreichend Ressourcen verfügen um einen bestimmten

Companiondienst erbringen zu können. Dies kann je nach Companiondienst ausreichend verfügbare Rechenleistung oder zum Beispiel lokal vorhandener Pufferspeicherplatz sein.

Als Betreiber von *Companions* bieten sich daher vor allem ISPs (Internet Service Provider) oder Drittanbieter wie existierende Cloud Infrastruktur Anbieter an. Beide könnten den Zugriff auf *Companions* zum Beispiel kostenpflichtig oder als Teil eines Netzanschlusses verfügbar machen. Genauso ist es denkbar, dass private *Companions* innerhalb von Firmennetzen eingesetzt werden. Nicht zuletzt hat aber auch jede Privatperson die Möglichkeit, ihre eigenen bevorzugten *Companions* zu betreiben. Zum Beispiel in Form von einem privaten dedizierten Server im Internet oder als Teil eines Routers zuhause im Heimnetz.

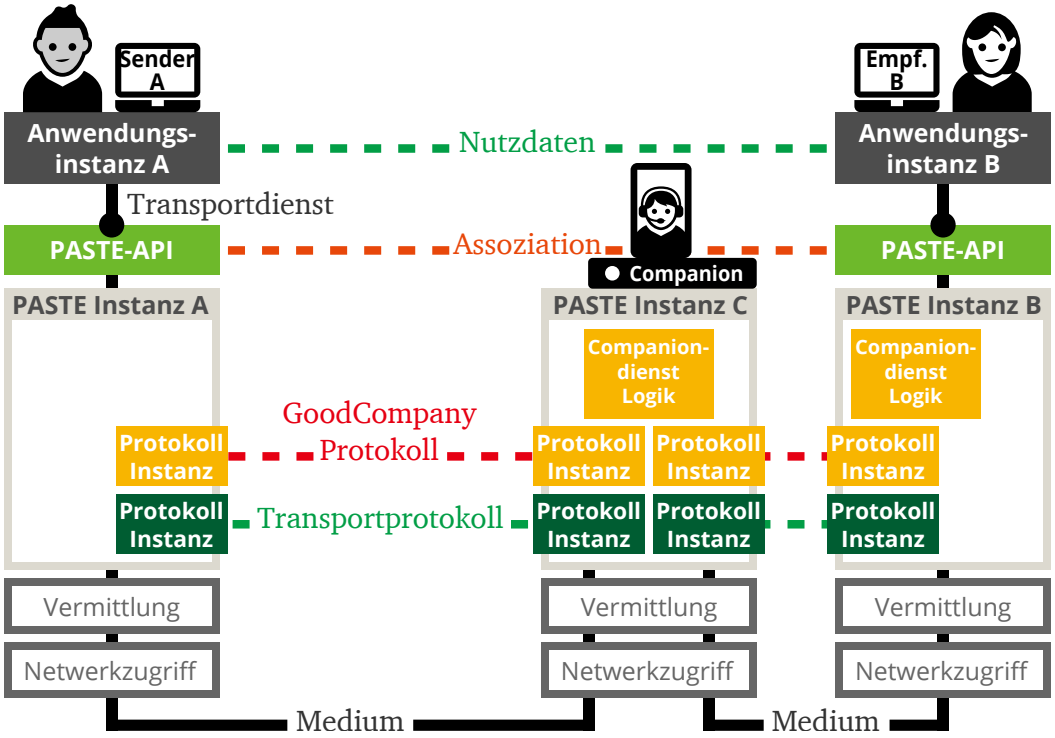


Abbildung 4.2: *Eine Kommunikationsassoziation zwischen zwei Anwendungsinstanzen mit Companion Unterstützung.*

Abbildung 4.2 zeigt vereinfacht die Kommunikation zwischen zwei Anwendungsinstanzen mit drei beteiligten PASTE Instanzen. PASTE bietet über die PASTE-API verschiedene protokollagnostische Transportdienste an. Über die Anwendungsschnittstelle fordert Anwendungsinstanz A eine Assoziation zu Anwendungsinstanz B an. PASTE Instanz A legt daraufhin ein zugehöriges Assoziationszustandsobjekt an, welches verschiedene Transportdienstparameter verwaltet. Anschließend instanziiert PASTE ein geeignetes konkretes Transportprotokoll, welches die Eigenschaften des von Anwendungsinstanz A angeforderten Transportdienstes erfüllen kann. Zunächst wird so eine Transportverbindung zur

PASTE Instanz B aufgebaut. Vorab muss Anwendungsinstanz B sich als empfangsbereit für den selben Transportdienst an PASTE Instanz B registriert haben. Nachdem alle relevanten Assoziationszustandsinformationen an PASTE Instanz B übertragen wurden, können Nutzdaten zwischen Anwendungsinstanz A und B ausgetauscht werden. Der Austausch der Assoziationszustandsinformationen zwischen PASTE Instanz A & B findet über das *GoodCompany* Protokoll statt. Es dient zum Austausch sämtlicher Signalisierungsdaten zwischen verschiedenen PASTE Instanzen und sitzt über der Transportschicht.

Soll zu einem späteren Zeitpunkt ein *Companion* zur Diensterbringung oder zur Dienstunterstützung eingesetzt werden, wird dieser von einer der beiden PASTE Instanzen angefordert. PASTE Instanz A würde beispielsweise zunächst per Companionverzeichnis einen passenden *Companion* suchen und dann kontaktiert. In dem Zuge werden ebenfalls alle relevanten Assoziationszustandsinformationen per *GoodCompany* Protokoll an PASTE Instanz C übertragen, die Transportverbindung zwischen PASTE Instanzen A & B abgebaut und neue Transportverbindungen jeweils zwischen PASTE Instanzen A & C und PASTE Instanzen C & B aufgebaut, angestoßen von PASTE Instanz A beziehungsweise PASTE Instanz C. Zudem werden benötigte Companiondienst Logik Komponenten instanziiert. Abhängig vom erbrachten Dienst können diese sich auf nur einem System befinden (beispielsweise zum Puffern von Daten) oder an den Enden eines (Teil)pfades (beispielsweise, wenn Daten von einem Zwischensystem komprimiert werden und auf einem der Endsysteme wieder entkomprimiert werden müssen). All das geschieht für die Anwendungsinstanzen transparent, da die Assoziation zwischen den PASTE Instanzen erhalten bleibt. Die Transportprotokollverbindungen bleiben jedoch nicht notwendigerweise konstant über die Lebenszeit einer Assoziation hinweg erhalten. Zudem können auf den unterschiedlichen Teilpfaden (zwischen PASTE Instanzen A & C und PASTE Instanzen C & B) verschiedene Transportprotokolle genutzt werden. Logisch gesehen besteht also zwischen Anwendungsinstanz A & B durchgängig eine Assoziation, welche von den jeweils beteiligten PASTE Instanzen aufrechterhalten wird. Abschnitt 4.5.2 beschreibt die mit einer Assoziation zusammenhängenden Zustandsinformationen genauer, während Abschnitt 4.5.3 ausführlich den Assoziationsaufbau beschreibt.

4.2 PASTE Komponenten

In Abbildung 4.3 sind sämtliche Komponenten aus denen es sich das PASTE Rahmenwerk zusammensetzt dargestellt. Zusätzlich sind jeweils von den Komponenten verwaltete Module und Zustandsobjekte abgebildet.

Als Schnittstelle zwischen Anwendungen und dem PASTE Rahmenwerk findet sich an oberster Stelle die PASTE-API. Einzelne Anwendungsinstanzen erzeugen über Aufrufe

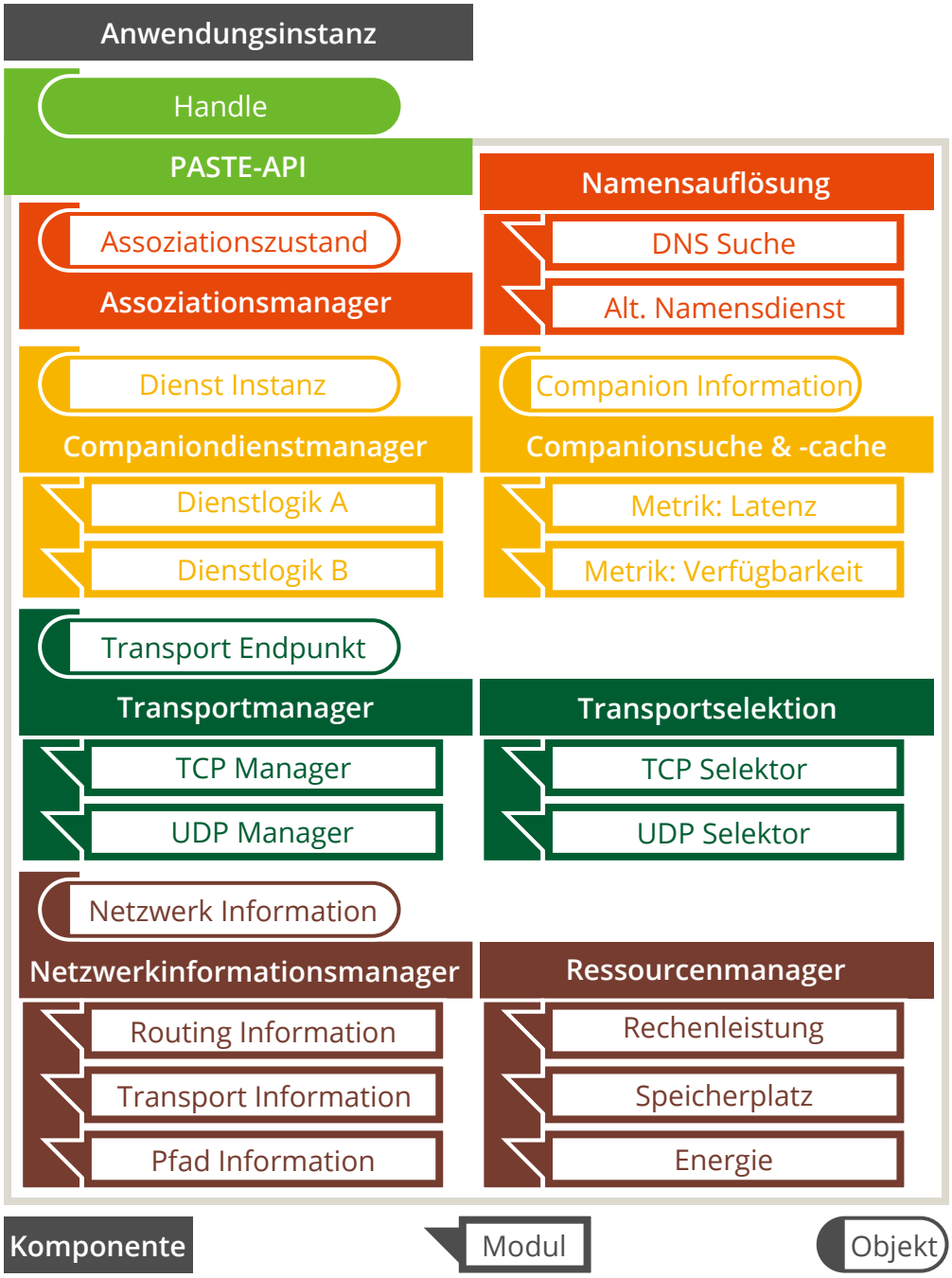


Abbildung 4.3: *Aufbau und Komponenten des PASTE Rahmenwerks.*

der PASTE-API *PASTE-Handles*, über die sie mit Hilfe des PASTE Rahmenwerks auf das Netzwerk zugreifen und untereinander kommunizieren können.

Der *Assoziationsmanager* ist für das Anlegen, Verwalten und Löschen sämtlicher Assoziationszustände in Form von Assoziationszustandsobjekten verantwortlich. Anfragen die

von Anwendungsinstanzen über die PASTE-API gestellt werden, werden hier verarbeitet. Assoziationszustände werden pro Anwendungsinstanz unter einem Sitzungsidentifikator gruppiert und verwaltet, so dass diese, unabhängig davon ob eine Anwendungsinstanz aktuell ausgeführt wird oder nicht, erhalten bleiben. Dabei müssen die Assoziationszustände regelmäßig persistent abgelegt werden, um über Neustarts des Systems auf dem eine PASTE Instanz läuft hinweg, Assoziationen wiederaufnehmen zu können. Diese werden daher regelmäßig bei Änderungen und dem beenden des PASTE Rahmenwerkes lokal auf dem ausführenden Rechner vom PASTE Rahmenwerk gespeichert. Der Assoziationsmanager wird in Abschnitt 4.5 näher beschrieben.

Die *Namensauflösung* kümmert sich um die Abbildung von Namen auf konkrete Adressen unter denen ein Anwendungsdienst verfügbar ist. Durch die modulare Unterstützung verschiedener Namensdienste, werden unterschiedliche Adressräume unterstützt. Dadurch lässt sich PASTE auch leicht um die Unterstützung neuer Adressen beziehungsweise Lokatoren über IPv4/6 hinaus erweitern.

Der *Transportmanager* erzeugt generische Endpunkte für alle von der jeweiligen PASTE Instanz unterstützten Transportprotokolle und kümmert sich um den Aufbau von konkreten Transportverbindungen, welche von einzelnen Kommunikationsassoziationen zum Datenaustausch benötigt werden. Zudem baut er Transportverbindungen auf, über die PASTE Instanzen untereinander Signalisierungsdaten austauschen können. Pro unterstütztem Protokoll wird ein Modul benötigt, welches einen Protokoll-spezifischen Endpunkt anbietet und verwaltet. Im Falle des TCP Moduls wird beispielsweise ein TCP-Serversocket erzeugt, der an einen *„well known“* Port gebunden wird und über IPv4 und IPv6 erreichbar ist. Im Falle des UDP Moduls wird ein UDP-Socket erzeugt, welcher auf dem gleichen *„well known“* Port auf eingehende Pakete lauscht. Die Protokollmodule sind demnach nicht nur Protokoll spezifisch, sondern können auch an bestimmte Adressräume beziehungsweise Lokatoren gebunden sein. Für die Unterstützung neuer Adressräume müssen also auch die Protokollmodule erweitert werden oder neue Module für Transportprotokolle hinzugefügt werden, welche über den alternativen Adressräumen eingesetzt werden können.

Die *Transportselektion* setzt von einer Anwendungsinstanz erhaltene Transportdienstbeschreibungen auf passende Transportprotokolle um. Dazu müssen zwei Bedingungen erfüllt werden. Es muss ein Transportprotokoll gefunden werden, welches die von der Anwendungsinstanz geforderten Anforderungen umsetzen kann. Das Transportprotokoll muss zudem für einen Adressraum verfügbar sein, unter dem alle an der Kommunikationsassoziation beteiligten Endsysteme erreichbar sind. Während die lokal verfügbaren Adressräume der PASTE Instanz bekannt sind, muss zunächst eine Namensauflösung durchgeführt werden um die Erreichbarkeit der Gegenseite in den verschiedenen Adressräumen festzustellen. Anschließend können optionale Parameter der Transportdienstbeschreibung über den Transportmanager ausgehandelt werden, bevor ein zum aktuellen

Zeitpunkt geeignetes Transportprotokoll für die Kommunikationsassoziation gewählt werden kann.

Der *Companiondienstmanager* instanziiert für die von einer PASTE Instanz angebotenen Companiondienste entsprechende Dienstkomponenten, welche die Programmlogik enthalten um den geforderten Companiondienst für eine bestimmte Assoziation umsetzen zu können. Zudem wird eine Auflistung der angebotenen Companiondienste über einen Verzeichnisdienst verfügbar gemacht, so dass eine PASTE Instanz, welche bestimmte Companiondienste anbietet, von anderen PASTE Instanzen gefunden werden kann.

Die *Companionsuche* erlaubt das Auffinden von PASTE Instanzen im Netz, welche bestimmte Companiondienste anbieten. Mit Hilfe verschiedener Metriken, kann aus einer Menge von gefundenen *Companions* ein geeigneter nach unterschiedlichen Kriterien herausgesucht werden. Das Auffinden (anderer) *Companions* im Netz geschieht initial über einen *Companion* Verzeichnisdienst. Dieser liefert Lokatoren von zuvor registrierten PASTE Instanzen, welche den gewünschten Dienst anbieten. Zusätzlich können im Verzeichnisdienst Informationen über den *Companion*, wie beispielsweise dessen Verfügbarkeit, abgelegt sein. Über den Transportmanager können gefundene *Companions* aber auch direkt kontaktiert werden, um weitere Informationen wie beispielsweise die RTT zwischen *Companion* und der eigenen PASTE Instanz zu bestimmen. Abhängig von der verwendeten Suchmetrik, werden die benötigten Parameter vom Verzeichnisdienst oder den gefundenen *Companions* abgefragt, um einen konkreten *Companion* auszuwählen.

Die *Companionsuche* verwaltet ebenfalls einen *Companioncache* in dem PASTE bereits bekannte oder zuvor genutzte *Companions* speichert. Durch die initiale Suche abgefragte Parameter werden ebenfalls mit im Cache abgelegt. Durch den Netzwerkinformationsmanager, können *Companions* und deren assoziierte Suchmetrik Parameter regelmäßig im Hintergrund auf Verfügbarkeit geprüft und aktualisiert werden. Dadurch kann bei zukünftigen Suchen im Idealfall zunächst der lokale Cache befragt werden, bevor der Verzeichnisdienst und einzelne *Companions* direkt kontaktiert werden müssen.

Der *Netzwerkinformationsmanager* sammelt über verschiedene Protokolle und Netzwerkmanagement Werkzeuge Informationen über das Netzwerk. Dazu werden auch wie bereits erwähnt Informationen über bekannte *Companions* in regelmäßigen Abständen erfasst. Des Weiteren werden Anfragen zur Verfügbarkeit bestimmter Ressourcen an andere bekannte PASTE Instanzen verschickt oder Informationen über die Latenz zwischen bekannten PASTE Instanzen ausgetauscht. Eine weitere Aufgabe ist das Überwachen aktiver Kommunikationsassoziationen, um zu überprüfen ob deren initiale Anforderungen noch erfüllt werden (können).

Analog dazu sammelt und verwaltet der *Ressourcenmanager* Informationen über den Zustand des lokalen Netzwerkknotens auf dem sich die PASTE Instanz selbst befindet. Dazu gehören Ressourcen wie beispielsweise verfügbarer Pufferspeicher oder die Auslastung der verfügbaren Rechenleistung.

Im Folgenden werden einzelne Komponenten und wie sie miteinander zusammenhängen und interagieren detailliert beschrieben. Neben einer allgemeinen Beschreibung der Komponenten wird dabei auch auf die PASTE-API eingegangen und inwiefern das PASTE Rahmenwerk diese umsetzt.

4.3 PASTE Module & Objekte

Bevor das PASTE Rahmenwerk von Anwendungen genutzt werden kann, müssen zunächst beim Start sämtliche Komponenten initialisiert werden. Dazu werden von jeder Komponente verwaltete Module beziehungsweise Objekte geladen und innerhalb von PASTE registriert. Namensauflösung, Companiondienstmanager, Companionsuche und -cache, Transportmanager und Transportselektion sowie Netzwerkinformationsmanager und der Ressourcenmanager verwalten alle unterschiedliche Module. Diese sind in Abbildung 4.3 den jeweiligen Komponenten zugeordnet. Diese unterstützen jeweils unterschiedliche Mechanismen zur Namensauflösung, enthalten Companiondienst spezifische Programmlogik und unterstützen verschiedene Suchmetriken. Der Transportmanager verwaltet ein Managementmodul pro unterstütztem Transportprotokoll und im Falle der Transportselektion, jeweils ein Protokoll spezifisches Selektionsmodul zur Abbildung von Transportdienstbeschreibungen auf die von der PASTE Instanz unterstützten Transportprotokolle. Netzwerkinformations- und Ressourcenmanager verwalten jeweils Module zum Sammeln Netzwerk spezifischer Kontextinformationen beziehungsweise spezifischer von einem Knoten bereitgestellter Ressourcen.

Verwaltete Objekte beinhalten Assoziationszustandsobjekte des Assoziationsmanagers und Einträge im Companioncache, sowie sämtliche PASTE-Handles. Im Netzwerkinformationsmanager werden die von verschiedenen Modulen gesammelten Informationen über lokale und entfernte Ressourcen und den Zustand aktuell genutzter Netzwerkpfade ebenfalls als Netzwerk Informationsobjekte bereitgestellt. Assoziationszustandsobjekte für alle offenen Kommunikationsassoziationen müssen beim Beenden einer PASTE Instanz persistent abgelegt und beim Start wieder geladen werden. Zustandsinformationen im Companioncache und Netzwerkinformationsmanager können hingegen in einem regelmäßigen Intervall gespeichert und der letzte bekannte Zustand beim Start von PASTE wiederhergestellt werden. Da diese Informationen regelmäßig erneuert werden, ist ihre Konsistenz über mehrere PASTE Sitzungen hinweg im Gegensatz zu den Assoziationszustandsobjekten von geringerer Priorität.

Der Transportmanager erzeugt Transport Endpunktobjekte für einzelne Assoziationen, welche sich über die Lebensdauer einer Assoziation und PASTE Sitzungen hinweg allerdings ändern können. Der Companiondienstmanager instanziiert schließlich Dienst-

objekte, welche Assoziations-spezifische Informationen für konkrete Companiondienste kapseln.

Unterschiedliche Module benötigen eindeutige Namen, mit denen sie alle bei dem Transportmanager angemeldet werden. Diese müssen vom Ersteller eines Moduls festgelegt werden, damit später über den Transportmanager eingehende GoodCompany (also Signalisierungs)Nachrichten anderer PASTE Instanzen intern an die richtigen Komponenten und Module weitergeleitet werden können, da die einzelnen Komponenten untereinander häufig Informationen austauschen müssen. In dem Transportmanager wird zudem ein Endpunkt für jedes unterstützte Transportprotokoll angelegt. In Abbildung 4.3 sind TCP und UDP als unterstützte Protokolle aufgeführt. Im Falle von TCP wird dementsprechend ein *TCP-ServerSocket* erzeugt und im Falle von UDP ein Socket geöffnet, der auf alle eingehenden IP Pakete für einen bestimmten *Port* lauscht. Im Folgenden *PASTE-Management-Port* genannt, handelt es sich dabei um einen beliebig gewählten jedoch für alle PASTE Instanzen einheitlichen *„well known“* Port.

4.4 Datenpfade im PASTE Rahmenwerk

Abbildung 4.4 zeigt sämtliche Datenpfade innerhalb des PASTE Rahmenwerkes und zwischen verschiedenen PASTE Instanzen, sowie zwischen PASTE Rahmenwerk und einzelnen Anwendungsinstanzen.

Zunächst existiert pro vom Transportmanager unterstütztem Protokoll ein entsprechender Endpunkt. In Abbildung 4.4 sowohl für TCP als auch UDP. Diese werden von den Protokoll-spezifischen Managermodulen verwaltet. Über den *PASTE-Management-Port* lassen sich zwischen diesen Endpunkten Verbindungen zu entfernten PASTE Instanzen aufbauen beziehungsweise direkt Nachrichten verschicken. Diese dienen nur der Übertragung von *GoodCompany* Nachrichten und somit dem Austausch von Signalisierungsnachrichten in Abwesenheit konkreter Kommunikationsassoziationen.

Transportverbindungen werden im Rahmen von Kommunikationsassoziationen automatisch von PASTE auf- und abgebaut. Dies wird mit Hilfe der Endpunkte innerhalb des Transportmanagers erreicht. Zunächst wird dazu ein Lokator mit dem Ziel der Transportverbindung benötigt. Dieser wird für die weiteren Ausführungen zunächst als gegeben angenommen. Man erhält ihn im Rahmen des Kommunikationsassoziationsaufbaus per Namensauflösung wie in Abschnitt 4.5.3 beschrieben.

Der konkrete Aufbau einer Transportverbindung hängt vom verwendeten Protokoll ab. Jedes Protokollmanager Modul des Transportmanagers ist einer Kommunikationsdomäne beziehungsweise einem Adressraum zugeordnet und für den Aufbau von Verbindungen über ein dediziertes Protokoll zuständig. Aktuell existiert als einzige Domäne der IPv4/6

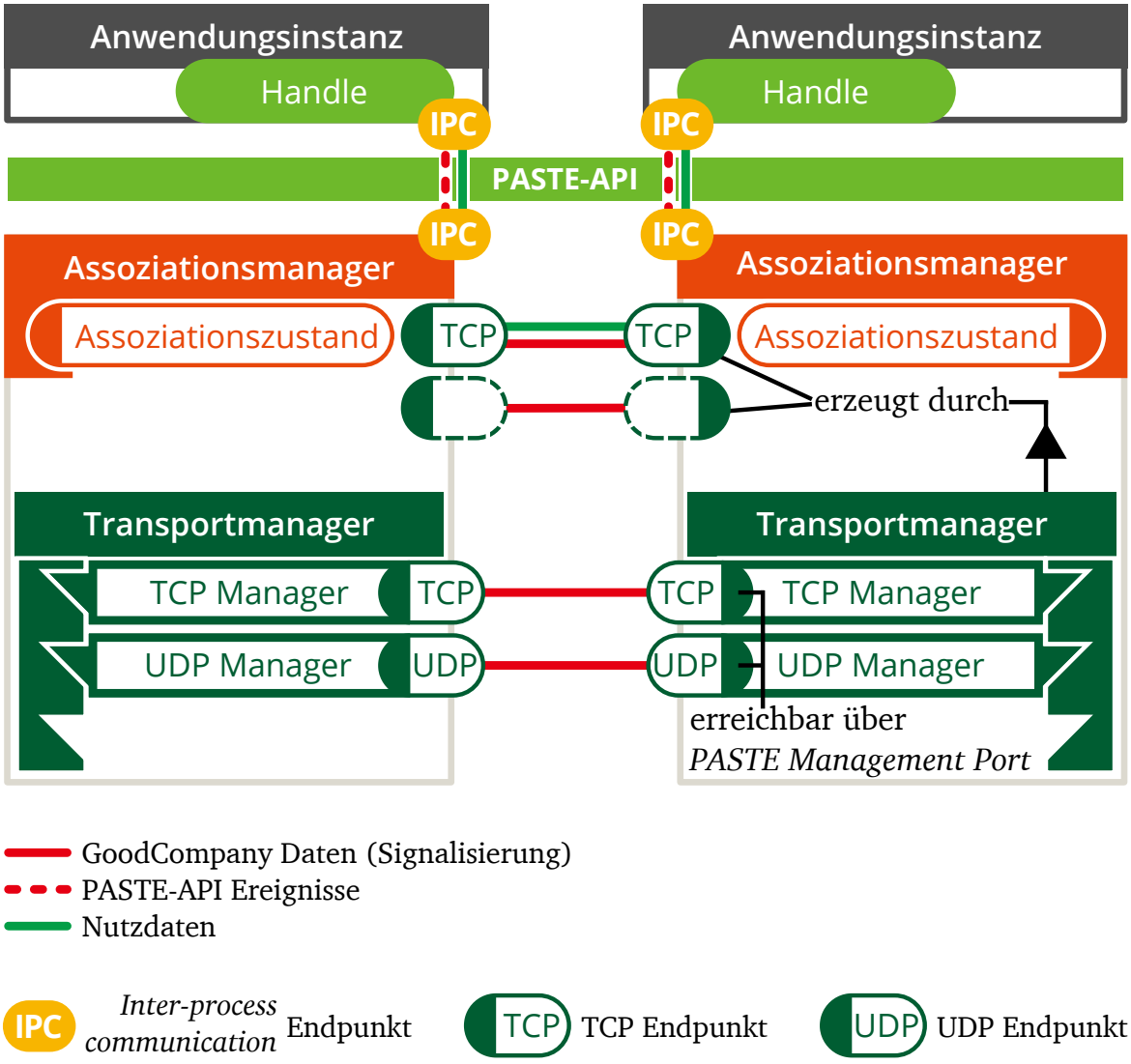


Abbildung 4.4: *Datenpfade und GoodCompany Signalisierung im PASTE Rahmenwerk.*

Adressraum für den unter anderem die Protokolle TCP, UDP, DCCP und SCTP von der IETF standardisiert sind.

Die in Abbildung 4.3 dargestellten TCP und UDP Module des Transportmanagers setzen demnach die Mechanismen zum Erzeugen neuer TCP Verbindungen beziehungsweise neuer UDP Endpunkte um.

TCP Verbindungen zu einer bekannten IP Adresse werden mittels `connect()` an den TCP Endpunkt des Transportmanagers der Gegenseite gerichtet. Als Ziel-Port wird dabei der *PASTE-Management-Port* verwendet, während als Quell-Port ein beliebiger freier Port gewählt wird. Die Gegenseite kann nun mittels `accept()` einen neuen dedizierten Socket erzeugen. Da dieser eindeutig durch das 5-Tupel aus [Protokoll: TCP, Quell-IP, Ziel-IP, Quell-

Port, PASTE-Management-Port (=Ziel-Port)] definiert ist, können mehrere Verbindungen zwischen den gleichen PASTE Instanzen aufgebaut werden, indem der Quell-Port für Anfragen an die gleiche IP-Adresse variiert wird. Hierzu merkt sich der Transportmanager für jede Transportverbindung pro Ziel-IP alle in Verwendung befindlichen Ports. Diese Information wird nur lokal benötigt und muss nicht mit der Gegenseite ausgetauscht werden.

Für UDP *Verbindungen* wird lokal ein zufälliger freier Port gewählt und ein UDP Socket für diesen Port erstellt. Von diesem wird an den UDP Endpunkt des Transportmanagers der Gegenseite eine Nachricht geschickt (ebenfalls an den *PASTE-Management-Port* als Ziel). Diese erzeugt daraufhin ihrerseits einen neuen UDP Socket für einen zufällig gewählten freien Port, über den eine Nachricht an den zuvor neu erzeugten UDP Socket der Gegenseite geschickt wird. Nun können Nachrichten über die jeweils neu erstellten UDP Sockets ausgetauscht werden, nachdem der initiale Handshake über den *PASTE-Management-Port* mit Hilfe der in den Transportmanagern verfügbaren UDP Endpunkte abgewickelt wurde.

In beiden Fällen erstellt PASTE beidseitig einen neuen Socket (entweder für TCP oder UDP), welcher an den Assoziationsmanager übergeben wird. Dieser ordnet ihn der entsprechenden Assoziation zu, für die eine Transportverbindung aufgebaut werden soll. Der Verbindungsabbau kann später durch den Assoziationsmanager erledigt werden, da der Transportmanager keinerlei Wissen über existierende Transportverbindungen benötigt, abgesehen von den Endpunkten für eingehende Verbindungsanfragen.

Auf diese Weise erzeugte Transportverbindungen dienen hauptsächlich, wie in Abbildung 4.4 dargestellt, dem Transport von Nutzdaten zwischen den an der zugehörigen Kommunikationsassoziation beteiligten Anwendungsinstanzen. Darüber hinaus können über die gleiche Transportverbindung ebenfalls *GoodCompany* Nachrichten ausgetauscht werden, welche zur Verwaltung und Aufrechterhaltung der Assoziation dienen. Gegebenenfalls kann für diese *GoodCompany* Nachrichten auch eine eigene (dedizierte) Transportverbindung für eine Assoziation erzeugt werden. In dem Fall werden *GoodCompany* Daten und Nutzdaten einer Assoziation über verschiedene Transportverbindungen übertragen. Dieser Fall tritt immer dann ein, wenn für Nutzdaten und *GoodCompany* Daten unterschiedliche Dienstgüteanforderungen benötigt werden.

Schließlich existieren noch über die PASTE-API aufgebaute lokale Inter-process communication (IPC) Kanäle zwischen PASTE Rahmenwerk und Anwendungsinstanzen. Über diese werden neben den Nutzdaten ebenfalls PASTE-API Ereignisse übertragen.

4.5 Assoziationsmanagement

Innerhalb des PASTE Rahmenwerks werden sämtliche andauernden Kommunikationsbeziehungen durch jeweils eine Kommunikationsassoziation gekapselt und im Assoziationsmanager verwaltet. Auf der Anwendungsebene erhalten Anwendungsinstanzen pro Kommunikationsassoziation ein PASTE-Handle ähnlich den klassischen *connection handles* der Socket-API. Jede PASTE Instanz die an einer Assoziation beteiligt ist, hält pro Assoziation ein Assoziationszustandsobjekt, welches alle relevante Zustandsinformationen einer Kommunikationsassoziation kapselt.

Des Weiteren ist der Assoziationsmanager zuständig für die Vergabe beziehungsweise das Generieren eindeutiger Identifikatoren für die Kommunikationsassoziationen. Jede Kommunikationsassoziation hat eine global eindeutige AssoziationsID. Zusätzlich existiert eine SitzungsID, welche für jede Anwendungsinstanz vergeben wird. Anwendungsinstanzen registrieren sich beim PASTE Rahmenwerk, bevor sie Befehle der PASTE-API aufrufen können. Initial wird dabei vom Assoziationsmanager eine lokal eindeutige SitzungsID vergeben. Unter dieser ID werden sämtliche zu einer Anwendungsinstanz gehörenden Assoziationen gruppiert. Anwendungsinstanzen können beim erneuten registrieren die SitzungsID übergeben und offene Assoziationen wiederaufnehmen. Alternativ können Anwendungsinstanzen sich auch die AssoziationsIDs selber merken und eine bestimmte Assoziation nach einem Neustart direkt wiederaufnehmen. SitzungsIDs werden dabei pro Anwendungsinstanz vergeben und sämtlichen Assoziationen der Anwendungsinstanz zugeordnet. Diese sind gültig, solange lokal Assoziationen für eine SitzungsID existieren. AssoziationsIDs sind so lange gültig, bis sämtliche Assoziationszustandsobjekte auf allen beteiligten Systemen gelöscht wurden. Im Folgenden wird erläutert, wie Assoziationen auf- und abgebaut, sowie pausiert und wiederaufgenommen werden können.

Kommunikationsassoziationen bestehen grundsätzlich zwischen zwei Endsystemen und können einen oder mehrere *Companions* während ihrer Lebensdauer verwenden, wobei zu einem konkreten Zeitpunkt immer nur ein *Companion* pro Datenflussrichtung auf einmal aktiv ist, über den sämtliche Daten geleitet werden. Sie werden von Anwendungsinstanzen mit Hilfe der PASTE-API erzeugt. Für jede Primitive der PASTE-API lässt sich eine anfragende Seite und eine die gestellte Anfrage beantwortende Seite identifizieren. Im Folgenden wird das anfragende Endsystem auch als *Client(seite)* bezeichnet. Das eine Anfrage beantwortende Endsystem wird entsprechend als *Server(seite)* bezeichnet. An einer Assoziation beteiligte Systeme können dabei sowohl die Client als auch die Server Rolle in Bezug auf verschiedene Anfragen annehmen. Die Beschränkung auf einen *Companion* pro Datenflussrichtung wurde dabei bewusst gewählt. Zunächst vereinfacht es die Signalisierung und die Zustandshaltung auf allen beteiligten System. Hinzu kommt, dass ein *Companion* mehr als einen Dienst auf einmal anbieten kann, so dass Dateneinheiten auch mehrere Companiondienste hintereinander durchlaufen

können. Da davon ausgegangen wird, dass es sich bei *Companions* um hochverfügbare Systeme im Netz handelt, ist es sinnvoller genutzte Dienste wenn möglich auf einem Knoten zu konzentrieren, statt zusätzliche Latenzen durch weitere *Hops* einzuführen.

Da es sich beim Companion ebenfalls um eine auf einem Zwischensystem ausgeführte Paste-Instanz handelt, kann ein Companiondienst zudem selbst auch weitere Kommunikationsassoziationen zu anderen Paste-Instanzen und somit Companions im Netzwerk aufbauen. Dadurch können Entwickler von Companiondiensten und somit auch Anwendungsentwickler indirekt komplexere Dienste im Netzwerk realisieren und nutzen, welche von mehreren Companions gemeinsam erbracht werden.

4.5.1 Anwendungsdienste anbieten

Bevor überhaupt eine Assoziation zwischen zwei Systemen aufgebaut werden kann, muss eine Anwendungsinstanz einen Anwendungsdienst anbieten und im Netzwerk unter einem Namen verfügbar machen, so dass sich eine andere Anwendungsinstanz zu diesem Anwendungsdienst verbinden kann. Abbildung 4.5 zeigt ein Weg-Zeit Diagramm dieses Vorgangs. Einen Dienst anbietende Endsysteme fungieren initial als Server-seitiger Teil einer Assoziation. Auf der linken Seite befindet sich das Endsystem *ES*, auf dem eine PASTE Instanz *P* läuft. Eine Anwendungsinstanz *A*, ebenfalls auf dem Endsystem *ES* befindlich, registriert sich bei einem Namensserver *N*.

Die Kommunikation der Instanzen untereinander findet, wie in Abbildung 4.4 dargestellt, über drei verschiedene Methoden statt:

- Anwendungsinstanzen kommunizieren mit der lokalen PASTE Instanz, indem sie Befehle der PASTE-API aufruft.

- PASTE Instanzen (auf verschiedenen Systemen) kommunizieren untereinander über das *GoodCompany* Protokoll. *GoodCompany* Nachrichten werden oberhalb der Transportschicht *in-band* über existierende Transportprotokoll Verbindungen einer Assoziation verschickt. Existiert für eine Assoziation gerade keine aktive Transportprotokoll Verbindung, werden *GoodCompany* Nachrichten *out-of-band* über den Transportmanager an andere PASTE Instanzen geschickt.

- PASTE Instanzen kommunizieren mit lokalen Anwendungsinstanzen über den Ereigniskanal der PASTE-API. Das Ergebnis eines von einer Anwendungsinstanz aufgerufenen Befehls, wird dieser über Ereignisse beziehungsweise Fehlermeldungen mitgeteilt. Anfragen entfernter Anwendungsinstanzen werden ebenfalls als Ereignis an lokale Anwendungsinstanzen weitergereicht, sofern sie nicht ausschließlich innerhalb des PASTE Frameworks bearbeitet werden können.

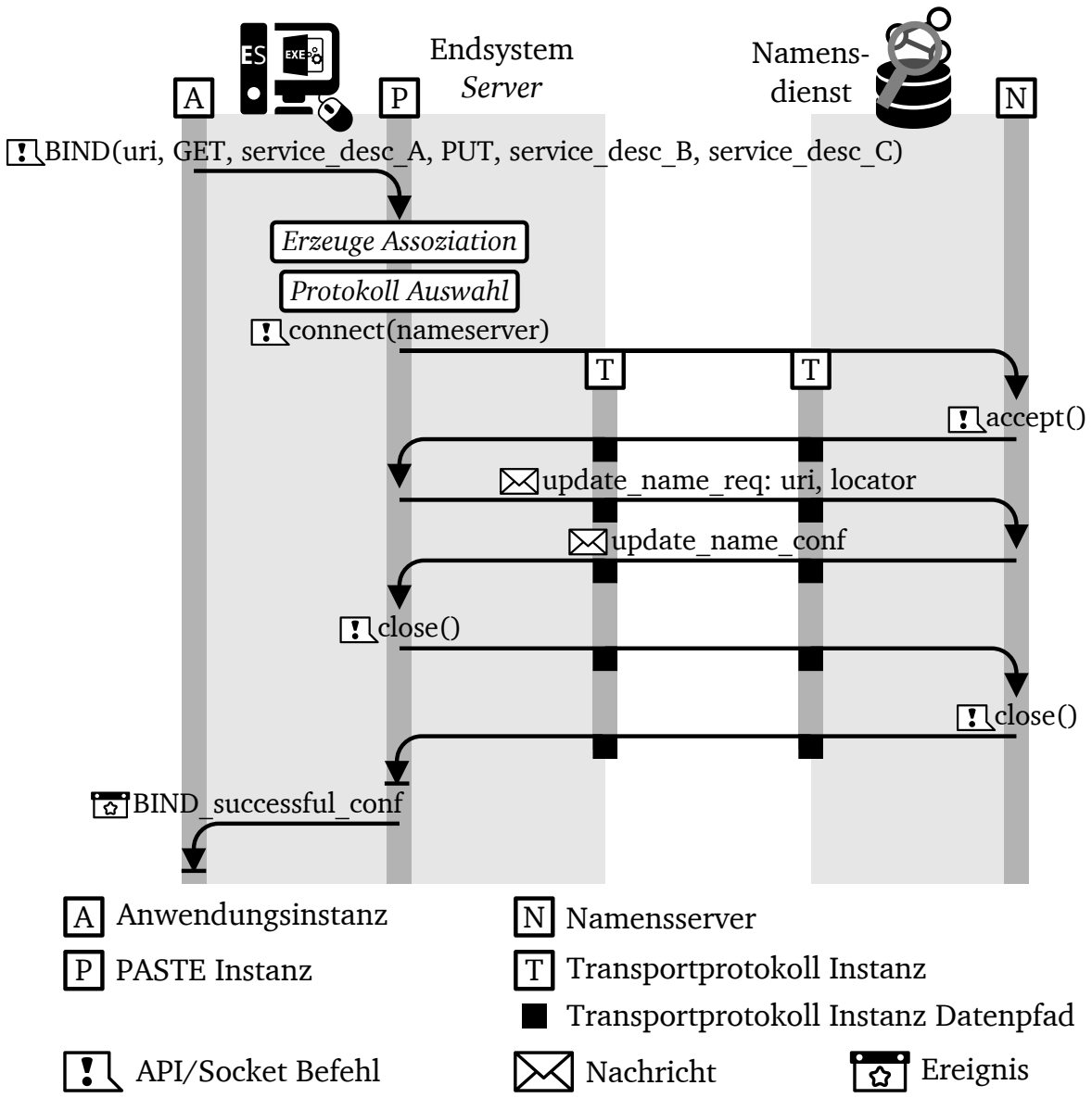


Abbildung 4.5: *Binden einer Anwendungsinstanz an eine URI.*

Damit eine Anwendungsinstanz erreichbar ist, muss diese sich an einen Namen beziehungsweise eine URI binden. Dazu fordert die Anwendungsinstanz *A* auf dem System *ES* mit Hilfe von `BIND(name, service_desc, primitives)` einen PASTE-Handle an. Sofern der Name lokal nicht bereits in Form von einer existierenden Assoziation bekannt ist, wird als erstes ein Assoziationszustandsobjekt angelegt. Dieses wird in Abschnitt 4.5.2 beschrieben. Danach werden durch die Transportselektion geeignete Protokolle zur Umsetzung der Dienstbeschreibungen gesucht. Anschließend wird wie in Abschnitt 4.4 beschrieben eine neue Transportverbindung zu einem autoritativen Namensverzeichnis *N*

aufgebaut. Die lokale PASTE Instanz registriert dort unter dem neuen Namen den Lokator über den das System auf dem die Anwendungsinstanz läuft aktuell erreichbar ist und schließt danach die Verbindung wieder.

Abhängig von den übergebenen Dienstbeschreibungen und den anhand der Dienstbeschreibungen gefundenen Protokollen, wird ein passender Namensdienst und Adressraum gewählt. Dies geschieht mit Hilfe des Transportmanagers, da alle Protokollmodule einem bestimmten Namens- beziehungsweise Adressraum zugeordnet sind. Kommen mehrere Adressräume in Frage, wird entweder ein bevorzugter ausgewählt oder der neue Name bei mehreren Namensdiensten registriert. Das gewünschte Verhalten kann vom Betreiber der PASTE Instanz *P* für diese festgelegt werden. Abschließend wird der Anwendungsinstanz *A* die erfolgreiche Registrierung ihres Anwendungsdienstes durch ein Ereignis signalisiert.

4.5.2 Assoziationszustandsobjekte

Abbildung 4.6 zeigt den zu einer Assoziation gehörigen Zustand. Der Assoziationsidentifikator wird von PASTE beim Erzeugen einer neuen Assoziation vergeben und ist global eindeutig. Er wird gebildet durch die Konkatenation des initialen Quell- sowie Zielnamens. Damit mehrere Verbindungen zwischen den gleichen Endsystemen eindeutige IDs erhalten wird zusätzlich noch ein eindeutiger Zeitstempel beim Erstellen des Assoziationszustandsobjektes angehängt. Über das Ergebnis der Konkatenation, welches eine eindeutige Zeichenfolge darstellt, wird dann ein 128bit Hash berechnet, welcher als Assoziationsidentifikator verwendet wird. Dadurch wird erreicht, dass sämtliche Assoziationsidentifikatoren die gleiche feste Länge aufweisen, wodurch diese sich unter Anderem leichter maschinell verarbeiten lassen und immer den gleichen Platz im Protokollheader des *GoodCompany* Protokolls benötigen. Dazu kann beispielsweise ein MD5 Hash verwendet werden. Bei per `BIND` Aufruf erzeugten Assoziationen wird der Zielname auf „unspezifiziert“ gesetzt.

Darauf folgt beziehungsweise folgen die von der Anwendungsinstanz übergebene(n) Transportdienstbeschreibung(en) sowie die unterstützen Dienstprimitive. Die Transportdienstbeschreibungen werden jedes Mal verwendet, wenn ein geeignetes Transportprotokoll zum Datentransport auszuwählen ist, um eine Transportverbindung zu einer anderen PASTE Instanz aufzubauen.

Es folgen der ursprüngliche Quell- sowie Zielname selbst. Diese müssen zusätzlich zum Assoziationsidentifikator gespeichert werden, da die Namen einen Anwendungsdienst spezifischen Teil am Ende enthalten können, welcher bei eingehenden Assoziationsanfragen an die Anwendungsinstanz weitergeleitet werden muss (siehe Kapitel 3.4.2).

Es folgen die Namen von allen *Companions*, die aktuell an der Transportdiensterbringung beteiligt sind und welcher Companiondienst verwendet wird. Je nachdem ob eine

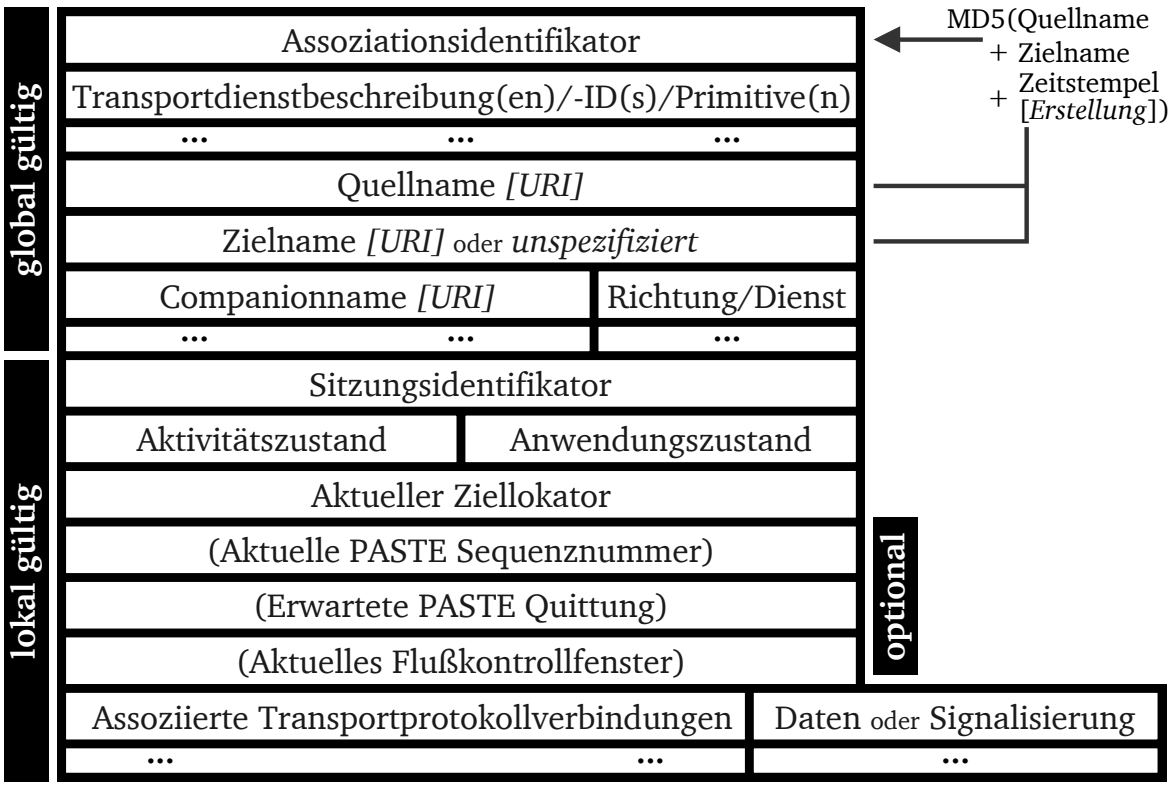


Abbildung 4.6: *Pro Assoziationsobjekt von PASTE verwalteter Zustand.*

Assoziation zum Bi- oder Unidirektionalen Transfer von Daten geöffnet wurde, können Daten in Richtung vom Client zum Server laufen oder umgekehrt. Abhängig vom gewählten Transportdienst durchlaufen Daten von der (initialen) Quelle zum (initialen) Ziel nicht den gleichen Pfad wie Daten vom Ziel zur Quelle und müssen auch nicht notwendigerweise identisch behandelt werden. Zusätzlich zum Companionnamen wird deswegen noch gespeichert, welche Daten über den *Companion* geleitet werden: *upstream* von der initialen Quelle zum initialen Ziel, oder *downstream* vom initialen Ziel zur initialen Quelle.

Dieser Teil des Assoziationszustandes ist global eindeutig, und muss zwischen den Endsystemen und allen *Companions* konsistent gehalten werden.

Die folgenden Felder des Assoziationszustandsobjektes beinhalten im Gegensatz dazu Daten, welche auf den beteiligten Systemen unterschiedliche Werte annehmen können.

Der Sitzungsidentifikator ordnet das Assoziationszustandsobjekt einer konkreten Anwendungsinstanz zu.

Der Aktivitätszustand gibt an, ob eine Assoziation aktuell aus der Sicht der lokalen PASTE Instanz aktiv ist oder zum Beispiel pausiert wurde. Der Aktivitätszustand wird in Abschnitt 4.8 detaillierter beschrieben.

Der Anwendungszustand speichert, ob die zugehörige Anwendungsinstanz gerade ausgeführt wird und erreichbar ist. Im Falle dass die Anwendungsinstanz beendet wird, muss die Assoziation von PASTE pausiert werden, sofern die Anwendungsinstanz nicht selbsttätig die Pausierung anstößt.

Der aktuelle Ziellokator bestimmt die Adresse, an die sämtliche Daten dieser Assoziation aus Sicht der lokalen PASTE Instanz aktuell gesendet werden. Initial handelt es sich dabei um die Adresse des Endsystems mit dem Ziel- beziehungsweise Quellnamen. Sofern für die Daten in Richtung von lokaler zu entfernter PASTE Instanz ein *Companion* verwendet wird, so ist der aktuelle Ziellokator die Adresse des *Companions*, welcher während der Lebensdauer einer Assoziation auch wechseln kann.

Optional folgen Felder für die aktuelle und erwartete Sequenznummer, sowie die Größe beziehungsweise Senderate vorhandener Fluss- oder Staukontrollfenster. Ob diese Parameter benötigt werden hängt vom gewünschten Transportdienst ab. Sofern das zur Diensterbringung eingesetzte Transportprotokoll Fluss- oder Staukontrolle verwendet, dienen die entsprechenden Felder der Synchronisation der Fenster über die Teilpfade bei Einsatz eines *Companions* hinweg; also den Pfad von Quelle zum *Companionen* und dem Pfad vom *Companion* zum Ziel.

Um die Übertragung sämtlicher Daten bei Verwendung eines zuverlässigen Dienstes sicherstellen zu können, werden die (protokollunabhängigen) PASTE Sequenznummern benötigt. Diesen dienen der Überprüfung, welches der an einer Assoziation beteiligten Systeme bereits ein bestimmtes Datum empfangen oder auch weitergeleitet hat. Auf die Felder wird in Abschnitt 4.7 näher eingegangen.

Zuletzt folgt eine Liste aller aktiven Transportprotokollverbindungen, welche zum Übertragen der Nutz- und Signalisierungsdaten der an der Assoziation beteiligten Systeme dienen. In der Regel handelt es sich dabei um eine Transportprotokollverbindung zum aktuellen Ziel; Also der ursprünglichen Quelle beziehungsweise dem ursprünglichen Ziel oder einem *Companion*. In verschiedenen Situationen kann eine Assoziation aber auch mehrere parallele Transportverbindungen offen haben:

- Wenn das Transportprotokoll gewechselt werden soll, da der gewünschte Transportdienst aufgrund von Änderungen im Netzwerk oder des Gerätekontextes von einem anderen Transportprotokoll besser erbracht werden kann.

- Beim Aufbau einer neuen Transportverbindung, wenn ein *Companion* in die Assoziation mit einbezogen wird.

- Wenn ein dedizierter Signalisierungskanal benötigt wird, der andere Dienstgüteanforderungen hat als der von der Anwendungsinstanz angeforderte Transportdienst.

4.5.3 Aufbau von Kommunikationsassoziationen

Abbildung 4.7 zeigt den Ablauf beim Aufbau einer Kommunikationsassoziation zwischen zwei Endsystemen *EC* (Client: Stößt Assoziationsaufbau an) und *ES* (Server: Beantwortet Assoziationsanfrage). Abgebildet sind die beiden Endsysteme (jeweils grau hinterlegt) und die auf den Endsystemen laufenden PASTE Instanzen *P* und Anwendungsinstanzen *A*.

Die Anwendungsinstanz *A* auf Endsystem *EC* fordert durch den Aufruf von `PUT(name, service_desc)` einen PASTE-Handle und eine Kommunikationsassoziation zu dem in Abschnitt 4.5.1 angebotenen Anwendungsdienst an.

Als erstes wird von der lokalen PASTE Instanz *P (EC)* ein Assoziationszustandsobjekt angelegt. Anschließend wird der per `PUT(..)` übergebene Name per Namensauflösung auf einen Lokator beziehungsweise eine Adresse abgebildet, unter der die entfernte PASTE Instanz und somit der gewünschte Anwendungsdienst verfügbar ist. Da eine vollständige Dienstbeschreibung übergeben wurde, müssen die Werte der einzelnen Parameter und optionale Parameter des gewünschten Dienstes zunächst ausgehandelt werden, um ein konkretes Protokoll wählen zu können. Daher wird per *GoodCompany* Protokoll zunächst ein `association_request` über den Transportmanager verschickt, da initial noch keine Transportprotokoll Verbindung aufgebaut werden kann. Dabei werden die verwendete Primitive, der (Quell)Name der anfordernden Instanz, der zum Anfordern der Assoziation verwendete Name, die gewünschte Dienstbeschreibung und der Zeitstempel, mit dem das lokale Assoziationszustandsobjekt erzeugt wurde, übergeben.

In der PASTE Instanz des Servers *P (ES)* findet bei Erhalt des `association_request` ein Abgleich der gewünschten Dienstbeschreibung mit den initial vom Dienstanbieter in Abschnitt 4.5.1 übergebenen Dienstbeschreibungen durch die Transportselektion statt. Dabei wird der gewünschte Dienst des Clients *EC* (*service_description*) gegebenenfalls so modifiziert, dass er erfüllbar ist (*satisfiable_service_description*). Die PASTE Instanz *P (ES)* legt nun ebenfalls ein Assoziationszustandsobjekt an und generiert ein `PUT_request` Ereignis, welches an die Anwendungsinstanz *A (ES)* gesendet wird. Mit diesem wird insbesondere der zum Anfordern von *A (EC)* verwendete Name übergeben, inklusive dem Anwendungsdienst spezifischen Teil am Ende des Namens. Parallel wird eine `association_ind` über den Transportmanager mit der aktualisierten Dienstbeschreibung an *P (EC)* zurückgeschickt, auf deren Basis PASTE Instanz *P (EC)* nun ein passendes Transportprotokoll wählen kann.

Sobald Anwendungsinstanz *A (ES)* die eingehende Assoziationsanfrage per `ACCEPT` angenommen hat, wird eine `association_accepted_res` an Paste Instanz *P (EC)* geschickt. Diese kann zunächst das Assoziationszustandsobjekt mit der neuen Dienstbeschreibung aktualisieren und anschließend über das gewählte Transportprotokoll mit Hilfe des Transportmanagers einen Datenkanal zur PASTE Instanz *P (ES)* aufbauen. Durch

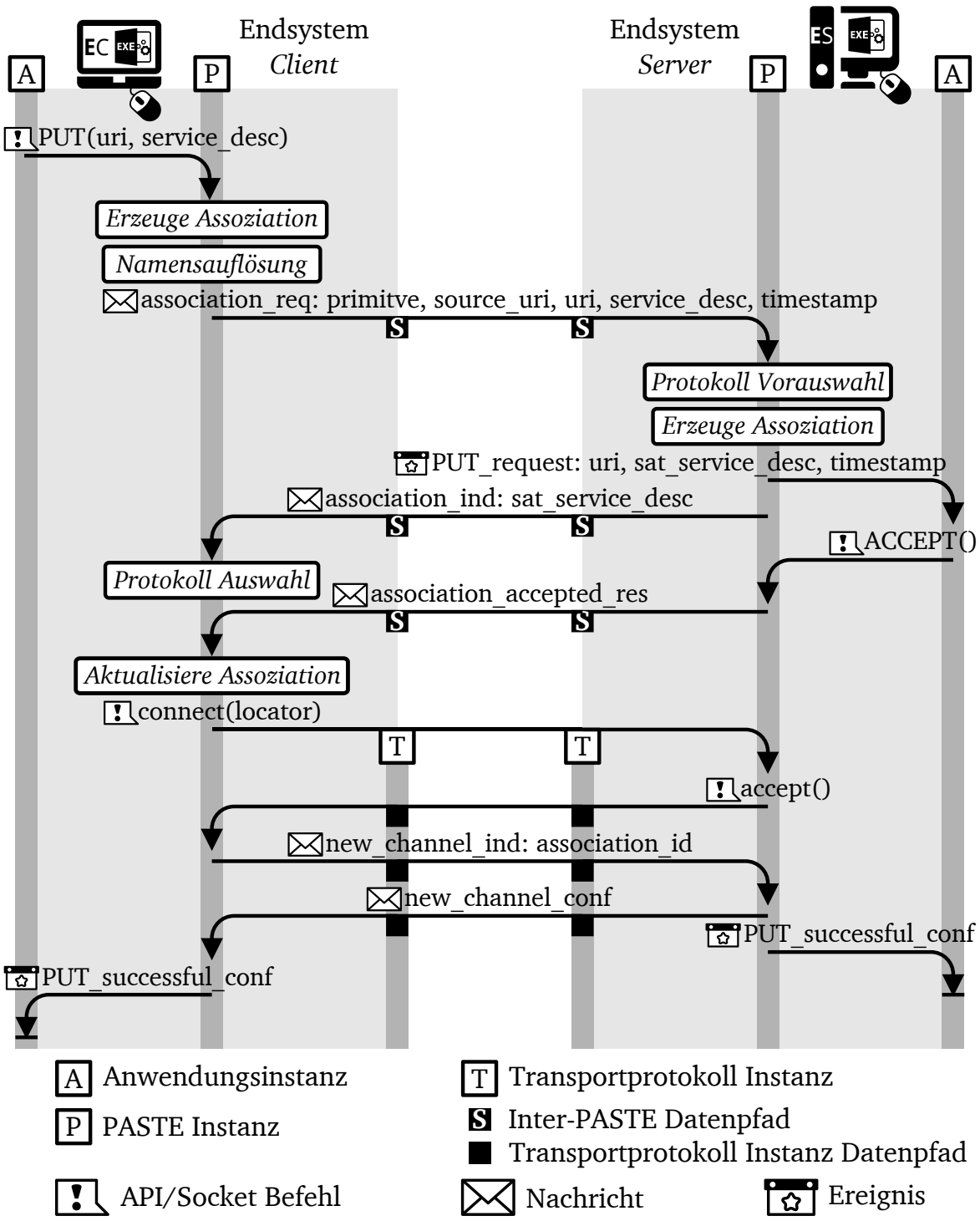


Abbildung 4.7: *Aufbau einer Kommunikationsassoziation zwischen zwei Endsystemen.*

den anschließenden Austausch einer `new_channel_ind` und einer `new_channel_conf` Nachricht, wird der PASTE Instanz *P (ES)* mitgeteilt, zu welcher Assoziation der neue

Datenkanal gehört. Nachdem dieser erfolgreich etabliert ist, generiert PASTE Instanz *P (EC)* ein `PUT_successful_conf` Ereignis, mit dem Anwendungsinstanz *A (EC)* die erfolgreiche Etablierung der ursprünglich angeforderten Assoziation mitgeteilt wird. Ebenfalls wird für die Anwendungsinstanz *A (ES)* ein `PUT_successful_conf` Ereignis erzeugt.

4.5.4 Verkürzter Aufbau von Kommunikationsassoziationen

Wie in Kapitel 3.5.3 beschrieben, existieren neben vollständigen Transportdienstbeschreibungen zwei weitere jeweils verkürzte Versionen. Häufig verwendete Kombinationen von Transportdiensteigenschaften können alternativ in Form von Dienstschablonen vorliegen, für die nur die konkreten Werte der einzelnen Transportdiensteigenschaften ausgefüllt werden müssen. In diesem Fall ändert sich am eigentlichen Ablauf für den Aufbau einer Kommunikationsassoziation nichts. Es müssen jedoch beim Senden der Dienstbeschreibung weniger Daten übertragen werden und die Protokollselektion vereinfacht sich, durch den Wegfall optionaler Transportdiensteigenschaften.

Im einfachsten Fall besteht die übergebene Transportdienstbeschreibung nur aus einer Transportdienst-ID beziehungsweise einem Transportdienstidentifikator statt einer vollständigen Beschreibung. Dadurch ergibt sich ein verkürzter Verbindungsaufbau, welcher in Abbildung 4.8 dargestellt ist. Die Ausgangssituation ist analog zu Abbildung 4.7.

Die Anwendungsinstanz *A* auf Endsysteme *EC* fordert durch den gleichen Aufruf von `PUT(name, service_ID)` einen PASTE-Handle und eine Kommunikationsassoziation an.

Als erstes wird auch hier von der lokalen PASTE Instanz *P (EC)* ein Assoziationszustandsobjekt angelegt und der per `PUT(..)` übergebene Name per Namensauflösung auf einen Lokator beziehungsweise eine Adresse abgebildet. Da mit einer Transportdienst-ID beidseitig ein fester Dienst mit vorgegebener Parametrisierung definiert ist, kann die Aushandlung selbiger entfallen.

PASTE Instanz *P (EC)* kann direkt ein passendes Transportprotokoll wählen. Parallel zum über den Transportmanager verschickten `association_request` kann PASTE Instanz *P (EC)* den Aufbau eines Datenkanals zur PASTE Instanz *P (ES)* anstoßen. Existieren mehrere Transportprotokolle, welche den gewünschten Transportdienst umsetzen können, wird von PASTE Instanz *P (EC)* per `connect(locator)` für jedes lokal verfügbare Transportprotokoll der Aufbau eines Datenkanals angestoßen. Die erste antwortende Instanz wird dann verwendet und die restlichen Transportprotokollinstanzen verworfen, ähnlich zum Happy Eyeballs Algorithmus [WY12].

In der PASTE Instanz des Servers *P (ES)* kann bei Erhalt des `association_request` ein Abgleich der gewünschten Dienstbeschreibung mit den initial vom Dienstanbieter übergebenen Dienstbeschreibungen entfallen und direkt ein Assoziationszustandsobjekt angelegt werden. Wie zuvor wird ein `PUT_request` Ereignis an die Anwendungsinstanz *A (ES)* gesendet. Sobald Anwendungsinstanz *A (ES)* die eingehende Assoziationsanfrage

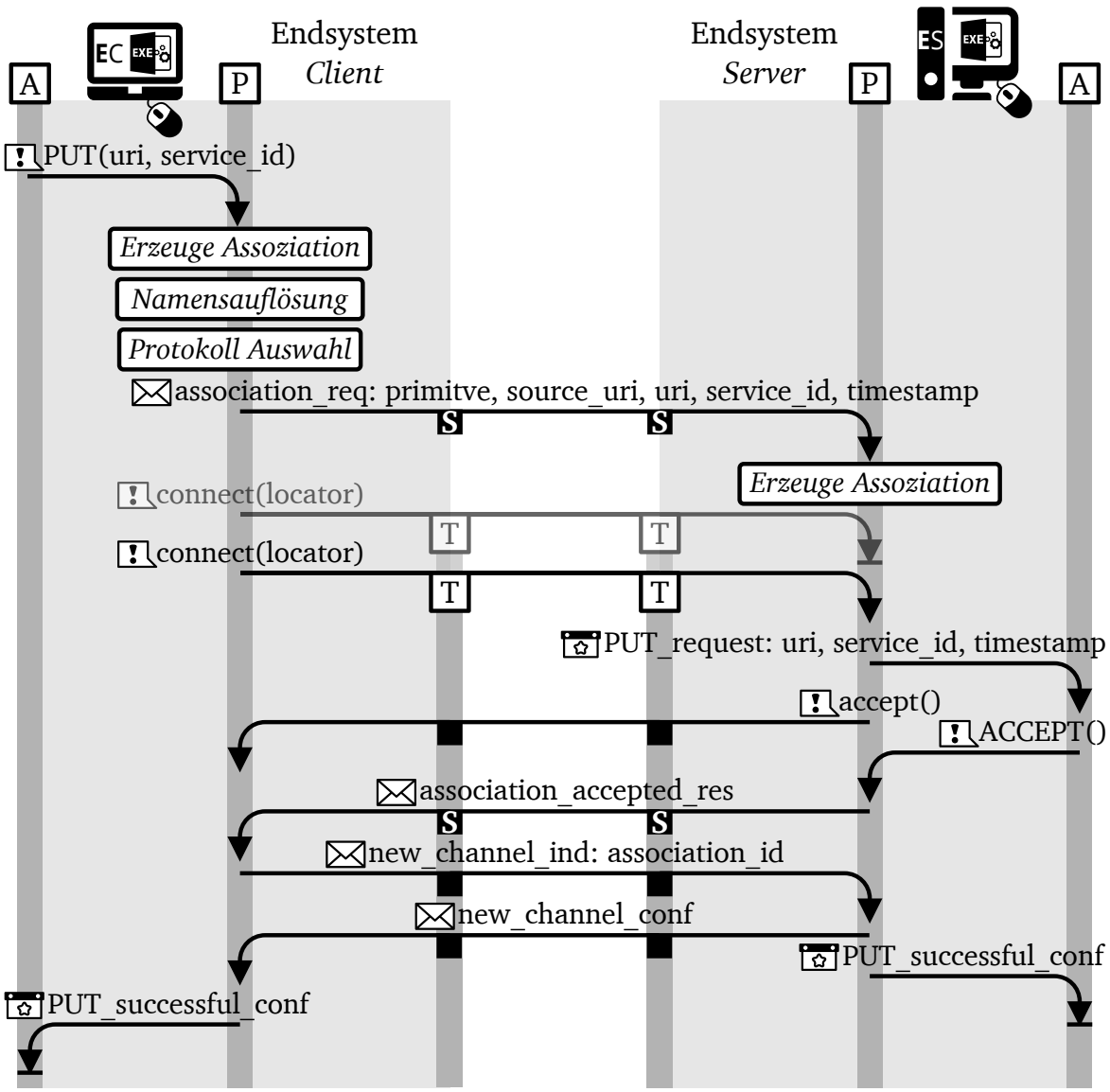


Abbildung 4.8: *Verkürzter Aufbau einer Kommunikationsassoziation zwischen zwei Endsystemen, mittels Transportdienstidentifikator.*

per `ACCEPT` angenommen hat, wird eine `association_accepted_res` an Paste Instanz *P (EC)* geschickt.

Der anschließende Austausch einer `new_channel_ind` und einer `new_channel_conf` Nachricht ist identisch zum vorherigen Assoziationsaufbau und ordnet dem bereits aufgebauten Datenkanal der entsprechenden Assoziation zu. Anschließend generiert PASTE Instanz *P (EC)* auch hier ein `PUT_successful_conf` Ereignis. Analog wird für Anwendungsinstanz *A (ES)* ein `PUT_successful_conf` Ereignis generiert.

4.5.5 Namensauflösung & Protokollselektion beim Kommunikationsassoziationsaufbau

Abbildung 4.9 zeigt den PASTE internen Ablauf beim Aufbau einer Kommunikationsassoziation und das Zusammenspiel der einzelnen Komponenten untereinander.

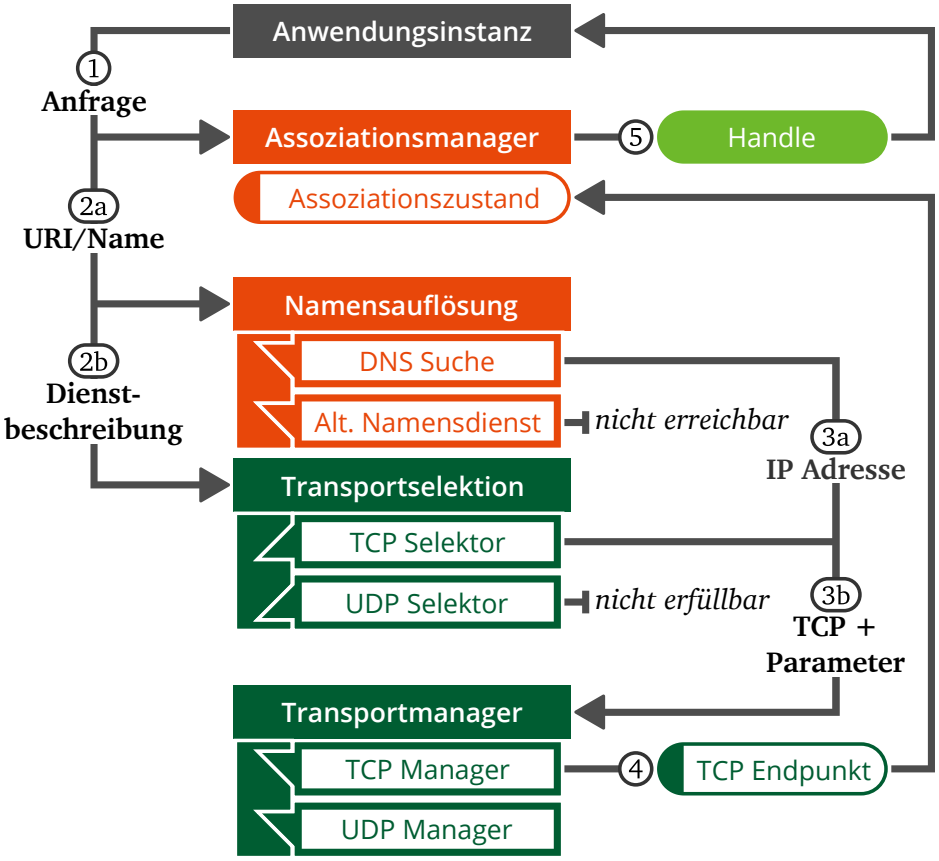


Abbildung 4.9: *Lokaler Ablauf des Kommunikationsassoziationsaufbaus.*

Von einer Anwendungsinstanz wird eine Assoziationsanfrage über die PASTE-API an die lokale PASTE Instanz übermittelt. Die Anfrage wird direkt an den Assoziationsmanager geleitet und dort ein Assoziationszustandsobjekt für die Anfrage erstellt *(1)*. Der übergebene (Ziel)Name und die übergebene Transportdienstbeschreibung werden parallel an die Namensauflösungskomponente und die Transportselektion weitergeleitet *(2a & 2b)*

Die Namensauflösung führt eine Suche für alle lokal verfügbaren Namensdienste durch. In diesem Fall wird per DNS eine IP Adresse gefunden *(3a)*. Gleichzeitig wird die Transportdienstbeschreibung in der Transportselektion für alle lokal bekannten Transportprotokolle auf Erfüllbarkeit überprüft. Die einzelnen Selektor Module prüfen dabei jeweils für ihr Protokoll, ob es in der Lage ist die Dienstbeschreibung umzusetzen. Als Ergebnis wird ein bestimmtes Protokoll mit einer passenden Parametrisierung zurückgeliefert:

In diesem Fall TCP *(3b)*, wobei beispielsweise abhängig von der Dienstbeschreibung die Verwendung der *TCP_NODELAY* Option (also der Verzicht auf die Verwendung des Nagle-Algorithmus) gewünscht sein kann.

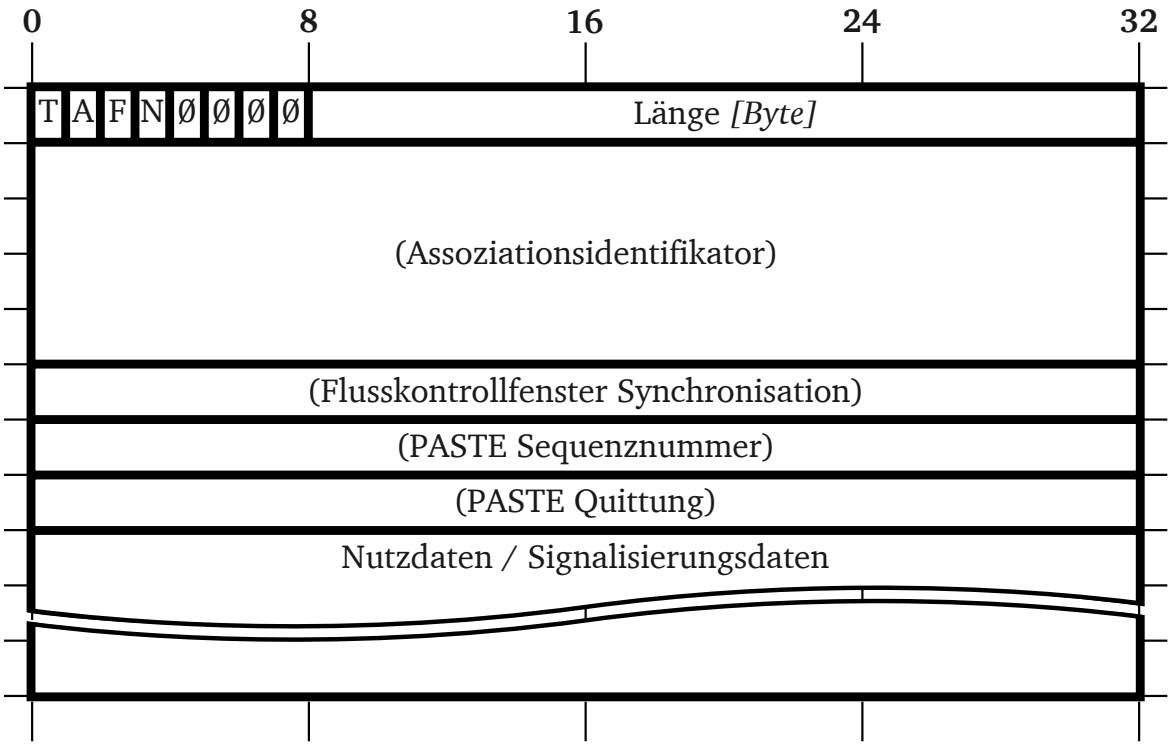
Der gefundene Lokator und das passende Protokoll werden an den Transportmanager übergeben. Dort kann nun über das passende Modul eine TCP Verbindung aufgebaut werden, welche direkt lokal der zugehörigen Assoziation zugeordnet wird *(4)*. Nach dem Austausch von `new_channel_ind`, `new_channel_conf` und `association_accepted_res` mit der Gegenseite, welche ebenfalls von dem Transportmanager empfangen werden, wird dem Assoziationsmanager der erfolgreiche Assoziationsaufbau auf der Gegenseite signalisiert und dieser kann an die Anwendungsinstanz ein für die Assoziation angelegtes PASTE-Handle übergeben *(5)*.

4.5.6 Der PASTE Rahmen und das GoodCompany Protokoll

Damit PASTE zwischen ausgetauschten Nutzdaten und Signalisierungsnachrichten zwischen an einer Assoziation beteiligten PASTE Instanzen unterscheiden kann, werden sämtliche übertragenen Daten mit einem PASTE spezifischem Rahmen versehen. Dieser transportiert sowohl Nutzdaten als auch Signalisierungsnachrichten des *GoodCompany* Protokolls.

ADUs (Application Data Units) werden von PASTE standardmäßig erhalten und Daten werden in der gleichen Granularität in der sie an PASTE übergeben wurden, beim Empfänger abgeliefert. Damit ist die Kommunikation grundsätzlich Paket-basiert ohne jedoch eine Annahme über die Größe individueller Pakete zu machen. Für jede Sendeoperation, welche auf einem Assoziationsidentifikator aufgerufen wurde, wird ein Rahmen an die von der Anwendungsinstanz übergebenen Nutzdaten angefügt. Je nach eingesetztem Transportprotokoll können diese Rahmen weiter fragmentiert oder modifiziert werden, zum Beispiel könnte ein *Companion* die Daten verlustfrei komprimieren um leistungsfähige Endgeräte mit schlechter Netzanbindung zu entlasten. In dem Fall werden die komprimierten und fragmentierten Daten von der PASTE Instanz des Zielsystems dekomprimiert und wieder zusammengesetzt, bevor sie an die Anwendungsinstanz gereicht werden. Auf dem Zielsystem werden die Rahmen entfernt und die Daten direkt an die Anwendungsinstanz übergeben. Somit können auch Companiondienste realisiert werden, welche die Nutzdaten manipulieren oder (teilweise) verwerfen können. Zudem können *Companions* welche Operationen auf den ADUs ausführen auch neue Rahmen erzeugen oder bestehende Rahmen modifizieren.

Der verwendete Rahmen ist in Abbildung 4.10 dargestellt. Er beginnt mit 4 *Flags*, welche den Nachrichtentyp und das Vorhandensein optionaler Felder anzeigen:



T = Typ.....[0: Nutzdaten / 1: Signalisierung]
A = Assoziationsidentifkator.....[0: fehlt / 1: vorhanden]
F = Flußkontrollfenster Synchronisation [0: fehlt / 1: vorhanden]
N = PASTE Sequenznummer.....[0: fehlt / 1: vorhanden]
Ø = nicht benutzt/reserviert
(...) = optionale Felder

Abbildung 4.10: *Aufbau des PASTE Rahmens für Nutz- beziehungsweise Signalisierungsdaten.*

- Das *T(yp)-Flag* bestimmt, ob sich im Datenteil der Nachricht Nutzdaten oder *GoodCompany* Signalisierungsinformationen befinden.

- Das *A(ssoziatonsID)-Flag* gibt an, ob der Assoziationsidentifikator im Rahmen enthalten ist oder nicht.

- Das *F(lusskontroll)-Flag* bestimmt, ob Informationen zur Größe des Flusskontrollfensters ausgetauscht werden.

- Das *N(ummern)-Flag* bestimmt, ob PASTE Sequenznummern verwendet werden, welche beispielsweise für zuverlässige Transportdienste benötigt werden.

Auf die *Flags* folgt ein Längenfeld, welches die Größe der *GoodCompany* Nachricht in Byte angibt. Darauf folgen die optionalen Felder und schließlich der Nutzdaten- beziehungsweise Signalisierungsdatenteil.

Wie in Abschnitt 4.4 erläutert wurde, können, abhängig vom zu erbringenden Dienst, zur Signalisierung benötigte Daten über existierende Transportverbindungen einer Assoziation *inband* mit den Nutzdaten übertragen werden. Je nach gewünschtem Transportdienst, kann auch ein dedizierter Datenkanal für Signalisierungsnachrichten für eine Assoziation aufgebaut werden. Letzterer Fall tritt beispielsweise ein, wenn die Dienstgüteanforderungen der Nutzdaten nicht denen der Signalisierungsdaten entsprechen, oder eine Assoziation aktuell keinen etablierten Datenkanal besitzt, da gerade keine Daten ausgetauscht werden. Zum Beispiel kann ein verlustbehafteter Transportdienst trotzdem eine zuverlässige Signalisierung benötigen.

Werden die Nachrichten *inband* oder über einen dedizierten Datenkanal übertragen, wird kein Assoziationsidentifikator benötigt. In dem Fall sind die jeweiligen Endpunkte der Transportverbindung auf den beteiligten PASTE Instanzen bereits einem Assoziationszustandsobjekt zugeordnet, anhand dessen eingehende Nachrichten den richtigen Komponenten beziehungsweise Anwendungsinstanzen zugeordnet werden können. Werden Signalisierungsnachrichten dagegen über den Transportmanager direkt übertragen, wird immer ein Assoziationsidentifikator mitgesendet, da Nachrichten für verschiedene Assoziationen über den Transportmanager empfangen werden können.

Das optionale Feld zur Flusskontrolle wird nur beim Einsatz eines *Companions* benötigt, sofern die zur Diensterbringung verwendeten Transportprotokolle eine Flusskontrolle bieten. Um in diesem Fall weiterhin eine Ende-zu-Ende Flusskontrolle zu ermöglichen und rechtzeitig auf dem gesamten Assoziationspfad auf Übertragungsengpässe reagieren zu können, müssen die Größen der Flusskontrollfenster angeglichen werden. Das heißt die Größe aller Flusskontrollfenster der Transportprotokollinstanzen, welche an einer Assoziation beteiligt sind. Dabei handelt es sich um die jeweils aktiven Transportprotokollinstanzen zwischen Quellsystem und eingesetztem Companion, zwischen eingesetztem Companion und Zielsystem, sowie zwischen Quell- und Zielsystem einer Assoziation.

Protokoll unabhängige Sequenznummern werden benötigt, wenn beispielsweise ein zuverlässiger Transportdienst beim Assoziationsaufbau angefordert wurde. Auch Eigenschaften wie Reihenfolgetreue benötigen Sequenznummern. Da Kommunikationsassoziationen über ihre Lebensdauer hinweg pausiert und wiederaufgenommen werden können, das eingesetzte Transportprotokoll wechseln kann und im Falle eines *Companions* wechselnde Systeme am Datentransport beteiligt sind, werden die PASTE Sequenznummern benötigt, um den vollständigen Erhalt und die Reihenfolge gewährleisten zu können und gegebenenfalls einzelne Dateneinheiten erneut anzufordern.

Zwar verwenden zuverlässige Transportprotokolle wie beispielsweise TCP bereits auf der Transportschicht Sequenznummern, diese sind beim Einsatz des PASTE Rahmenwerks

allerdings nicht ausreichend. Durch das Pausieren und Wiederaufnehmen von Kommunikationsassoziationen, sowie durch den Einsatz von *Companions*, können mehrere TCP Verbindungen oder auch Verbindungen über verschiedene Protokolle, über die Lebensdauer der Kommunikationsassoziation hinweg, verwendet werden. Daher werden eigene von den verwendeten Transportprotokollen unabhängige Sequenznummern benötigt, um beispielsweise einen zuverlässigen Transportdienst zwischen zwei oder mehr PASTE Instanzen zu realisieren.

4.5.7 Abbau von Kommunikationsassoziationen

Abbildung 4.11 zeigt den Ablauf beim Abbau einer Kommunikationsassoziation. Zu sehen sind die gleichen Endsysteme wie in Abbildung 4.7 & 4.8.

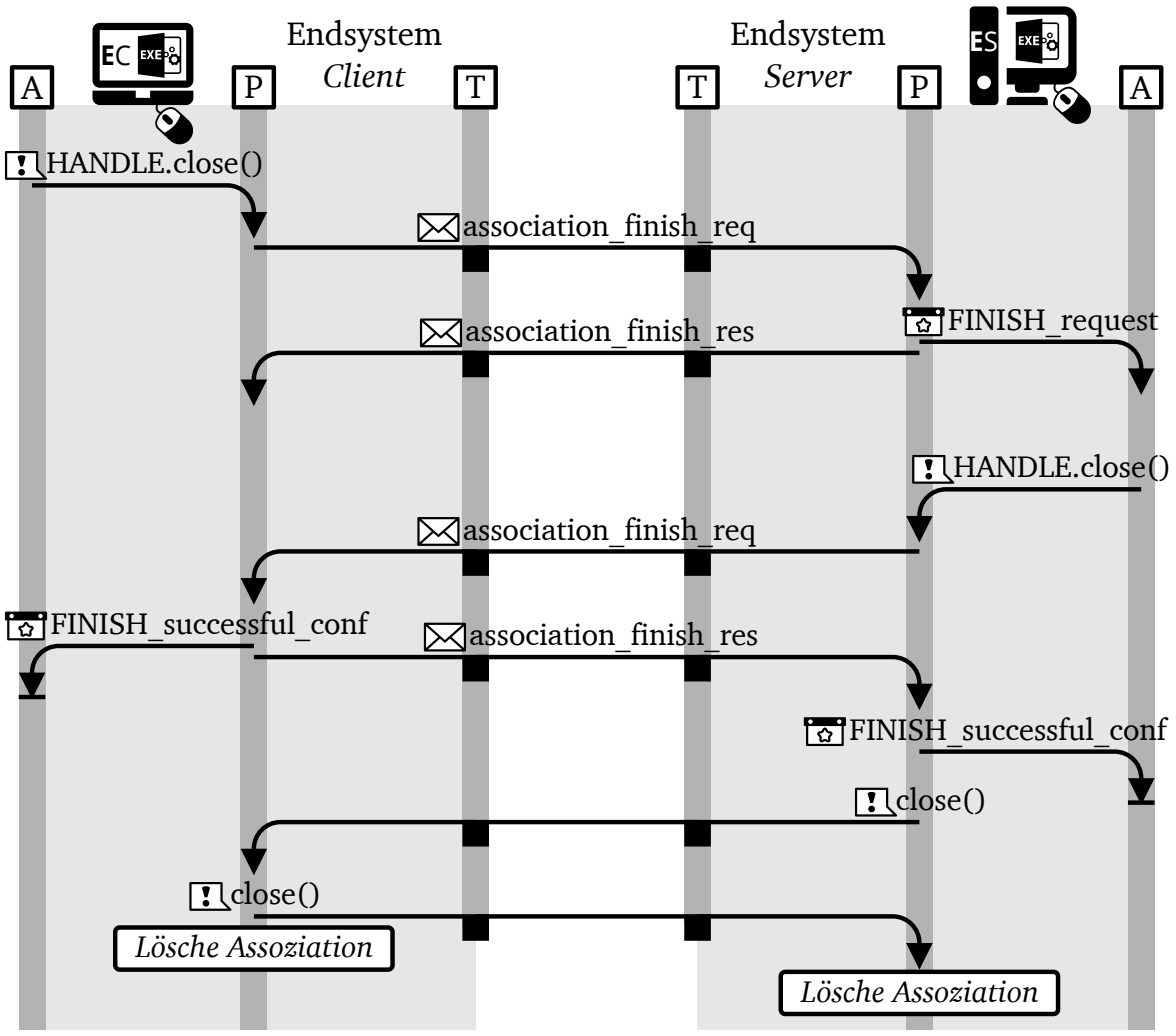


Abbildung 4.11: *Abbau einer Kommunikationsassoziation.*

Möchte eine Anwendungsinstanz eine Assoziation beenden, wird dies über den `close()` Befehl des PASTE-Handles initiiert. Hier beendet die Anwendungsinstanz *A (EC)* die Assoziation. PASTE Instanz *P (EC)* schickt dazu einen `association_finish_req` an PASTE Instanz *P (ES)*, welche ein `Finish_request` Ereignis an Anwendungsinstanz *A (ES)* auslöst und PASTE Instanz *P (EC)* diesen Vorgang mit einer `association_finish_res` Nachricht bestätigt. Anwendungsinstanz *A (EC)* darf nun keine Daten mehr senden und die PASTE Instanz *P (EC)* nimmt nach dem Absenden der `association_finish_req` Nachricht keine neuen Daten mehr von Anwendungsinstanz *A (EC)* entgegen. Damit muss die Gegenseite nach Erhalt der `association_finish_req` Nachricht keine neuen Daten mehr erwarten.

Hat die Anwendungsinstanz *A (ES)* ebenfalls keine Daten mehr zu senden, stößt sie den gleichen Nachrichtenaustausch an, in diesem Fall wird für Anwendungsinstanz *A (EC)* jedoch ein `Finish_successful_conf` Ereignis anstatt eines `Finish_request` Ereignisses generiert. Damit bekommt Anwendungsinstanz *A (EC)* bestätigt, dass auch die Gegenseite die Assoziation beendet hat und diese als geschlossen betrachtet werden kann.

Nach dem Austausch des zweiten `association_finish_req` und des zugehörigen `association_finish_res` zwischen PASTE Instanzen *P (ES)* und *P (EC)*, wird auch der Anwendungsinstanz *A (ES)* das erfolgreiche Schließen der Assoziation durch generieren eines `Finish_successful_conf` Ereignisses mitgeteilt. Anschließend kümmern sich die beiden PASTE Instanzen, für die Anwendungsinstanzen transparent, um den Abbau sämtlicher noch den Assoziationszustandsobjekten zugeordneten Transportverbindungen. Nachdem diese geschlossen sind, können die Assoziationszustandsobjekte auf beiden Systemen gelöscht werden.

4.5.8 Pausieren & Wiederaufnehmen von Kommunikationsassoziationen

Abbildung 4.12 zeigt den Ablauf beim Pausieren einer Kommunikationsassoziation.

Das Pausieren einer Assoziation kann von einer Anwendungsinstanz selbst aber auch durch das PASTE Rahmenwerk angestoßen werden. Der zweite Fall kann beispielsweise eintreten, wenn eine Anwendungsinstanz (temporär) beendet wird. In diesem Fall wird das Pausieren direkt vom PASTE Rahmenwerk angestoßen und pausierte Kommunikationsassoziationen später wieder aufgenommen, sobald die Anwendungsinstanzen erneut gestartet werden.

In Abbildung 4.12 wird das Pausieren der Assoziation durch die Anwendungsinstanz *A (EC)* eingeleitet, indem der `suspend()` Befehl des zugehörigen PASTE-Handles aufgerufen wird. PASTE Instanz *P (EC)* schickt dann einen `association_suspend_req` an PASTE Instanz *P (ES)*, welche ein `Suspend_request` Ereignis an Anwendungsinstanz *A (ES)* auslöst. Nachdem Anwendungsinstanz *A (ES)* den `Suspend_request` per `Accept()` bestätigt hat und PASTE Instanz *P (ES)* den lokalen Assoziationszustand aktualisiert

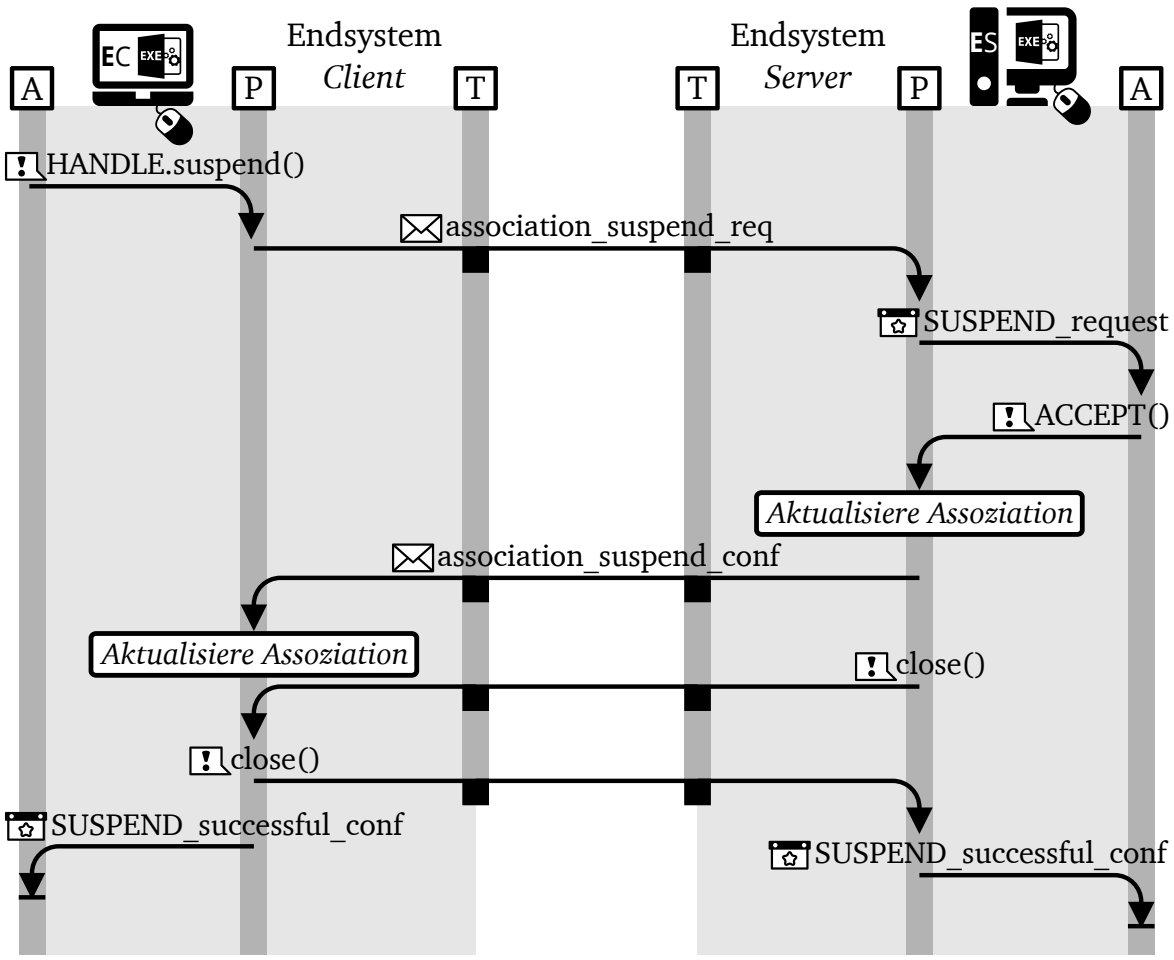


Abbildung 4.12: *Pausieren einer laufenden Kommunikationsassoziation.*

hat, wird PASTE Instanz *P (EC)* dies mit einer `association_suspend_conf` Nachricht bestätigt. PASTE Instanz *P (EC)* aktualisiert dann ebenfalls ihren Assoziationszustand.

Anschließend kümmern sich die beiden PASTE Instanzen wie beim Assoziationsabbau in Abschnitt 4.5.7, um den Abbau sämtlicher noch den Assoziationszustandsobjekten zugeordneten Transportverbindungen. Nachdem diese geschlossen sind, erhalten die Anwendungsinstanzen *A (EC)* & *A (ES)* jeweils ein `Suspend_successful_conf` Ereignis, um das erfolgreiche Pausieren der Assoziation zu bestätigen.

Abbildung 4.13 zeigt den Ablauf beim Wiederaufnehmen einer zuvor pausierten Kommunikationsassoziation.

Anwendungsinstanz *A (EC)* leitet die Wiederaufnahme der pausierten Assoziation durch die `Resume()` Primitive mit der gewünschten AssoziationsID ein. Dazu wird zunächst das zugehörige Assoziationszustandsobjekt aktualisiert. Eine Namensauflösung wird durchgeführt um den aktuellen Lokator der Gegenseite zu erhalten und die Protokollselektion

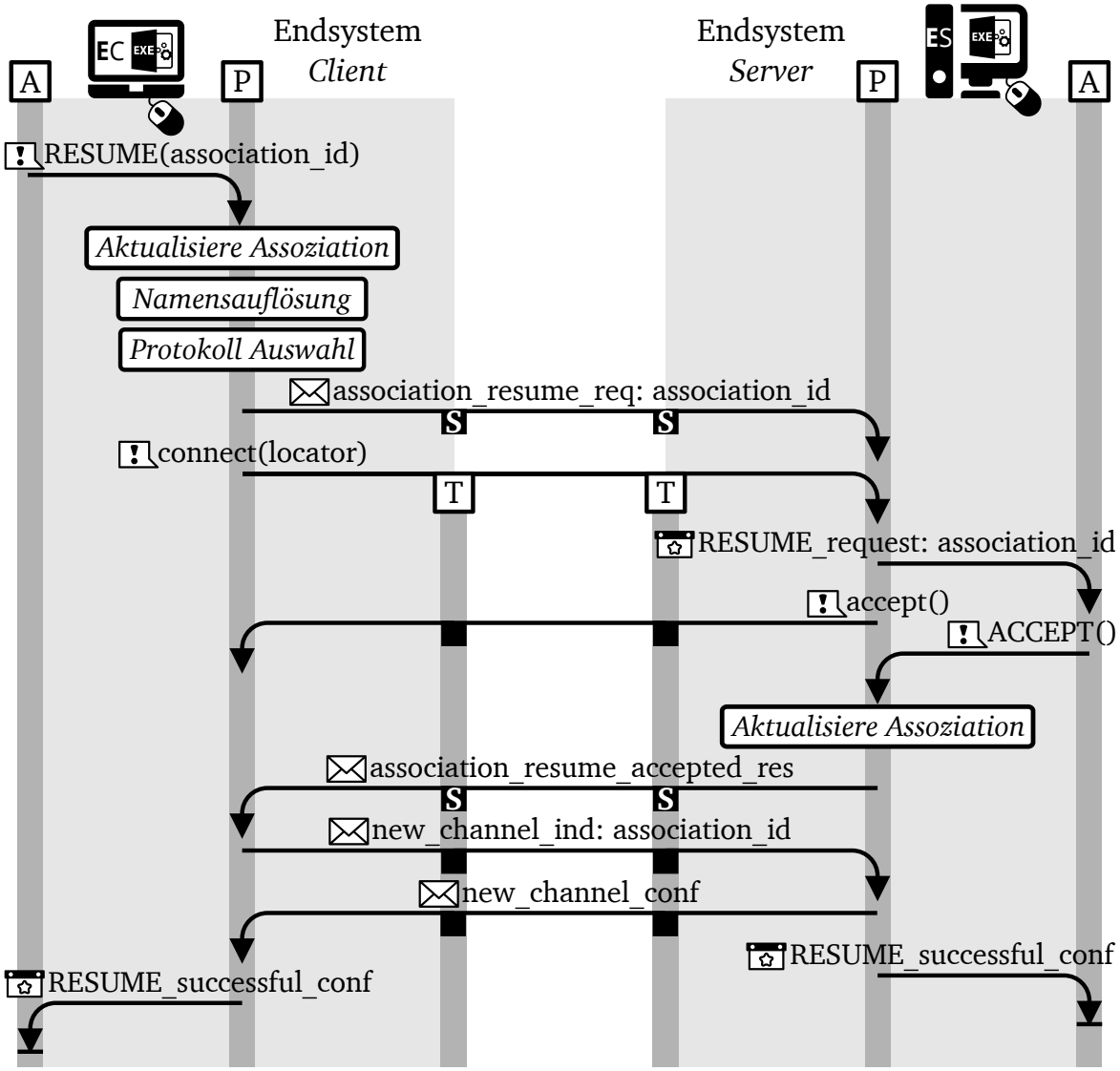


Abbildung 4.13: *Wiederaufnehmen einer pausierten Kommunikationsassoziation.*

wird erneut durchgeführt, da sich zwischenzeitlich der Kontext geändert haben kann, in dem sich beide PASTE Instanzen befinden. Eine erneute Aushandlung der optionalen Parameter ist zunächst nicht nötig, da die Assoziation mit dem zuvor ausgehandelten Dienst wiederaufgenommen werden soll. Der geänderte Kontext kann demnach zur Wahl eines anderen Protokolls für den gleichen zu erbringenden Transportdienst führen, ändert aber nichts am angeforderten Transportdienst selbst.

Mittels `association_resume_req` Nachricht, wird PASTE Instanz *P (ES)* über den Transportmanager signalisiert, welche Assoziation wiederaufgenommen werden soll. PASTE Instanz *P (ES)* sendet Anwendungsinstanz *A (ES)* daraufhin ein `Resume_request`

Ereignis, welche diese per `Accept()` Primitive annehmen kann. Dies wird der Gegenseite dann mit einer `association_resume_accepted_res` Nachricht bestätigt. Parallel dazu baut PASTE Instanz *P (EC)* über den Transportmanager bereits eine Transportverbindung mit dem gewünschten Protokoll zu PASTE Instanz *P (ES)* auf. Per `new_channel_ind` Nachricht wird PASTE Instanz *P (ES)* mitgeteilt, welcher Assoziation die Transportverbindung zuzuordnen ist. Sofern Anwendungsinstanz *A (ES)* das `Resume_request` Ereignis bestätigt hat, wird von PASTE Instanz *P (ES)* per `new_channel_conf` Nachricht die `new_channel_ind` und damit auch das initiale `Resume()` bestätigt. Daraufhin erhalten beide Anwendungsinstanzen ein `Resume_successful_conf` Ereignis, womit beide Anwendungsinstanzen wissen, dass die Assoziation erfolgreich wiederaufgenommen wurde und eine aktive Transportverbindung besteht.

4.6 Namensauflösung & Transportprotokollwahl

Um die Erweiterbarkeit von PASTE für zukünftige Protokolle, Netze und Adressschemata zu gewährleisten, sind sowohl die Namensauflösung als auch die Transportselektion modular aufgebaut. Da die Module autark arbeiten und nur ihre Ergebnisse beziehungsweise Rückgabewerte von der Namensauflösung und Transportselektion verwendet werden, lassen sich einfach neue Module hinzufügen.

Pro unterstütztem Adressschema enthält die Namensauflösung ein Modul, welches einen Namensdienst anbietet. Für das heutige Internet und den IPv4/IPv6 Adressraum beispielsweise ein Modul, welches das Auflösen von URIs auf IP-Adressen ermöglicht. Neben der Namensauflösung ist das Modul auch beim Anbieten von Anwendungsdiensten durch eine Anwendungsinstanz verantwortlich für die Registrierung neuer Namen.

Erlaubt ein verwendeter Namensdienst das Registrieren neuer URIs durch beliebige Systeme, kann dies beim Anbieten eines Anwendungsdienstes durch das PASTE Rahmenwerk durchgeführt werden. Gerade im heutigen Internet existiert ein solcher Dienst aber nicht. Netzwerknoten verfügen heute über einen oder mehrere Lokatoren in Form von IPv4 oder IPv6 Adressen, welche in der Regel vom Internetprovider vergeben werden. Eine Möglichkeit PASTE mit dem heutigen DNS System zu nutzen, besteht darin dem PASTE Rahmenwerk vorab registrierte URI zu übergeben. In dem Fall kann eine PASTE Instanz sich unter dieser URI mit ihrer aktuellen IP Adresse an den PASTE ManagementPort binden. Von Anwendungsinstanzen angeforderte Assoziationen können in dem Fall als URI beliebige Subdomänen der vergebenen URI wählen, welche von PASTE verwaltet werden.

Die Transportselektion hat ein Modul pro unterstütztem Protokoll und Namensraum. Nicht alle Protokolle sind zwingend in allen Namensräumen verfügbar. Dabei kommen jedem Modul zwei Aufgaben zu: Zunächst muss es für sein Protokoll eine Transport-

dienstbeschreibung auf Erfüllbarkeit hin überprüfen. Für eine übergebene Transportdienstbeschreibung wird also entweder eine geeignete Parametrisierung des Protokolls zurückgeliefert oder kein Ergebnis, sofern das Protokoll ungeeignet ist. Letzterer Fall tritt ein, sofern ein Protokoll nicht mindestens sämtliche verpflichtenden Transportdiensteigenschaften der übergebenen Transportdienstbeschreibung durch den vom Protokoll bereitgestellten Dienst erbringen kann. Die zweite Aufgabe ist das Überprüfen der Verfügbarkeit Protokoll-spezifischer Eigenschaften und Mechanismen auf einem bestimmten Netzwerkpfad.

Für einen übergebenen Namen und eine übergebene Transportdienstbeschreibung, stößt die Namensauflösung zunächst alle Namensdienst Module an. Diese können einen oder mehrere Lokatoren aus unterschiedlichen Adressbereichen zurückliefern. Der Transportselektion werden sowohl die Lokatoren als auch die Transportdienstbeschreibung übergeben. Die einzelnen Selektor Module prüfen daraufhin nicht nur die Dienstbeschreibung, sondern auch ob ein bestimmtes Protokoll für einen der gefundenen Adressbereiche verfügbar ist. Nur wenn dies der Fall ist, wird ein Ergebnis zurückgeliefert.

Besitzt ein Protokoll Parameter die mit der Gegenseite ausgehandelt werden können, beispielsweise die Unterstützung verschiedener Staukontrollverfahren, so prüft die Transportselektion mit Hilfe des Transportmanagers deren Verfügbarkeit für ein konkretes Ziel. Dazu werden sämtliche PASTE Instanzen der über die Namensauflösung zurückgelieferten Lokatoren abgefragt. Über den PASTE Management Port können Module einer PASTE Instanz grundsätzlich mit den Modulen einer anderen PASTE Instanz kommunizieren.

Im einfachsten Fall besteht die übergebene Transportdienstbeschreibung nur aus einer Transportdienst-ID. Anhand der Transportdienst-ID kann in einer vordefinierten Tabelle der Transportselektion ein entsprechendes Protokoll, welches den gewünschten Dienst bereitstellt gewählt werden. Für eine Transportdienst-ID sind sämtliche Parameter fest definiert, da sie in Form einer einheitlichen Transportdienstbeschreibung auf den Endsystemen verfügbar ist und diese entweder vollständig oder gar nicht unterstützt wird. Dadurch entfällt in dem Fall die Aushandlung und das Überprüfen der Verfügbarkeit einzelner Protokollmechanismen. Für individuell auf eine Anwendung zugeschnittene Dienste, müssen diese also vollständige Transportdienstbeschreibungen statt Transportdienst-IDs verwenden.

Als Endergebnis produziert die Transportselektion eine Menge von möglichen Protokollen und eine Menge von möglichen Lokatoren, für die jeweils mindestens eines der zurückgelieferten Protokolle verfügbar ist. Im einfachsten Fall erhält man nur eine Lokator/Protokoll Kombination. Ein zuverlässiger Datentransportdienst könnte aber zum Beispiel von TCP, SCTP oder auch MPTCP erbracht werden. Stehen mehrere gleichwertige Protokolle für einen angeforderten Dienst zur Verfügung, wird der Aufbau der Kommunikationsassoziation parallel über alle Verfügbaren Protokolle angestoßen. Als gleichwertig werden dabei sämtliche Protokolle betrachtet, deren bereitgestellter Dienst

einen identischen Satz an Transportdiensteigenschaften der übergebenen Transportdienstbeschreibung erfüllen kann. Sofern keine zusätzlichen Präferenzen oder systemweite Regeln konfiguriert sind, wird dann die zuerst erfolgreich etablierte Protokollinstanz verwendet und die restlichen verworfen. Die Wahl kann aber auch hierarchisch erfolgen: Auf Geräten mit mehreren Netzwerkschnittstellen kann zum Beispiel grundsätzlich zunächst MPTCP der Vorzug vor TCP gegeben werden. Genauso kann Domänen-spezifisch zum Beispiel vom Internet Provider ein bevorzugtes Protokoll vorgegeben sein. Befinden sich die an einer Kommunikationsassoziation beteiligten PASTE Instanzen in unterschiedlichen Domänen mit unterschiedlichen bevorzugten Protokollen, wird erneut der Aufbau der Kommunikationsassoziation parallel für die bevorzugten Protokolle angestoßen und dann die zuerst erfolgreich etablierte Protokollinstanz verwendet.

Sowohl die Namensauflösung als auch die Transportselektion erlaubt das Spezifizieren bevorzugter Protokolle und Adressschemata. Diese müssen für eine lokale PASTE Instanz konfiguriert werden und können von verschiedenen Faktoren abhängen. Sowohl der Nutzer als auch ein Systemadministrator oder das Betriebssystem können bestimmte Vorgaben haben. Ist keine Präferenz festgelegt, kann über den Transportmanager in zufälliger Reihenfolge oder auch parallel der Aufbau eines Datenkanals für sämtliche Lokator/Protokoll Kombination angestoßen werden. PASTE wählt dann das erste Protokoll, über das erfolgreich eine Verbindung aufgebaut werden konnte.

4.7 Companion Einsatz

Für eine bestehende Kommunikationsassoziation kann beidseitig *Companion* Unterstützung von einer der beiden involvierten PASTE Instanzen auf den Endsystemen angefordert werden. Es können verschiedene *Companion* Dienste existieren, welche anhand eines eindeutigen Companiondienst Namens identifiziert werden können. Diese werden durch PASTE Instanzen auf Zwischensystemen im Netz bereitgestellt, welche sich zuvor mit allen unterstützten *Companion* Diensten beim *Companion* Verzeichnis angemeldet haben. Im Folgenden werden beispielhaft ein Datenkompressionsdienst und ein Netzwerkzwischenspeicherdienst verwendet. Anhand eines detaillierten Beispiels wird zudem die *Companion* Nutzung in Abschnitt 4.9 beschrieben.

Ein *Companion* kann aber beispielsweise auch eingesetzt werden, um ein Ende zu Ende nicht verfügbares Protokoll zumindest auf einem Teilabschnitt der Kommunikationsstrecke zu verwenden. So können in einem Heimnetz Geräte mit verschiedenen Netzen wie etwa *Ethernet*, *Powerline* oder *Wireless* angebunden sein, was den Einsatz von MPTCP reizvoll macht. Da dies im Internet aber noch nicht weit verbreitet ist, kann eine MPTCP Gegenstelle in Form eines *Companions* auf einem Heimnetz-internen Router dazu beitragen, mehrere Netzwerkinterfaces eines Gerätes parallel zu nutzen und

gleichzeitig mit nicht MPTCP fähigen Systemen zu kommunizieren. In diesem Fall würde ein *Companion* erhaltene Pakete einfach nur weiterleiten ohne diese oder die in ihnen transportierten Daten in irgendeiner Form zu bearbeiten. In der Regel findet jedoch auch eine Verarbeitung der Pakete beziehungsweise Daten statt.

Damit es zum Einsatz eines *Companions* kommt, muss zunächst eine der an einer Assoziation beteiligten PASTE Instanzen die Nutzung eines bestimmten Companiondienstes anfordern. Dies kann verschiedene Auslöser haben:

(1) Fehlersituationen der einer Assoziation zugeordneten Transportprotokolle, wie zum Beispiel die Signalisierung einer Überlastsituation durch ECN.

(2) Informationen zur Ressourcennutzung vom Betriebssystem. Zum Beispiel eine nur geringe Auslastung eines Netzwerkinterfaces oder ein Engpass bei der Verarbeitung eingehender Daten.

(3) Ereignisse im Netzwerk selbst. So kann Anhand von einer kontinuierlich sinkenden Empfangsstärke eines drahtlosen Mediums beispielsweise rechtzeitig ein Ereignis generiert werden, wenn das Verlassen des Empfangsbereichs unmittelbar bevorsteht.

(4) Das Verhalten einer Anwendung oder des Nutzers. Eine Anwendungsinstanz selbst kann temporär ausfallen beziehungsweise nicht verfügbar sein, zum Beispiel angestoßen durch den Benutzer der eine Anwendung auf einem mobilen Gerät beendet beziehungsweise in den Hintergrund befördert. Oder vom Betriebssystem selbst angestoßen, um zum Beispiel Strom zu sparen und besonders rechenintensive Anwendungen bei geringem Akkustand zu pausieren.

Die PASTE Instanz eines MPTCP fähigen Endgerätes, welches mit einem Endgerät kommuniziert das kein MPTCP unterstützt, könnte beispielsweise über den Ressourcenmanager feststellen, dass nicht alle Netzwerkinterfaces benutzt werden beziehungsweise ausgelastet sind. Daraufhin könnte ein entsprechender *Companion* gesucht werden. Dies macht beispielsweise bei mobilen Endgeräten Sinn, die häufig drahtlos und mit niedrigerer Datentransferrate mit dem Netzwerk verbunden sind als (stationäre) drahtgebundene Systeme. Durch die Nutzung weitere Netzwerkinterfaces könnte so die Datentransferrate der Kommunikationsassoziation gesteigert werden.

Abhängig vom Auslöser kann anschließend ein passender *Companion* Dienst gewählt und gesucht werden. Dieser kann auch Teil einer Transportdienstbeschreibung sein. Je nach verwendetem Transportdienst können bestimmte Companiondienste zur Diensterbringung möglicherweise notwendig sein, während andere Transportdienste die Nutzung bestimmter Companiondienste auch explizit verbieten können. So haben Anwendungsinstanzen bereits bei der Wahl ihres bevorzugten Transportdienstes die Kontrolle ob und

welche Companiondienste zum Einsatz kommen dürfen. Da Companiondienste einer Kommunikationsassoziation neue Diensteigenschaften hinzufügen können, können diese als Teil der verpflichtenden Diensteigenschaften der Transportdienstbeschreibung aufgeführt werden. Wird beispielsweise die Transcodierung übergebener Nutzdaten gewünscht, kann eine entsprechende Diensteigenschaft eingefügt werden. Beim Aufbau der Assoziation wird dann direkt ein *Companion* gesucht, welcher die geforderte Transcodierung unterstützt. Ist für einen Transportdienst die Unterstützung durch Companiondienste grundsätzlich vorgesehen, passiert die tatsächliche Nutzung von *Companions* und den von ihnen angebotenen Diensten für die Anwendung transparent innerhalb des PASTE Rahmenwerks.

Da beim Einsatz eines *Companions* Transportprotokollverbindungen zwischen den beteiligten Systemen abgebaut und zwischen *Companion* und den Endsystemen aufgebaut werden müssen, kann es vorkommen dass im Transit befindliche Daten beim Wechsel, doppelt oder mit vertauschter Reihenfolge ankommen. Da je nach Companiondienst einzelne an einer Assoziation beteiligte Systeme temporär nicht verfügbar sein können, können Daten auch verloren gehen. Daher wird beim Einsatz eines *Companions* die PASTE Sequenznummer des *GoodCompany* Protokolls benötigt. Sobald die Anwendung reihenfolgetreue oder einen fehlerfreien Dienst angefordert hat, ist die PASTE Sequenznummer nötig, um zwischen Sender und Empfänger das erfolgreiche Ende einer Datenübertragung feststellen zu können, ohne auf beteiligte *Companions* zurückgreifen zu müssen, da der Empfänger grundsätzlich noch ausstehende Daten des Senders von dritten Knoten empfangen kann und diese vor dem Schließen einer Assoziation dem Sender bestätigen muss. Sonst lässt sich nicht feststellen, ob im Netz noch Daten für den Empfänger unterwegs sind. Auf die PASTE Sequenznummer kann jedoch verzichtet werden, wenn der Verlust einzelner Dateneinheiten für die Endsysteme tolerierbar ist.

Dieser Umstand ist in Abbildung 4.14 dargestellt. Sie zeigt potentielle Teilpfad einer Assoziation, wobei damit alle Datenübertragungsabschnitte einer Assoziation gemeint sind, auf der individuelle Transportprotokolle verwendet werden (können). Tatsächlich zeigt Abbildung 4.14 nicht nur die Teilpfade, sondern alle anderen möglichen Warteschlagen mit im Transit befindlichen Daten einer Kommunikationsassoziation.

Initial werden die Daten von einer Anwendungsinstanz an die lokale PASTE Instanz übergeben, wo sie gepuffert werden *(1)*, bevor sie über eine Transportverbindung *(2)* an das Zielsystem den *Server* gesendet werden. Auf diesem werden die Daten ebenfalls gepuffert *(3)*, bevor sie an die Anwendungsinstanz des *Servers* gereicht werden. Wird nun ein neuer *Companion* zwischengeschaltet *(Companion A)*, werden neue Transportverbindungen etabliert. Vom Startsystem dem *Client* zum *Companion A (4)* und von *Companion A* zum *Server (6)*. Auf *Companion A* werden eingehende Daten ebenfalls gepuffert *(5)* bevor sie an den *Server* weitergereicht werden.

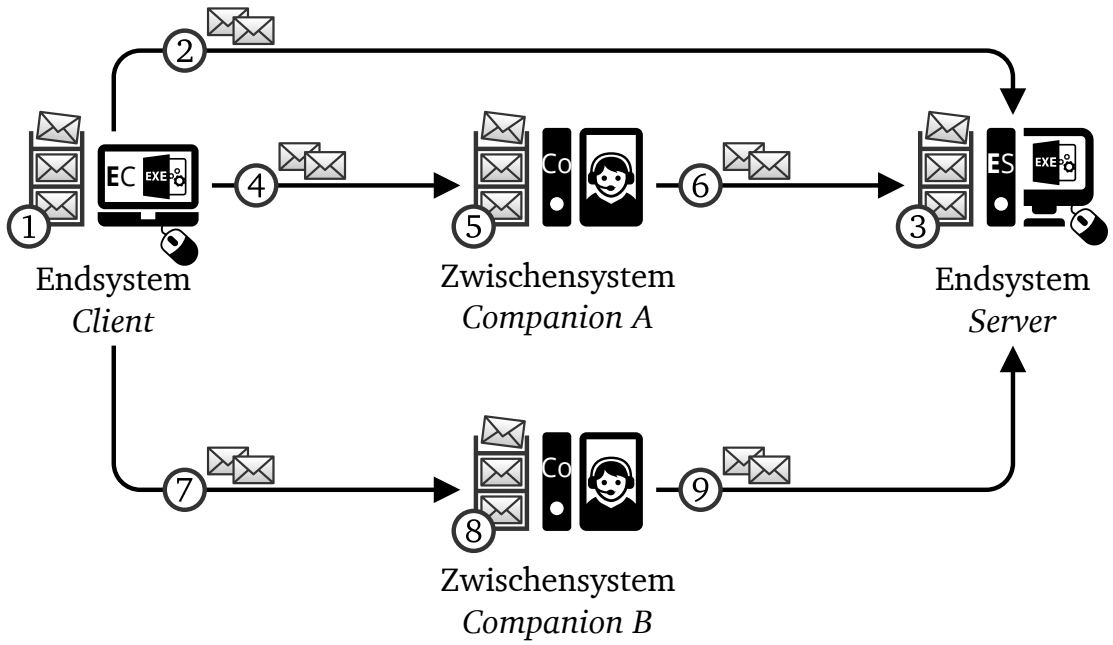


Abbildung 4.14: *Teilpfade und Warteschlangen einer Kommunikationsassoziation beim Einsatz von Companions.*

Soll zu einem späteren Zeitpunkt der *Companion* gewechselt werden zu *Companion B*, so werden erneut zwei neue Transportverbindungen *(7)* und *(9)* aufgebaut. Während die bisherigen Teilpfade, jeweils abgebaut werden. Je nachdem auf welchem Teilpfad noch ausstehende Daten unterwegs sind, können jedoch beliebige Transportverbindungen für eine bestimmte Zeit parallel existieren, bevor diese abgebaut werden.

Teilpfade einer Assoziation sind demnach in Abbildung 4.14 die Abschnitte *(2)*, *(4)*, *(6)*, *(7)* und *(9)*.

4.7.1 Companiondienstmanager

Sofern eine PASTE Instanz auf einem System im Netz als *Companion* fungiert und Companiondienste anbietet, werden diese vom Companiondienstmanager verwaltet. Für jeden angebotenen Companiondienst existiert genau ein *Companion* Dienstlogik Modul, welches den konkreten Dienst umsetzt. Genau wie Anwendungsdienste die sich, wie in Abschnitt 4.5.1 beschrieben, bei einem Namensdienst registrieren müssen, werden Companiondienste beim Start einer PASTE Instanz bei einem *Companion* Verzeichnis registriert. Dies wird für jedes *Companion* Dienstlogik Modul durchgeführt und vom Companiondienstmanager aktualisiert, wenn sich die Adresse beziehungsweise Adressen des Systems auf dem die PASTE Instanz läuft ändern oder zu einem späteren Zeitpunkt *Companion* Dienstlogik Modul entfernt oder hinzugefügt werden.

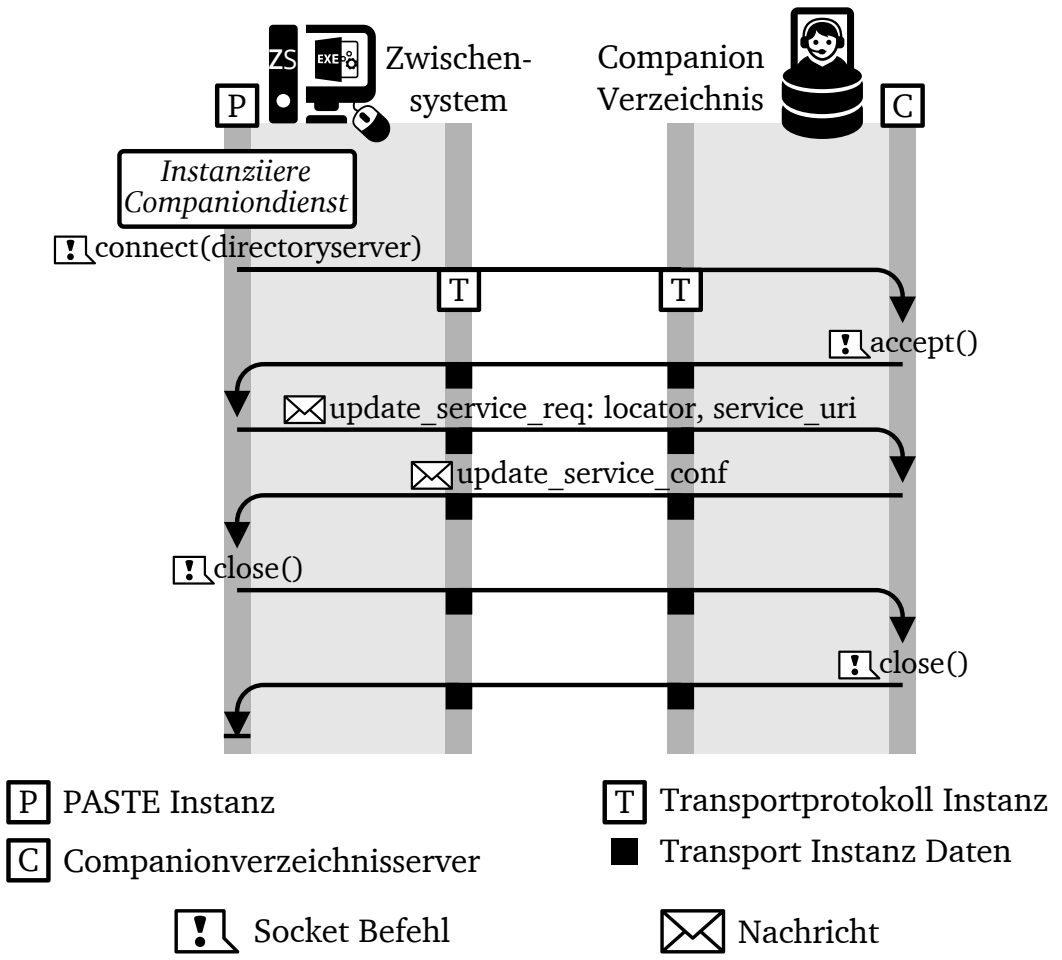


Abbildung 4.15: *Registrieren eines Companiondienstes beim Companionverzeichnis.*

Abbildung zeigt den Ablauf für einen angebotenen Companiondienst. Nachdem von der PASTE Instanz (*P*) ein zugehöriges Companiondienstlogik Modul instanziiert wurde, wird eine Verbindung zum Companionverzeichnis (*C*) aufgebaut. Companiondienste werden durch einen eindeutigen Namen identifiziert, der für neue Companiondienste frei gewählt werden kann. Die einen Companiondienst umsetzenden Companiondienstlogik Module können unter diesem Namen gefunden werden und enthalten zusätzlich Informationen über die von ihnen beeinflussten Transportdiensteigenschaften. Mit einer `update_service_req` Nachricht wird daher neben einem Lokator unter dem ein *Companion* aktuell erreichbar ist auch der Name des zu registrierenden Companiondienstes an das Verzeichnis (*C*) geschickt. Dieses bestätigt das erfolgreiche Anlegen oder Erneuern eines Eintrages mit einer `update_service_conf` Nachricht. Anschließend kann die Verbindung zum Companionverzeichnis (*C*) wieder abgebaut werden.

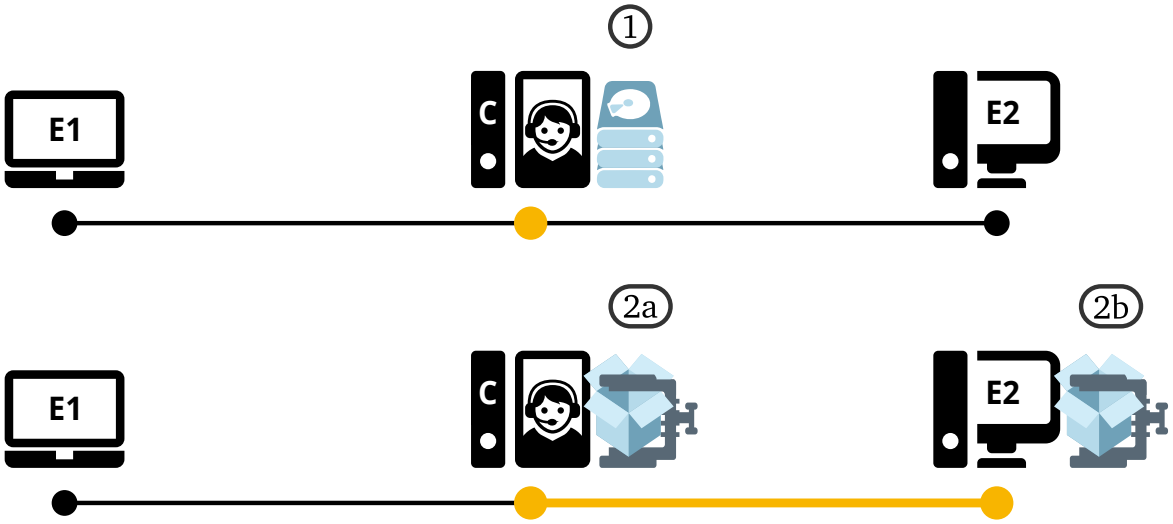


Abbildung 4.16: *Companiondienst Arten: Auf einem Knoten oder auf einem Pfad.*

Companions können also einen dedizierten Companiondienst unterstützen oder parallel mehrere Companiondienste anbieten. Je nach Leistungsfähigkeit des Systems jeweils für nur einen Nutzer beziehungsweise einen kleinen eingeschränkten Nutzerkreis oder auch für eine Vielzahl von Nutzern. Es existieren dabei zwei Arten von Companiondiensten, welche in Abbildung 4.16 dargestellt sind:

- Die zur Erbringung eines Companiondienstes benötigte Companiondienstlogik kann auf einem einzelnen System beziehungsweise Knoten liegen, der an einer Assoziation beteiligt ist. Dies ist beispielsweise der Fall, wenn Daten im Netz zwischengespeichert werden sollen (*Abbildung 4.16 1*). Diese Dienste ändern die zu übertragenden Daten entweder nicht, oder manipulieren die Daten nur in einer Weise, die auf dem Zielsystem nicht wieder rückgängig gemacht werden muss. Der letztere Fall kann beispielsweise bei einem *Concast* eintreten, wenn aufeinanderfolgende Pakete im Netz aggregiert werden.

- Die zur Erbringung eines Companiondienstes benötigte *Companion* Dienstlogik kann auf zwei Systemen beziehungsweise Knoten liegen, die sich am Anfang und am Ende eines Teilpfades einer Assoziation befinden. Dies ist beispielsweise der Fall, wenn Daten im Netz komprimiert werden sollen (*Abbildung 4.16 2a & 2b*). Diese Dienste verändern die Daten am Anfang eines Teilpfades und benötigen eine entsprechende Funktion, um diese Änderung rückgängig zu machen. In diesem Fall müssen die komprimierten Daten beispielsweise auf dem Zielsystem wieder entkomprimiert werden, bevor sie von PASTE an die Anwendungsinstanz gereicht werden können. Bevor ein passender *Companion* für einen solchen Dienst gesucht und verwendet werden kann, muss in diesem Fall zuvor überprüft werden, ob

das Zielsystem ein entsprechendes *Companion* Dienstlogik Modul kennt und lokal verfügbar hat. Dies geschieht anhand des Namens des Companiondienstes.

Wird ein bestimmter Companiondienst von einem der Endsysteme einer Assoziation angefordert, wird auf einem den Companiondienst anbietenden *Companion* das benötigte *Companion* Dienstlogik Modul für die Assoziation instanziiert und der Assoziation zugeordnet. Handelt es sich um einen Pfad spezifischen Companiondienst, so muss abhängig vom Datenstrom (*upstream* oder *downstream*) für den der Companiondienst verwendet werden soll, zusätzlich auf dem entsprechenden Endsystem (*Client* oder *Server*) das passende *Companion* Dienstlogik Modul instanziiert werden.

Auf einem *Companion* über eine Transportverbindung eingehende Pakete werden dann PASTE intern an das *Companion* Dienstlogik Modul weitergeleitet, welches mit der gleichen Assoziation verbunden ist. Dort werden die Pakete verarbeitet und anschließend über eine ausgehende Transportverbindung an das Zielsystem weitergeleitet. Auf dem Zielsystem kann je nachdem über welche Transportverbindung Pakete eingehen entschieden werden, ob diese noch ein *Companion* Dienstlogik Modul passieren müssen oder direkt an die Anwendungsinstanz weitergeleitet werden können.

4.7.2 Companionsuche & Companioncache

Bevor ein Companiondienst genutzt werden kann, muss zunächst ein passender *Companion* gefunden werden. Initial können *Companions* die einen bestimmten Companiondienst anbieten über die Companionsuche abgefragt werden. Diese befragt dazu wiederum ein bekanntes Companionverzeichnis, bei dem sich *Companions* zuvor registriert haben. Einmal bekannte *Companions* werden lokal anhand eines Lokators und der verfügbaren Companiondienste im Companioncache gespeichert und können auch mit anderen Endsystemen geteilt werden, so dass Kommunikationspartnern bekannte *Companions* ebenfalls gefunden und verwendet werden können. Der Inhalt des Companioncaches einer anderen PASTE Instanz kann über das *GoodCompany* Protokoll abgefragt werden. Beispielsweise einmalig pro neu aufgebauter Assoziation zu einem Endgerät.

Pro bekanntem *Companion* kann die Companionsuche dann regelmäßig aber mindestens einmal initial bei der Suche weitere Informationen abfragen, welche ebenfalls im Companioncache gespeichert werden. Dazu stehen der Companionsuche verschiedene Suchmetrik Module zur Verfügung (Abbildung 4.3). Neben den verfügbaren Transportprotokollen eines *Companions* und seinen angebotenen Diensten, erfassen die Suchmetrik Module unterschiedliche weitere Informationen: Über die zuletzt erzielte *up-* und *downlink* Bandbreite oder die Latenz zwischen der lokalen PASTE Instanz und dem *Companion*.

Abhängig vom Anwendungsfall beziehungsweise dem gewünschten Transportdienst und den zur Unterstützung in Frage kommenden Companiondiensten, sind unterschied-

liche Parameter neben dem angebotenen Companiondienst selbst relevant. Beispiele dafür sind der *Abstand* zwischen *Companion* und Sender oder Empfänger einer Assoziation in Form von Latenz oder des *Hop-Counts* zu einem System. Die Bandbreite mit der ein *Companion* angebunden ist, der geringste Jitter zwischen zwei Systemen oder die Verfügbarkeit eines *Companions*.

Die Suchmetrikmodule bieten abhängig vom Transportdienst also das Finden geeigneter *Companions* nach unterschiedlichen Kriterien an und greifen dabei auf Informationen zurück, die entweder von einer zurückliegenden Suchanfrage im *Companion* Cache abgelegt worden sind oder aktiv von einem *Companion* angefragt werden. Dazu wird auch auf Informationen zurückgegriffen, welche vom Ressourcenmanager bereitgestellt werden oder vom Netzwerksinformationsmanager erhoben wurden.

4.8 Aktivitätszustand

Der Aktivitätszustand einer Assoziation ist Teil des bereits in Abschnitt 4.5.2 beschriebenen Assoziationszustandsobjektes. Für andauernde Kommunikationsassoziationen gibt er grundsätzlich an, ob aktuell Daten ausgetauscht werden. Beim Einsatz eines *Companions* wechselt der Aktivitätszustand der beteiligten Systeme je nachdem auf welchen Teilpfaden der Assoziation gerade Daten übertragen werden. So kann sich beispielsweise eine Assoziation auf Senderseite im Zustand *ruhend* befinden, da der Sender bereits sämtliche Daten an einen *Companion* übertragen hat. Während sich die Assoziation beim *Companion* und dem Empfänger noch im Zustand *aktiv* befindet, da der *Companion* noch vom Sender erhaltene Daten an den Empfänger schickt. Der Aktivitätszustand kann folgende Werte annehmen:

- Uninitialisiert - In diesem Zustand befindet sich jede Kommunikationsassoziation initial, nach dem Erzeugen eines lokalen Assoziationszustandsobjektes.

- Aktiv - Daten werden aktuell aktiv über mindestens eine Transportprotokollverbindung übertragen.

- Ruhend - Aktuell werden von diesem System keine Daten übertragen, da eine der Anwendungsinstanzen nicht verfügbar ist oder über einen bestimmten Teilpfad der Assoziation bereits sämtliche Daten übertragen wurden. Es sind aber möglicherweise noch ausstehende Daten für eine spätere Übertragung vorhanden.

- Unbestätigt - Es werden weder Daten übertragen noch sind ausstehende Daten vorhanden. Weiterhin werden auch keine eingehenden Daten mehr erwartet. Es können aber noch ausstehende Bestätigungen existieren.

- Lauschend - Anwendungsinstanzen welche ihre Anwendungsdienste über das PASTE Rahmenwerk anbieten öffnen hierzu lokale Kommunikationsassoziationen. Diese befinden sich in diesem Zustand, während sie auf eingehende Assoziationsaufbau Anfragen anderer PASTE Instanzen warten um auf diese reagieren zu können. Dazu muss die dienstanbietende Anwendungsinstanz aktiv mit PASTE verbunden sein und ihren Dienst per BIND registriert haben, wie in Abschnitt 4.5.1 erläutert.
- Geschlossen - Die Datenübertragung wurde erfolgreich beendet und die Assoziation auf allen beteiligten Systemen abgebaut, beziehungsweise ein Anwendungsdienst wird nicht länger angeboten und die entsprechende lokale Assoziation entfernt.
- Fehlgeschlagen - Der Assoziationszustand ist lokal unbekannt oder unvollständig; und keine der bekannten involvierten PASTE Instanzen hat zu dieser AssoziationsID vollständige Informationen.

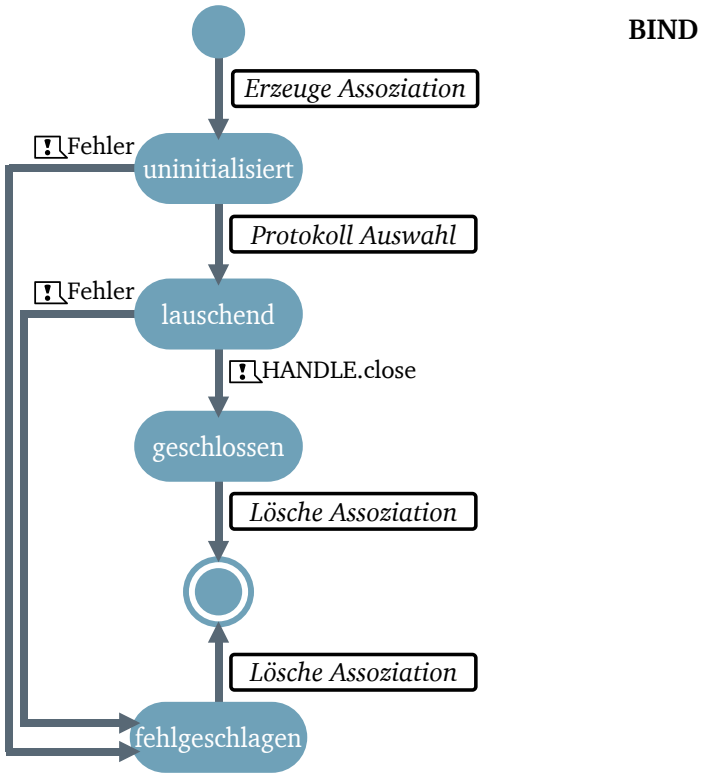


Abbildung 4.17: *Zustandsautomat des Aktivitätszustands für eine per BIND erzeugte Assoziation.*

Der Letzte Zustand kann eintreten, wenn kritische nicht behebbare Fehler beim Datentransport auftreten, zum Beispiel durch den dauerhaften Ausfall eines am Datentransport beteiligten Systems. Fehlgeschlagene Assoziationen werden zunächst weiter verwaltet

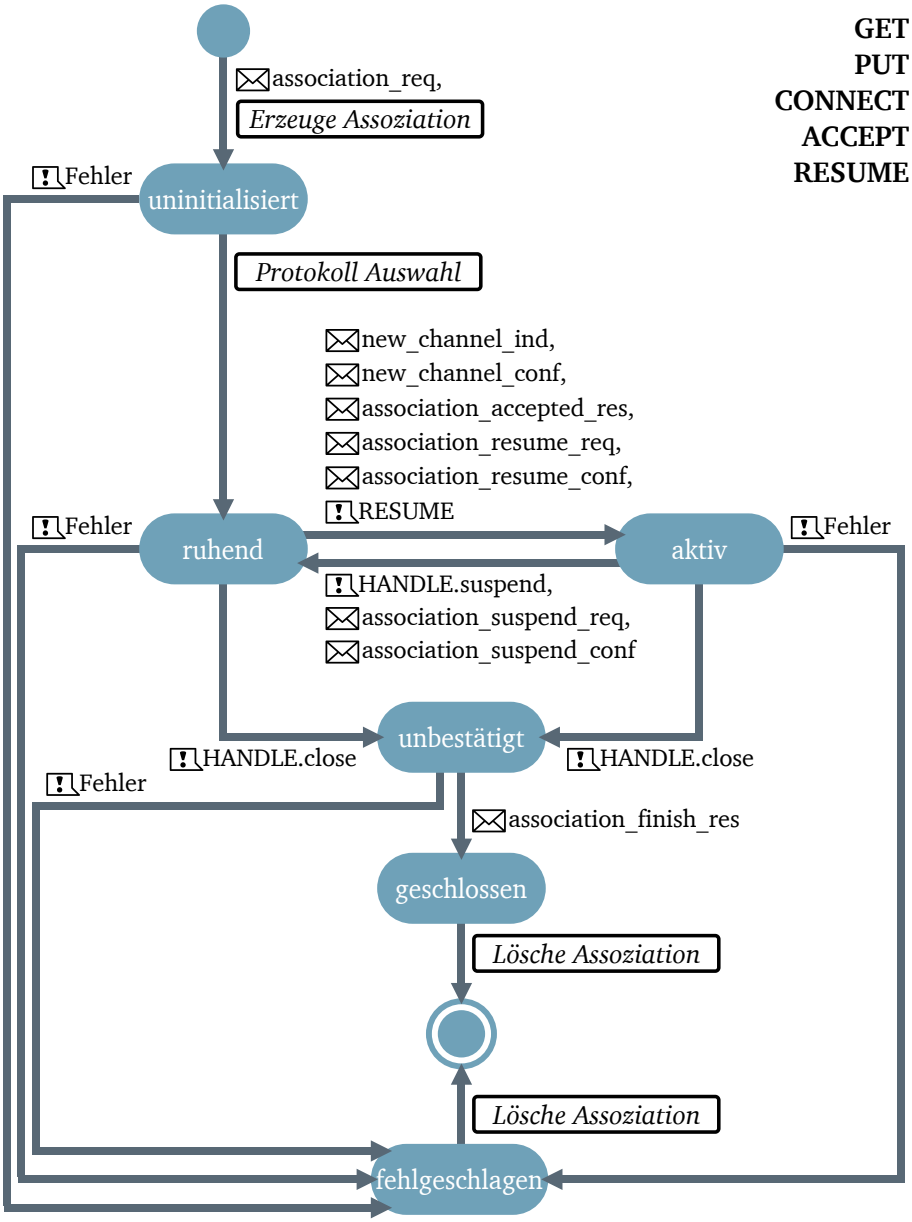


Abbildung 4.18: *Zustandsautomat des Aktivitätszustands für eine per GET, PUT, CONNECT, ACCEPT oder RESUME erzeugte beziehungsweise wiederaufgenommene Assoziation.*

und nicht direkt verworfen. Dies ermöglicht die Ausgabe eines entsprechenden Fehlers, wenn eine Anwendung zu einem späteren Zeitpunkt Informationen über eine verlorene AssoziationsID abfragen will. Dies ermöglicht Anwendungen fehlgeschlagene Assoziation zu schließen, auch wenn sie zum Zeitpunkt des auftretenden Fehlers nicht aktiv waren, da sie sich im Zustand *ruhend* befanden. Ansonsten wird die Assoziation nach einem

vom jeweiligen Transportdienst abhängigen Zeitraum von der PASTE Instanz entfernt. Dieser Zeitraum variiert da zum Beispiel ein zuverlässiger, verzögerungstoleranter Dienst in unregelmäßigen Abständen über einen längeren Zeitraum hinweg Daten versenden und empfangen kann. Werden hingegen beispielsweise über eine Assoziation häufig kurze Nachrichten unzuverlässig verschickt, kann diese fehlgeschlagene Assoziation schneller entfernt werden. Daher muss der Zeitraum als Teil der Dienstbeschreibung vom Anwendungsdienstentwickler festgelegt werden. Zudem kann auch eine Maximalgröße des Zeitraums pro PASTE Instanz konfiguriert werden, welcher Vorrang vor der Dienstbeschreibung hat. Dadurch kann die maximale Wartezeit bevor fehlgeschlagene Assoziationen gelöscht werden beschränkt werden, beispielsweise für Systeme, welche viele aktive Assoziationen und viele Assoziationsanfragen verarbeiten müssen.

Abbildung 4.17 und Abbildung 4.18 zeigen einen Überblick der Aktivitätszustände und unter welchen Bedingungen diese sich im Rahmen der Assoziationsverwaltung durch den Assoziationsmanager ändern: Einmal für per BIND erzeugte Assoziationen, welche zum Anbieten von Anwendungsdiensten dienen (Abbildung 4.17) und einmal für Assoziationen, welche per GET, PUT, CONNECT, ACCEPT oder RESUME erzeugt wurden (Abbildung 4.18). Im Laufe des *Companion* Anwendungsfalls in Abschnitt 4.9 wird ebenfalls noch näher auf einige der Nachrichten eingegangen.

4.9 Companion Anwendungsfall

Im folgenden Anwendungsfall, soll eine größere Datenmenge zum Beispiel in Form einer großen Datei übertragen werden. Die Ausgangssituation ist in Abbildung 4.19 dargestellt. Endsystem E1 möchte an das mobile Endsystem E2 eine größere Datenmenge senden. Eine Kommunikationsassoziation ist in der Abbildung bereits etabliert und Daten werden bereits übertragen.

Zuvor hat sich also Empfänger E2 in einen empfangsbereiten Zustand versetzt. Dazu hat sich seine Anwendungsinstanz an einen eindeutigen Namen gebunden, welcher von seiner PASTE Instanz per im Netz verfügbaren Namensdienst bekanntgegeben wurde. Dann kann sich der Sender E1 zum Empfänger E2 verbinden, indem er zum Beispiel eine zuverlässige, Reihenfolge erhaltende, Fehler überprüfende Assoziation anfordert. Da zwischen Sender und Empfänger nur TCP als Protokoll verfügbar ist, welches diese Anforderung erfüllt, wird von PASTE eine Assoziation erzeugt, welche eine TCP Verbindung aufbaut. Dabei wird direkt die beidseitige PASTE Unterstützung festgestellt. Nun startet die Anwendungsinstanz auf der Senderseite E1 einen großen Dateitransfer.

Bei Endsystem E1 handelt es sich um einen kabelgebundenen Rechner an einem leistungsfähigen Internetanschluss, hier mit einer maximalen Datentransferrate von 1 Gbit/s *upstream*. Da der Empfänger E2 der Daten allerdings nur mit einer geringeren

Datentransferrate von 100 Mbit/s *downstream* angebunden ist, kann Endsystem E1 seinen *upstream* nicht vollständig ausnutzen.

Abbildung 4.19: *Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung.*

Abbildung 4.20: Endsystem E1 entscheidet sich daher, einen Companiondienst einzusetzen. Zunächst wird im Companion Cache nach einem entsprechenden Companion gesucht. Fall sich dort keiner findet, wird eine Anfrage an das Companion Verzeichnis gestellt. Ist ein passender Companion gefunden fordert E1 mittels `companion_service_req` einen Datenkompressionsdienst an. Dies teilt E1 zudem E2 mittels `companion_service_ind` mit, damit E2 prüfen kann ob ein Kompressions Companion Dienstlogik Modul lokal verfügbar ist und dieses für die Assoziation instanziieren kann. Haben beide Systeme die Anfragen mittels `companion_service_conf` beziehungsweise `companion_service_res` bestätigt, wird der Assoziationszustand von E1 zum Companion C übertragen und bestätigt, mittels `association_transfer_ind` und `association_transfer_conf`. Anschließend kann die PASTE Instanz von E1 die bestehende TCP Verbindung abbauen und es werden neue TCP Verbindungen zwischen Sender E1 und Companion C sowie Companion C und Empfänger E2 durch PASTE aufgebaut. Anschließend kann der Sender E1 unter Ausnutzung einer besseren Datentransferrate seine Daten an den Companion C schicken, der diese komprimiert an den Empfänger E2 weiterleitet und die ankommenden Daten des Senders puffert, bis die PASTE Instanz des Empfängers diese entgegennehmen kann.

Der Empfänger benötigt in diesem Fall lokal genug Rechenleistung, um die Daten zu dekomprimieren. Da das effiziente Komprimieren von Daten im Allgemeinen aufwendiger ist, als das anschließende Dekomprimieren, kann aber auch der Empfänger vom Companion profitieren. In diesem Fall tauscht der Empfänger eine kürzere Übertragungszeit und einen geringeren Datenverbrauch zu Ungunsten des gestiegenen Rechenaufwands zum

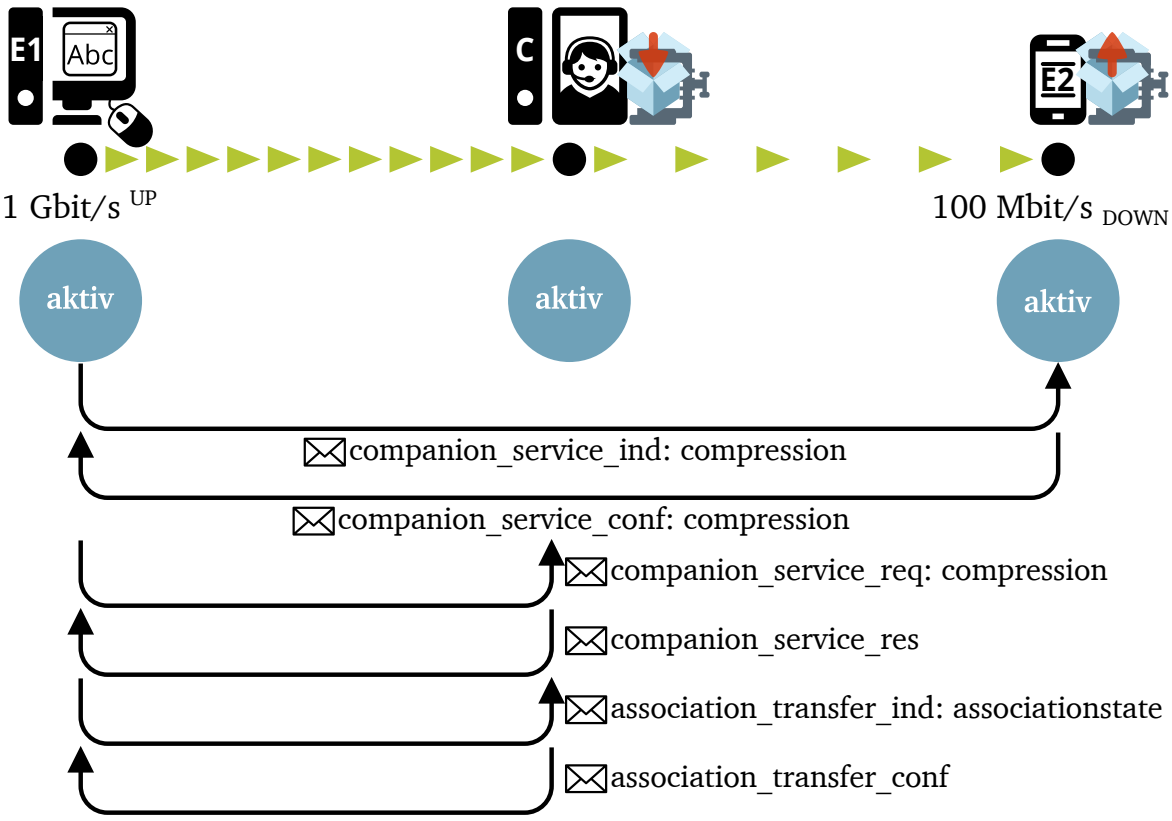


Abbildung 4.20: *Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung.*

dekomprimieren der Daten ein. Art und Stärke der Datenkompression werden zwischen den beiden PASTE Instanzen durch das Companion Dienstlogik Modul vorgegeben.

Abbildung 4.21: Sobald der Sender E1 sämtliche Daten übertragen hat, schickt er an den Companion C ein `association_finish_req` welches ihm anschließend durch eine `association_finish_res` bestätigt wird. Danach kann der Sender E1 seine Assoziation in den Zustand unbestätigt setzen. Er hat nicht mehr vor weitere Daten zu senden, wartet aber noch auf eine Empfangsbestätigung von E2. Der Companion C weiß zudem, dass keine weiteren Daten zu erwarten sind und muss lediglich noch die bisher erhaltenen an E2 ausliefern.

Abbildung 4.22: Wie zuvor E1 schickt der Companion C, sobald er sämtliche Daten an E2 übertragen hat, an E2 ein `association_finish_req` welches ihm durch eine `association_finish_res` bestätigt wird. Danach kann der Companion C seine Assoziation in den Zustand geschlossen setzen, da weder weitere Daten übertragen werden müssen, noch eine Empfangsbestätigung des Empfängers E2 aussteht.

Abbildung 4.23: Zum Abschluss sendet E2 ein `association_finish_req` an E1. Dieses wird von E1 mit einer `association_finish_res` bestätigt. Der Companion C ist zu

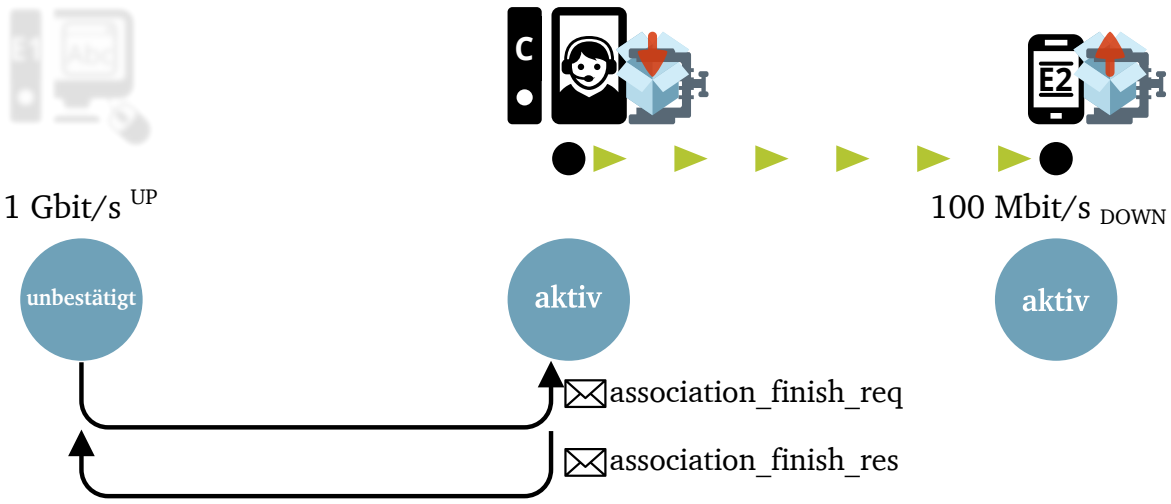


Abbildung 4.21: *Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung.*

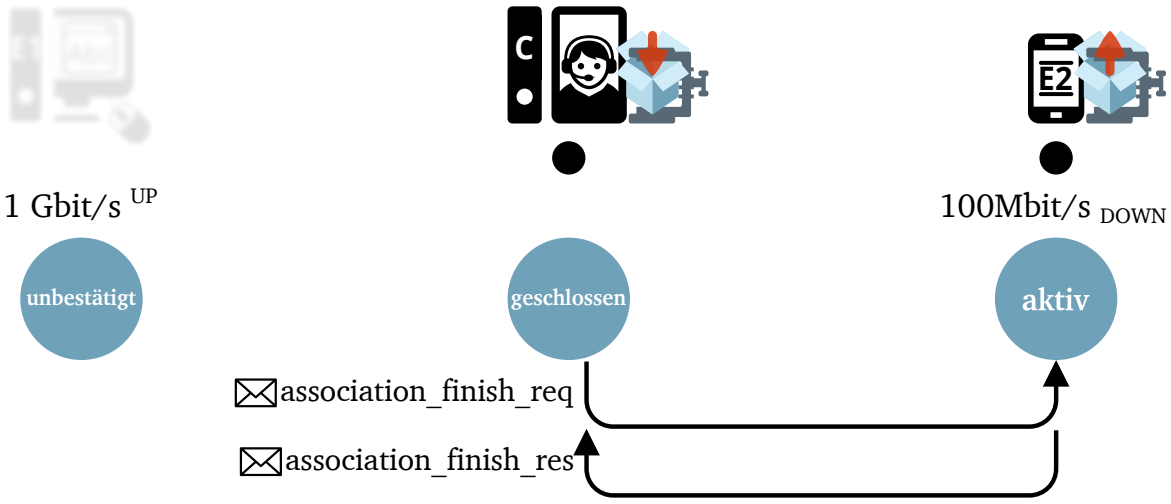


Abbildung 4.22: *Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung.*

diesem Zeitpunkt nicht mehr an der Datenübertragung beteiligt und hat den Assoziationszustand bereits gelöscht. Nachdem E1 und E2 die Assoziation Ende-zu-Ende geschlossen haben, können beide den Assoziationszustand auf geschlossen setzen, da keines der Endsysteme weitere Daten senden möchte oder ausstehende Daten erwartet.

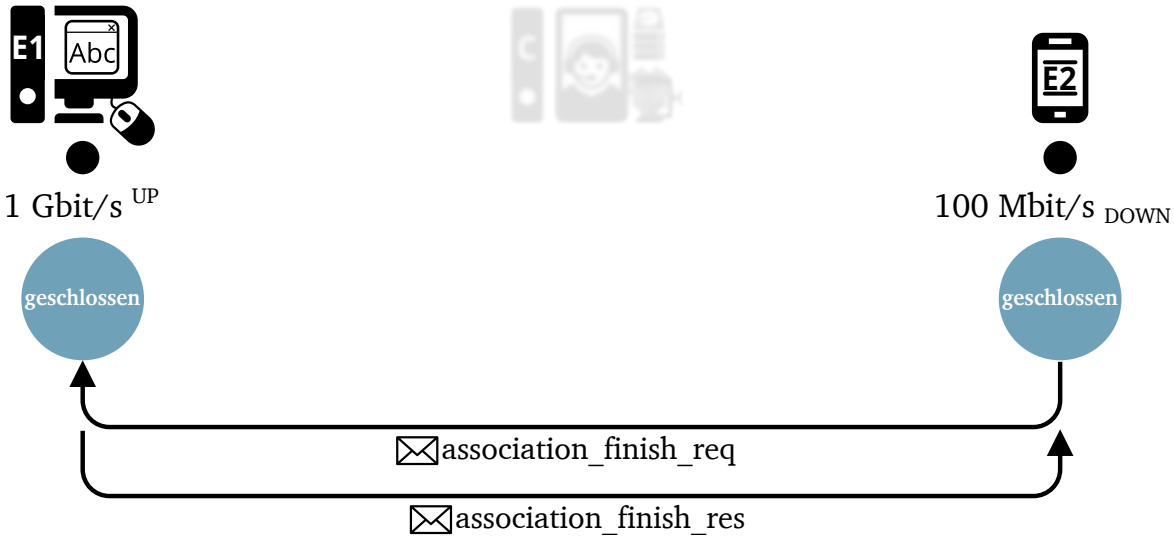


Abbildung 4.23: *Eine Dateiübertragung mit Companion Unterstützung zur Datenkomprimierung.*

4.10 Zusammenfassung

In diesem Kapitel wurde das Protocol Agnostic Services & Transport Enrichment (PASTE) Rahmenwerk vorgestellt und sein Aufbau sowie die einzelnen Komponenten aus denen es besteht ausführlich erläutert. Anhand von Anforderungen und Zielen des Rahmenwerks wurde, ausgehend von der in Kapitel 3 vorgestellten PASTE-API, erläutert, wie sich Protokoll-agnostische Transportdienste für Anwendungen umsetzten lassen.

Dabei wurde insbesondere auf die Kommunikationsassoziationsverwaltung durch das PASTE Rahmenwerk eingegangen. Anschließend wurde beschrieben, wie erweiterte Transportdienste durch die Nutzung von Zwischensystemen im Netz umgesetzt werden können, beziehungsweise wie diese sich unterstützen lassen. Dazu wurden *Companions* eingeführt, im Netz verortet Zwischensysteme auf denen über das PASTE Rahmenwerk für Endsysteme verschiedene Netzwerkdienste angeboten werden können. Die Umsetzung und der Einsatz von Companions wurde anhand eines Beispiels näher erläutert.

Kapitel 5

Evaluation und Implementierung

Um einen Einblick in das Verhalten des in dieser Arbeit entwickelten PASTE Rahmenwerkes, der PASTE-API und der Transportdienstbeschreibungen zu erhalten, wird im Folgenden zunächst eine analytische Betrachtung der einzelnen Komponenten angestellt. Neben den bereits in Kapitel 3.6 vorgestellten Anwendungsfällen der PASTE-API, wurde die Anwendungsschnittstelle bereits partiell als Teil des NENA Rahmenwerks [MVZ11; MWB11] implementiert. Anhand einiger Arbeiten wird zunächst ihre Einsetzbarkeit zur Umsetzung verschiedener Kommunikationsparadigmen aus Anwendungssicht gezeigt, unabhängig vom PASTE Rahmenwerk. Dabei wird auch der durch die Transportdienstbeschreibungen eingeführte Zusatzaufwand betrachtet. Zusätzlich wird ein quantitativer Codevergleich zwischen PASTE-API und Berkeley Socket-API angestellt.

Die Umsetzbarkeit des PASTE Rahmenwerks selbst soll qualitativ gezeigt werden. Anhand einer prototypischen Implementierung des PASTE Rahmenwerks, wird die Funktionalität des in Kapitel 4.9 vorgestellten Anwendungsfalls gezeigt. Ferner werden verschiedene Leistungsparameter für drei umgesetzte Beispielszenarios quantitativ betrachtet. Als Vergleich wird dazu eine Datenübertragung ohne das PASTE Rahmenwerk mit Hilfe der Socket-API herangezogen. In beiden Fällen kommt TCP als verwendetes Transportprotokoll zum Einsatz. Durch das PASTE Rahmenwerk erzielte Leistungszuwächse und -einbußen für verschiedene Parameter werden dabei aufgezeigt.

5.1 Anwendbarkeit der PASTE-API

Die in Kapitel 3 beschriebene PASTE-API bildet die Schnittstelle zwischen Anwendungen und der unterliegenden Netzwerkinfrastruktur. Ihre wesentliche Aufgabe ist die Entkopplung der Anwendungen von konkreten im unterliegenden Netzwerk verwendeten Protokollen. Eigenschaften der Netzwerkinfrastruktur wie etwa die Adressierung, Na-

mensauflösung oder Ports, sollen transparent für die Anwendung sein. Um die universelle Einsetzbarkeit der PASTE-API zu zeigen, wurden neben der Umsetzung für heutige Protokolle durch das PASTE Rahmenwerk verschiedene Arbeiten zur Abbildung auf weitere Netzwerkarchitekturen durchgeführt:

- Content-Centric Networking (CCN) beziehungsweise Named Data Networking
- Publish/Subscribe mittels Message Queue Telemetry Transport (MQTT)
- Eine NENA basierte Video-on-Demand Webstore Demonstration

5.1.1 Content-Centric Networking (CCN)

Im Gegensatz zu heutigen IP Netzwerken werden bei Content Centric Networking (CCN) [Jac+09a] direkt Inhalte anstatt Endsysteme adressiert. Die Grundidee ist, dass jeder Router über einen großen Speicher verfügt und Inhalte zwischenspeichern kann. Damit sollen sich Inhalte unabhängig von einer bestimmten Lokation im Netz verbreiten und im Optimalfall dort gespeichert werden, wo ein bestimmter Inhalt häufig abgerufen wird.

Wird von einem Endsystem eine Inhaltsanfrage gestellt, so wird diese im Netz verbreitet, bis ein System den gewünschten Inhalt zurückliefern kann. Auf dem Weg zum Zielsystem durchläuft der Inhalt dabei den Pfad der erfolgreichen Anfrage in inverser Reihenfolge und die passierten Zwischensystem speichern den Inhalt vor dem Weiterleiten lokal. Zukünftige Anfragen für den selben Inhalt können danach von allen passierten Zwischensystem beantwortet werden.

Dadurch kann der Netzwerkverkehr für wiederholte Anfragen oder Anfragen anderer Systeme in der gleichen Netzregion reduziert werden. Zudem kommt es auch zu keinen Lastengpässen an einem Server der alleine für einen Inhalt zuständig ist. Anfragen heißen bei CCN *Interests*. Bei Empfang eines *Interests*, welches nicht beantwortet werden kann, wird es weitergeleitet und in einer *Pending Interest Table* gespeichert, über welches Interface das *Interest* empfangen wurde. Dadurch kann später bei Beantwortung des *Interests* durch ein anderes System der Inhalt zum ursprünglichen *Interest* Steller zurückgeleitet werden. Dadurch wird auch verhindert das Daten doppelt versendet werden, da *Interests* von jedem System nur einmal beim ersten Erhalt beantwortet werden.

Ähnlich zu HTTP verwendet CCN *Get* und *Put* zum Abrufen und Ablegen von Inhalten im Netz und verwendet statt IP Adressen ausschließlich Namen. Da die PASTE-API einen ähnlichen Ansatz verfolgt, also die Adaption der HTTP Dienstprimitive und eine Namensbasierte Adressierung, ist die Abbildung der PASTE-API auf CCN für die Dienstprimitive GET und PUT eins zu eins möglich.

CCN unterstützt allerdings nicht direkt interaktive Ende-zu-Ende Verbindungen zwischen zwei Anwendungsinstanzen, wie beispielsweise für Sprachkommunikation. In der Arbeit „Realisierung des Content Centric Networking Ansatzes in der Netlet-based Node Architecture“ [Fun12] wurde daher zusätzlich ein Lösungsansatz umgesetzt, welcher in [Jac+09b] vorgeschlagen wurde. Dabei werden die Sprachdaten in kleine Einheiten zerlegt und in Datenpakete unterteilt. Diese werden konstant per *Interest* angefordert. Mit Hilfe der PASTE-API konnten daher in der Arbeit zusätzlich zu den Primitiven GET und PUT die Primitiven BIND und CONNECT unterstützt werden, welche von der *Netlet-based Node Architecture* umgesetzt wurden. Bei der *Netlet-based Node Architecture* (NENA) handelt es sich um ein Rahmenwerk zur Unterstützung zukünftiger Netze. Ein Aufruf von BIND hat demnach zur Folge, dass ein Endsystem bei einer eingehenden Assoziationsanfrage zu versendende Daten in kleine Dateneinheiten unterteilt und per PUT im Netz ablegt. Während per CONNECT automatisch kontinuierlich *Interests* per GET Primitive generiert werden.

5.1.2 Message Queue Telemetry Transport (MQTT): Publish/Subscribe

Message Queue Telemetry Transport (MQTT) [SN16] ist ein Client Server publish-subscribe basiertes Protokoll für den Austausch von Nachrichten. MQTT wurde als leichtgewichtiges Protokoll für den Einsatz im Bereich Maschine-zu-Maschine Kommunikation entworfen und findet auch im Internet der Dinge (Internet of Things - IoT) Anwendung. Es war ursprünglich für die Nutzung oberhalb von TCP/IP oder jedem anderen Transportprotokoll, welches einen reihenfolgetreuen, verlustfreien und bidirektionale Dienst bereitstellt, vorgesehen. Im Kontext des Internets der Dinge findet es auch häufig in nicht TCP/IP basierten Netzen Anwendung und wird beispielsweise in ZigBee Netzen eingesetzt.

Das publish-subscribe Kommunikationsmuster sieht einen Nachrichten-Broker vor. Dieser ist für die Verteilung von Nachrichten an interessierte Abonnenten (Subscriber) basierend auf dem Thema (Topic) einer Nachricht zuständig. Abonnenten melden dabei Interesse an bestimmten Themen an und erhalten somit alle Nachrichten die diesen Themen zugeordnet sind.

MQTT unterstützt dabei drei verschiedene Dienstklassen:

- „At most once“ (höchstens einmal): Nachrichten werden *Best-effort* zugestellt. Dabei können Nachrichten auch verloren gehen, da nur ein einziger Zustellversuch unternommen wird. Diese Klasse ist beispielsweise für Daten von Umgebungssensoren geeignet, bei denen ein einzelnes Datum nicht so wichtig ist da ihre Werte regelmäßig in kurzen Abständen publiziert werden.

- „At least once“ (mindestens einmal): Nachrichten werden sicher zugestellt, da der Empfang bestätigt wird. Da bei Ausbleiben einer Bestätigung weitere Zustellversuche unternommen werden, können Nachrichten bei Verlust einer Bestätigung allerdings mehrfach zugestellt werden.

- „Exactly once“ (genau einmal): Nachrichten werden genau einmal zugestellt. Dies erfordert den Aufbau einer Verbindung mit 3-Wege-Handshake zwischen dem Broker und dem Abonnenten. Diese Klasse ist beispielsweise für den Einsatz in Abrechnungssystemen gedacht, bei denen duplizierte oder verlorene Nachrichten zu Fehlern beispielsweise in Form von falschen Abbuchungen führen können.

In der Bachelorarbeit „Implementation of MQTT protocol inside NENA“ [Pra13] wurde die Abbildung der PASTE-API auf eine MQTT umsetzende Netzwerkarchitektur betrachtet. Wie sich das publish-subscribe Kommunikationsmuster allgemein umsetzen lässt, wurde bereits beispielhaft anhand eines Anwendungsfalles in Kapitel 3.6.4 gezeigt. Analog wurden für MQTT die Dienstprimitive GET und PUT verwendet, um mittels GET ein *Topic* zu abonnieren und mittel PUT Nachrichten für ein bestimmtes *Topic* zu versenden. Abonnenten öffnen also eine Kommunikationsassoziation pro *Topic* und können eintreffende Nachrichten per `read(..)` Methode über den zugehörigen PASTE-Handle abrufen. Analog werden Nachrichten per `write(..)` Methode verschickt über zuvor mit PUT geöffnete Kommunikationsassoziationen.

Die *Topics* selbst werden dabei über die Wahl der URI umgesetzt. Ein Broker bietet seinen Dienst per BIND Dienstprimitive im Netz unter einem festen Namen an:

```
mqtt://service-provider/broker/topics
```

Damit ist er für alle Unternamen zuständig und kann anhand von PUT und GET bei eingehenden Kommunikationsassoziationsanfragen unterscheiden ob über diese Nachrichten versendet oder empfangen werden sollen. Abonnenten und Sender verwenden entsprechende URIs, um ein *Topic* zu wählen:

```
GET(mqtt://service-provider/broker/topics/gossip, [service_desc])
PUT(mqtt://service-provider/broker/topics/temperature_KA_city, TCPlike)
```

Für die Umsetzung der Dienstklassen werden zwei Möglichkeiten betrachtet:

- Als Teil der Dienstbeschreibung

- Als Parameter der URI

Sofern man annimmt, dass die Dienstklassen als standardisierte Eigenschaften im Transportdiensteigenschaften Katalog existieren, können diese im Rahmen der Dienstbeschreibung übergeben werden. Für die „At most once“ Klasse eignet sich jeder unzuverlässige

Best-effort Transportdienst. Die „At least once“ Klasse und „Exactly once“ Klasse lassen sich über einen zuverlässigen Transportdienst realisieren, wobei für erste im Rahmen der Dienstbeschreibung auf die Duplikatfreiheit verzichtete werden kann.

Sollen die Klassen auf der Anwendungsschicht umgesetzt werden, kann dies mit Hilfe eines Parameters der URI geschehen, der vom Dienstanbieter, also dem Broker, festgelegt wird:

```
mqtt://service-provider/broker/topics/gossip?class=exactly_once
```

Da die Parameter beim Assoziationsaufbau an die Anwendungsschicht weitergereicht werden, kann ein Broker in dem Fall die entsprechenden Klassen mit eigenen Acknowledgements und Handshake innerhalb der Anwendungsschicht realisieren.

5.1.3 Video-on-Demand Dienst

In [BMW12] wurde als Machbarkeitsnachweis für die Anwendbarkeit der PASTE-API ein Demonstrator auf Basis der NENA Architektur [MVZ11] entworfen. Dazu wurde ein Video-on-Demand Dienst entworfen, der über drei verschiedene Netzwerke mit unterschiedlichen Architekturen realisiert wurde. Der Aufbau ist in Abbildung 5.1 dargestellt.

Die NENA Architektur ist für den parallelen Betrieb verschiedenster Netze gedacht. Es existieren drei Netzwerke, jeweils mit eigenem Namensraum:

- **nenaweb:** Ein Netz zur Präsentation des Video-On-Demand Dienstes im Browser
- **nenacdn:** Ein Netz zum *Download* eines angeforderten Videos
- **nenavideo:** Ein Netz zum *Streamen* eines angeforderten Videos

Der Zugriff auf alle drei Netzwerke erfolgt über die PASTE-API. Dabei benötigt die Anwendung weder Wissen über verwendete Protokolle oder die unterschiedlichen unterliegenden Netzwerke.

Die Anwendung ist in diesem Fall ein modifizierter Webbrowser, welcher die PASTE-API verwendet um Daten abzurufen. Abbildung 5.2 zeigt eine Webanwendung, welche Zugriff auf den Video-on-Demand Dienst gewährt und selbst über die PASTE-API übertragen wird. Mit Hilfe der PASTE-API ist eine intuitive und von der unterliegenden Architektur abstrahierende Umsetzung möglich, da NENA die Netzwerke über verschiedene Namensräume verfügbar macht. Die einzelnen Links der abgebildeten Webseite senden unterschiedliche Anfragen über die PASTE-API ab. Abhängig vom gewählten Namen, der übergebenen Dienstprimitive und dem gewünschten Transportdienst, kann das NENA Rahmenwerk entscheiden welche Art von Verbindung zum Webbrowser aufgebaut werden soll und

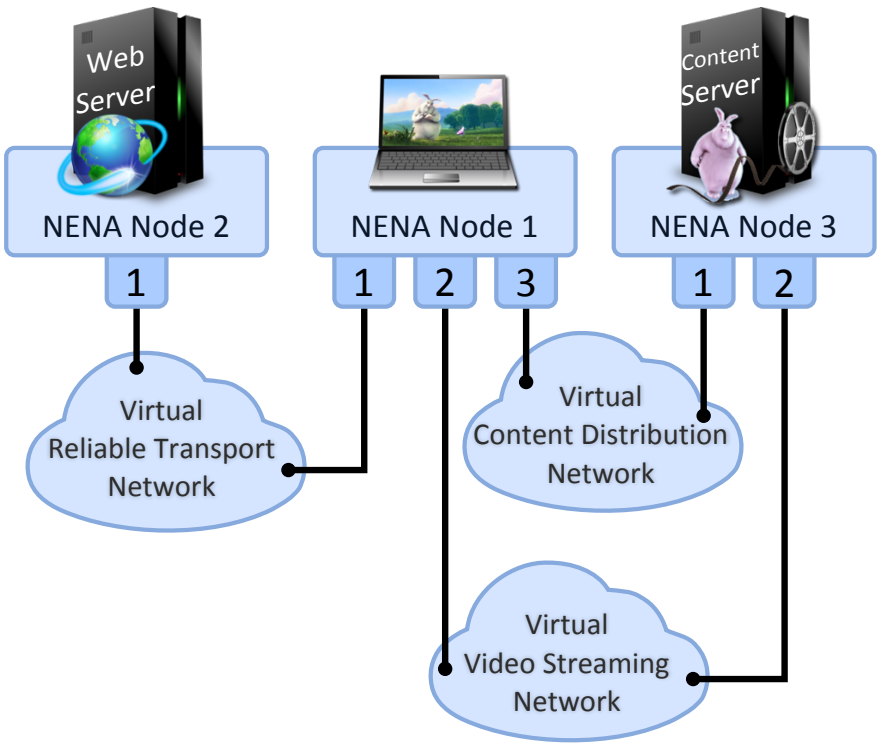


Abbildung 5.1: *Setup des Video-on-Demand Demonstrators.*

über welches Protokoll das angeforderte Video übertragen wird. Wird beispielsweise ein Videostream aus dem *nenavideo* Namensraum angefordert, wird je nachdem ob dabei die GET oder CONNECT Primitive verwendet wird ein nicht-interaktiver oder interaktiver Videostream zurückgeliefert.

5.1.4 Vergleich zwischen PASTE-API und Berkeley SocketAPI

Im Folgenden werden minimale *C* Implementierungen eines einfachen Datentransfers zwischen einem Client und einem Server betrachtet. Dazu wird eine Implementierung, welche die PASTE-API verwendet, einer Implementierung des gleichen Programms mit der Standard Berkeley Socket-API gegenübergestellt. Bei der PASTE-API handelt es sich um eine Umsetzung, welche mit Hilfe der `libuv` Bibliothek umgesetzt wurde. Die Bibliothek stellt eine Ereignis-basierte Programmschleife und Mechanismen zur einfachen Verwendung von Interprozess Kommunikation zur Verfügung.

Das Beispielprogramm besteht aus jeweils einem Client und einem Server. Der Server bindet sich an den Namen `paste://example.server` und wartet auf sich verbindende Clients. Verbindet sich ein Client mit dem Server, bekommt der Client eine zufällige Anzahl von Nachrichten mit dem Inhalt „Lorem ipsum dolor sit amet, consetetur sadipscing“

Listing 5.1: *Beispiel PASTE Server in C.*

```c
#include <stdlib.h>
#include <string.h>

#include <pl_api.h>
#include <pl_events.h>

const char buffer[50] = "Lorem ipsum dolor sit amet, consetetur sadipscing";

// paste_server.c
int main(int argc, char** argv) {
    char *name = "paste://example.server";

    // create server handle and listen for incoming connection requests
    pl_handle* serv_handle = BIND(name, CONNECT, sid_TCPlike);
    init_readEvent(serv_handle, processEvent);

    // run program logic
    uv_run(uv_default_loop(), UV_RUN_DEFAULT);

    close(serv_handle);

    return 0;
}

void processEvent(char* msg, event_type event) {
    if(event == ASSOCIATION_REQUEST) {
        // accept incoming connection request
        pl_handle* handle = ACCEPT(msg);

        // send the string "Lorem ipsum dolor sit amet, consetetur sadipscing"
        // a random number of times
        int n = rand();
        for(int i = 0; i < n; i++) {
            write(handle, buffer, strlen(buffer) + 1); // \0
        }

        uv_stop(uv_default_loop());

        close(handle);
    }
}
```

Listing 5.2: *Beispiel Socket-API Server in C.*

```c
#include <stdlib.h>
#include <string.h>

#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

const char buffer[50] = "Lorem ipsum dolor sit amet, consetetur sadipscing";

// berkeley_server.c
int main(int argc, char** argv) {
    short int port = 12345;
    struct sockaddr_in servaddr;

    // create server handle and listen for incoming connection requests
    int serv_handle = socket(AF_INET, SOCK_STREAM, 0));

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(port);

    bind(serv_handle, (struct sockaddr*)&servaddr, sizeof(servaddr));
    listen(serv_handle, 1024);

    // accept incoming connection request
    int handle = accept(serv_handle, NULL, NULL);

    // send the string "Lorem ipsum dolor sit amet, consetetur sadipscing"
    // a random number of times
    int n = rand();
    for(int i = 0; i < n; i++) {
        size_t nwritten = 0;
        size_t nleft = strlen(buffer) + 1; // \0
        char* send_buffer = buffer;

        while (nleft > 0) {
            nwritten = write(sockd, send_buffer, nleft);
            nleft -= nwritten;
            send_buffer += nwritten;
        }
    }

    close(handle);
    close(serv_handle);

    return 0;
}
```

Listing 5.3: *Beispiel PASTE Client in C.*

```c
#include <stdlib.h>
#include <stdio.h>

#include <pl_api.h>
#include <pl_events.h>

// paste_client.c
int main(int argc, char** argv) {
    char *name = "paste://example.server";

    // create handle and listen for incoming packets
    pl_handle* handle = CONNECT(name, sid_TCPlike);
    init_read(handle, processData);
    init_readEvent(handle, processEvent);

    // run program logic
    uv_run(uv_default_loop(), UV_RUN_DEFAULT);

    close(handle);

    return 0;
}

void processData(char* msg, size_t len) {
    printf("Received: \%s\n", msg);
}

void processEvent(char* msg, event_type event) {
    if(event == ASSOCIATION_CLOSED) {
        uv_stop(uv_default_loop());
    }
}
```

Listing 5.4: *Beispiel Socket-API Client in C.*

```c
#include <stdlib.h>
#include <stdio.h>

#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

// berkeley_client.c
int main(int argc, char** argv) {
    char *name = "paste://example.server";
    short int port = 54321;
    struct sockaddr_in servaddr;
    char* remoteIP;

    // lookup the IP for "paste://example.server"
    struct hostent *he = gethostbyname(name);
    struct in_addr **addr_list = (struct in_addr **) he->h_addr_list;
    for(int i = 0; addr_list[i] != NULL; i++) {
        // Return the first ip
        strcpy(remoteIP , inet_ntoa(*addr_list[i]));
        break;
    }

    // create handle and listen for incoming packets
    int handle = socket(AF_INET, SOCK_STREAM, 0);

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(port);

    inet_aton(remoteIP, &servaddr.sin_addr);
    connect(handle, (struct sockaddr*)&servaddr, sizeof(servaddr);

    int msgSize = 50;
    char buffer[50];
    while(true) {
        size_t nread = 0;
        size_t nleft = msgSize;
        while (nleft > 0) {
            nread = read(sockd, buffer + nread, nleft);
            nleft -= nread;
        }
        if(nread == 0) {
            break;
        }
        printf("Received: \%s\n", buffer);
    }

    close(handle);

    return 0;
}
```

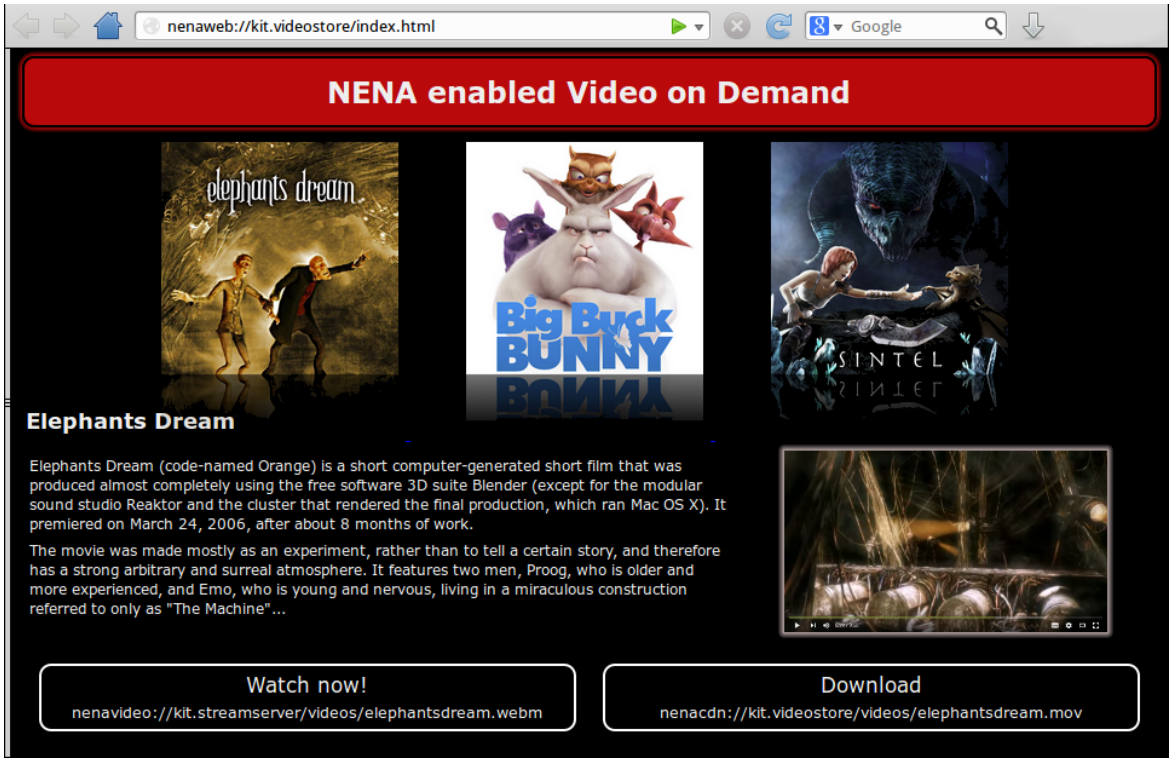


Abbildung 5.2: *Benutzerschnittstelle des Video-On-Demand Demonstrators im Webbrowser. (Quelle & Copyright der Bilder und Videos: Blender Foundation, www.blender.org)*

vom Server zugeschickt. Danach beendet der Server die Verbindung und sich selbst. Nach Erhalt aller Nachrichten beendet sich der Client ebenfalls.

Zunächst wird der PASTE Server in Auflistung 5.1 betrachtet. In Zeile 7 wird ein Puffer mit der zu übertragenden Nachricht angelegt. Innerhalb der *main()* Funktion wird zunächst in Zeile 11 der Name des anzubietenden Dienstes hinterlegt. In Zeile 14 wird der Server HANDLE erzeugt, welcher auf eingehende Verbindungsanfragen wartet. Dazu wird dem *BIND()* Aufruf der hinterlegte Dienstname „paste://example.server“, die gewünschte Primitive (CONNECT) und ein Dienstidentifikator übergeben. Der hier verwendete Dienstidentifikator ist in der PASTE-API vordefiniert und fordert dem Namen entsprechend einen mit TCP vergleichbaren Transportdienst an.

In Zeile 15 wird die *readEvent()* Methode der PASTE-API als Callback registriert. Anschließend wird die von `libuv` bereitgestellte Ereigniswarteschlange gestartet. Für eingehende Ereignisse der PASTE-API wird nun die *processEvent()* Funktion aufgerufen. Innerhalb dieser Funktion wird in Zeile 26 überprüft, ob es sich um eine Assoziationsanfrage handelt. Ist dies der Fall wird in Zeile 28 per Aufruf von *ACCEPT()* die Assoziationsanfrage angenommen. Anschließend wird in einer Schleife die zu übertragende Nachricht über

den Aufruf von *write()* versendet. Danach wird die Assoziation durch den Aufruf von *close()* in Zeile 39 geschlossen.

Der entsprechende Socket-API Server ist in Auflistung 5.2 aufgeführt. Auch hier wird zunächst ein Puffer mit der zu übertragenden Nachricht angelegt. Da die Socket-API keine Namens-basierte Schnittstelle ist, wird der Socket mit einem hinterlegten Port (Zeile 12) an die lokale IP-Adresse des Systems gebunden, was mehrere Aufrufe zur Festlegung der Adressfamilie und des Ports erfordert (Zeile 13–23). Dies ist beim PASTE Server nicht nötig, da sowohl das Protokoll als auch der Port vom PASTE Rahmenwerk automatisch gewählt werden.

In Zeile 24 wird auf eingehende Verbindungsanfragen gewartet. Ebenfalls durch einen Aufruf von *accept()* wird in Zeile 27 eine eingehende Verbindungsanfrage angenommen. Anschließend wird in einer Schleife die zu übertragende Nachricht versendet (Zeile 33–41). Dabei muss die Anwendung jedoch selbst überprüfen, ob eine Nachricht bereits vollständig übertragen wurde und *write()* so lange aufrufen, bis der Schreibvorgang abgeschlossen ist. Dies wird von der PASTE-API für den Entwickler erledigt, da die PASTE-API grundsätzlich ADU Grenzen erhält und selbstständig dafür sorgt, dass Nachrichten in einem Stück verschickt und erhalten werden. Danach wird die Verbindung durch den Aufruf von *close()* in Zeile 44 geschlossen.

Nun wird die Client Seite betrachtet. Auflistung 5.3 zeigt den PASTE Client. Innerhalb der *main()* Funktion wird zunächst in Zeile 9 der Name des Servers hinterlegt. In Zeile 12 wird ein HANDLE erzeugt mit einem Aufruf von *CONNECT()*. Dabei werden der hinterlegte Servername „paste://example.server“ und der gewünschte Transportdienst als Dienstidentifikator übergeben.

In Zeile 13 und 14 wird jeweils die *read()* und *readEvent()* Methode der PASTE-API als Callback registriert. Anschließend wird die von `libuv` bereitgestellte Ereigniswarteschlange gestartet. Für eingehende Nachrichten wird die *processData()* Funktion aufgerufen, welcher die vollständige Nachricht von der PASTE-API übergeben wird. Innerhalb dieser Funktion wird in Zeile 25 lediglich die erhaltene Nachricht durch einen Aufruf von *printf()* vom Programm ausgegeben.

Für eingehende Ereignisse der PASTE-API wird die *processEvent()* Funktion aufgerufen. Innerhalb dieser Funktion wird in Zeile 29 überprüft, ob ein *ASSOCIATION_CLOSED* Ereignis erhalten wurde. Dies wird durch den PASTE Server beim Aufruf von *close()* ausgelöst. Damit erhält der Client eine Benachrichtigung, dass die Assoziation geschlossen wurde und kann sich beenden.

Zuletzt wird der Socket-API Client in Auflistung 5.4 betrachtet. Innerhalb der *main()* Funktion wird in Zeile 10 ebenfalls der Name des Servers hinterlegt, mit dem eine Verbindung aufgebaut werden soll. Zusätzlich wird in Zeile 11 der zu verwendende Port angegeben. Anschließend muss für den Server Namen eine Namensauflösung durchge-

führt werden (Zeile 12–22). Dies entfällt bei der PASTE-API, da die Namensauflösung vom PASTE Rahmenwerk durchgeführt wird.

War die Namensauflösung erfolgreich, wird ein handle erzeugt, um auf eingehende Nachrichten zu warten. Erneut werden dazu im Gegensatz zum PASTE Client mehrere Aufrufe zur Festlegung der Adressfamilie und des Ports benötigt (Zeile 25–32).

Anschließend liest der Socket-API Client ankommende Daten innerhalb einer Endlosschleife (Zeile 37–45). Eingehende Daten können jedoch nicht direkt ausgegeben werden, da zunächst die ursprünglichen ADUs wiederhergestellt werden müssen. Dies ist in diesem Fall sehr einfach gelöst, da alle Nachrichten des Servers die gleiche Größe haben, welche dem Client hier bekannt ist. In der Praxis müsste die erwartete Größe einer eingehenden Nachricht eigentlich noch zusätzlich verschickt werden, was den Code des Socket-API Clients und Servers noch aufwendiger machen würde. Für jede wiederhergestellte ADU wird wie beim PASTE Client *printf()* zum Ausgeben der erhaltenen Nachricht aufgerufen.

Im Gegensatz zur PASTE-API besteht keine explizite Möglichkeit, das Server-seitige Beenden der Verbindung festzustellen. Der Client kann nur weiter versuchen Daten zu lesen, bis ein *Timeout* oder ein Fehler auftritt (Zeile 43–45), woraufhin sich auch der Socket-API Client beendet.

5.2 Transportdienstbeschreibungen

Beim Anfordern einer neuen Kommunikationsassoziation werden Transportdienstbeschreibungen ausgetauscht. Sowohl zwischen Anwendungsinstanzen und dem PASTE Rahmenwerk als auch Ende-zu-Ende zwischen den an der Kommunikationsassoziation beteiligten Anwendungsinstanzen. Im Folgenden wird betrachtet wie viel Zusatzaufwand durch den Austausch der Transportdienstbeschreibungen vor dem etablieren einer Transportverbindung entsteht.

Auflistung 21 zeigt zu diesem Zweck erneut die Beispielbeschreibung aus Kapitel 3.5.3, welche beispielsweise von einem Online Spiel verwendet werden könnte. Diese weißt zunächst eine Größe von *670 Byte* auf, wie in Tabelle 5.1 dargestellt ist. Diese Größe lässt sich jedoch deutlich reduzieren. Zunächst können alle Felder aus der Dienstbeschreibung entfernt werden, welche nicht benötigt werden, da diese optional sind. Beispielsweise da sie keine oder nur triviale Informationen enthalten. Dadurch lässt sich die gleiche Dienstbeschreibung bereits kompakter darstellen, wie es in Auflistung 22 dargestellt ist. In dieser Form werden nur noch *563 Byte* benötigt.

Als nächster Schritt kann die Dienstbeschreibung noch weiter minimiert werden, indem sämtliche Leerzeichen und Zeilenumbrüche entfernt werden. Damit lässt sich eine minimale Größe von zunächst *323 Byte* erreichen, wie in Auflistung 23 dargestellt. In dieser Form geht allerdings auch der Vorteil der menschlichen Lesbarkeit der Dienstbeschrei-

Auflistung 21: *Dienstbeschreibung für partielle Zuverlässigkeit.*

```
{
  "description" : {
    "serviceID"  : "nul",
    "templateID" : "nul",
    "mandatory" : [
      {
        "name"  : "qos://transport/data/lifetime",
        "value" : "30",
        "unit"  : "ms"
      },
      {
        "name"  : "qos://transport/reliability",
        "value" : "true",
        "unit"  : "boolean"
      }
    ],
    "optional" : [
      {
        "name"  : "qos://transport/bitrate/maximum",
        "value" : "7.5",
        "unit"  : "kbyte/s"
      }
    ],
    "informational" : [
      {
        "name"  : "qos://transport/umts-like/class",
        "value" : ["interactive"],
        "unit"  : "nul"
      }
    ]
  }
}
```

bung verloren. Soll die Größe weiter reduziert werden, können die aus den letzten drei Schritten erhaltenen Dienstbeschreibungen jeweils noch komprimiert werden. Dadurch lässt sie die Größe in diesem Fall auf *267 Byte* für die ursprüngliche Dienstbeschreibung, *234 Byte* für die kompakte und *198 Byte* für die minimierte Form reduzieren, wie ebenfalls in Tabelle 5.1 dargestellt. Dies führt jedoch zu zusätzlichem Rechenaufwand zum

Auflistung 22: *Dienstbeschreibung für partielle Zuverlässigkeit reduziert auf relevante Informationen.*

```
{
  "description": {
    "mandatory": [{
      "name": "qos://transport/data/lifetime",
      "value": "30",
      "unit": "ms"
    }, {
      "name": "qos://transport/reliability",
      "value": "true"
    }],
    "optional": [{
      "name": "qos://transport/bitrate/maximum",
      "value": "7.5",
      "unit": "kbyte/s"
    }],
    "informational": [{
      "name": "qos://transport/umts-like/class",
      "value": ["interactive"]
    }]
  }
}
```

Packen und Entpacken der Dienstbeschreibungen, welcher sich negativ auf die Verarbeitungszeit der Dienstbeschreibungen auswirkt. Selbst in unkomprimierter Textform, ist das Parsen der Dienstbeschreibungen aufwendiger als bei einem Binärformat. Dies liegt vor allem daran das sämtliche numerischen sowie booleschen Werte mittels einer String-Umwandlung zunächst in die entsprechenden primitiven Datentypen umgewandelt werden müssen. Weiter String-Operationen sind notwendig zur Identifikation von Zeilenenden und Trennzeichen zwischen Schlüssel-Wert-Paaren. String-Operationen sind komplex und erfordern mehr Rechenzeit als das einfache Einlesen eines Binär codierten Datums in einen primitiven Datentyp.

Um die Verarbeitungszeit zu reduzieren und die Transportdienstbeschreibungen maschinell besser verarbeiten zu können, wird für die Implementierung des PASTE Rahmenwerkes daher das Universal Binary JSON Format [Uni16] verwendet. Dieses ist darauf optimiert kompakter als die reine (noch lesbare) Textdarstellung zu sein und bietet eine einfache Transformation zwischen Binär- und Textformat an. So kann der

Auflistung 23: *Dienstbeschreibung für partielle Zuverlässigkeit in minimierter Form.*

```
{"description":{"mandatory":[{"name":"qos://transport/data/life
time","value":"30","unit":"ms"},{"name":"qos://transport/reliab
ility","value":"true"}],"optional":[{"name":"qos://transport/bi
trate/maximum","value":"7.5","unit":"kbyte/s"}],"informational"
:[{"name":"qos://transport/umts-like/class","value":["interacti
ve"]}]}}
```

Anwendungsentwickler bei Bedarf noch von den Vorteilen der Menschen-lesbaren Form profitieren, während eine effiziente Verarbeitung der Transportdienstbeschreibungen durch das PASTE Rahmenwerk gewährleistet ist.

Auflistung 24: *Generische Transportdienstbeschreibung und Abschätzung einer durchschnittlichen Transportdiensteigenschaft.*

```
{
  "description" : {
    "mandatory" : [
    ],
    "optional" : [
    ],
    "informational" : [
    ]
  }
}

{
  "name" : "name://abcdefghijk/lmnopqrst/uvwxyz",
  "value" : "1234.567",
  "unit" : "unit/unit"
}
```

Wie in Tabelle 5.1 zu sehen ist, lässt sich die Transportdienstbeschreibung damit auf 331 Byte reduzieren, welche in diesem Fall beim Assoziationsaufbau übertragen werden müssen. Nimmt man eine *Maximum Segment Size* (MSS) von 1280 Byte an, welche von IPv6 vorgeschrieben ist. Lässt sich die Transportdienstbeschreibung also problemlos innerhalb eines Segmentes übertragen. In Auflistung 24 ist die grundsätzliche Struktur der Transportdienstbeschreibungen und eine Abschätzung einer durchschnittlichen

Transportdiensteigenschaft aufgeführt. Im Universal Binary JSON Format sind diese 83 Byte beziehungsweise 65 Byte groß. Die 83 Byte fallen pro Transportdienstbeschreibung an, während die 65 Byte im Schnitt pro Transportdiensteigenschaft innerhalb der Beschreibung anfallen.

Tabelle 5.1: *Größe der Beispiel Transportdienstbeschreibung in Byte im Vergleich.*

	Text (JSON)	**GZip-Text**	**Binary (UBJSON)**
vollständig	670 Byte	267 Byte	394 Byte
kompakt	563 Byte	234 Byte	**331** Byte
minimiert	323 Byte Byte	198 Byte	**331** Byte

Betrachtet man erneut die MSS von 1280 Byte, müssen von dieser im Fall von IPv6 mindestens 40 Byte für den IPv6 Header abgezogen werden. Dazu kommen 8 Byte für einen UDP Header und 4-32 Byte für den PASTE Header. Insgesamt bleiben also 1200 Byte für Nutzdaten übrig. Von diesen werden noch einmalig 83 Byte für die Transportdienstbeschreibung abgezogen. Damit bleiben 1117 Byte für Transportdiensteigenschaften übrig. Da diese eine Größe von 65 Byte haben, können ausgehend von diesen Werten innerhalb eines Segmentes 17 Transportdiensteigenschaften übertragen werden. Dies ist ausreichend um die Eigenschaften von TCP, UDP oder auch dem in dieser Arbeit verwendetem Dienst für partielle Zuverlässigkeit abzubilden.

5.3 Evaluation des PASTE Prototypen

Im Folgenden wird als Anwendungsfall die Übertragung einer größeren Datenmenge in Form einer großen Datei betrachtet. Die Ausgangssituation ist in Abbildung 5.3 dargestellt. Endsystem E1 sendet an Endsystem E2 Daten. Auf beiden Systemen läuft eine Anwendung welche über TCP eine Verbindung aufbaut und die Daten überträgt.

Insgesamt wurden für die Evaluation drei identisch ausgestattete Testrechner verwendet. Dabei handelt es sich jeweils um:

- Intel(R) Xeon(R) CPU L5420 @2.50GHz
- 16GB RAM (8x 2GiB FB-DIMM DDR2 667 MHz (1.5 ns))
- 64Bit Linux (Ubuntu 15.10)

Die Rechner wurden als Endsystem E1 und E2 sowie als Companion C in den vorgestellten Szenarien verwendet. Endsystem E1 ist dabei mit 1 GBit/s per Ethernet an die anderen Systeme angebunden und fungiert als Dateiserver, im Folgenden auch Server

Seite genannt. Endsystem E2 ist lediglich mit 100 Mbit/s per Ethernet verbunden und fungiert als Client, welcher schwächer angebunden ist als die restlichen Systeme und wird im Folgenden auch als Client Seite bezeichnet. Der Companion C ist ebenfalls über 1 GBit/s Ethernet mit den anderen Systemen verbunden.

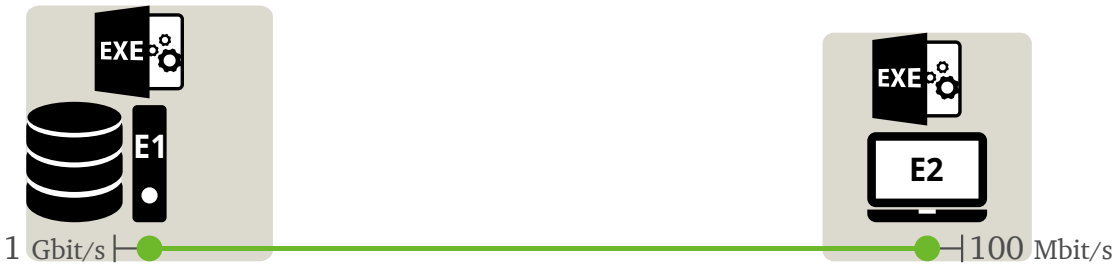


Abbildung 5.3: *Evaluationsszenario: Berkeley Socket-API Aufbau.*

Endsystem E1 ist über den Namen `paste://example.server` erreichbar. Endsystem E2 über den Namen `paste://example.client` und der Companion C über den Namen `paste://example.companion`. Für sämtliche Messungen hat sich Endsystem E1 bereits in einen Empfangsbereiten Zustand versetzt. Im Falle der Berkeley Socket-API also per Bind an einen TCP Server Socket gebunden beziehungsweise im Falle des PASTE Rahmenwerkes per BIND Primitive einen Anwendungsdienst angeboten. Zudem wurden sämtliche Systeme in einen Zustand versetzt, bei dem sich die Namen der jeweils anderen Systeme im Cache befinden, so dass die Namensauflösung beim Verbindungsaufbau in allen Fällen einen identischen Aufwand verursacht, der möglichst gering ist.

Im Folgenden werden drei Szenarien bei denen das PASTE Rahmenwerk und ein Companion zum Einsatz kommt betrachtet. Diese werden jeweils mit einer Datenübertragung mittels der Berkeley Socket-API verglichen:

- Im ersten Fall kommt ein Companion, der einen Zwischenspeicherdienst im Netz anbietet, zum Einsatz.
- Im zweiten Fall kommt ein Companion, der eine Datenkompression der an Endsystem E2 zu sendenden Daten anbietet, zum Einsatz.
- Im letzten Fall kommt ein Companion, der eine Kombination der beiden vorherigen Dienste anbietet, zum Einsatz.

5.3.1 Definition gemessener Größen

Folgende Parameter werden jeweils gesondert betrachtet:

- Der durch den Einsatz des *GoodCompany* Protokolls generierte *Overhead* bei der Datenübertragung, durch die Verwendung von PASTE Sequenznummern und die

zusätzlich benötigte PASTE Signalisierung. Dabei wurden sowohl der *Overhead* in Mbit/s gemessen, als auch der insgesamt angefallene *Overhead* der Datenübertragung in Megabyte.

- Die jeweils benötigte Zeit zum Senden beziehungsweise Empfangen sämtlicher Daten.
- Die benötigte Zeit zur Übertragung der ersten Dateneinheit. Dies umfasst den Verbindungsaufbau durch den Client auf dem Endsystem E2 zum Server auf dem Endsystem E1. Gemessen wurde die Zeit vom initiieren der Verbindung durch den Client bis zum vollständigen Empfang der ersten versendeten Dateneinheit vom Server an den Client.
- Die pro Sekunde erzielte Datenraten in Mbit/s. Dabei wurde auf der Server Seite die Senderate und auf der Client Seite die Empfangsrate gemessen.

Sämtliche Megabyte Angaben sind dem International Electrotechnical Commission (IEC) Standard entsprechend zur Basis 10 berechnet. Das heißt 1 Megabyte entspricht 1000^2 Bytes beziehungsweise 10^6 Bytes.

Für alle drei Szenarien wurden jeweils 21 Durchläufe durchgeführt. In den Diagrammen und Tabellen ist jeweils der Median aller gemessenen Werte abgetragen.

5.3.2 Szenario 1: PASTE Rahmenwerk mit Cache Companion

In Abbildung 5.4 ist der Evaluationsaufbau dargestellt. Zusätzlich zu den zwei Endsystemen E1 und E2 befindet sich ein drittes System C im Netzwerk. Auf diesem läuft eine PASTE Instanz, welche unterschiedliche Companiondienste anbietet. In diesem Fall einen Zwischenspeicherdienst um Daten im Netzwerk zu puffern. Für alle folgenden Szenarien ist der Companion ebenfalls mit 1 Gbit/s angebunden. Auf den beiden Endsystemen läuft ebenfalls jeweils eine PASTE Instanz. Es wird die gleiche Client und Server Anwendung wie in Abbildung 5.3 verwendet. Allerdings wurde die Berkeley Socket-API ausgetauscht durch die PASTE-API ersetzt. Die Anwendungsinstanzen kommunizieren nicht direkt über TCP miteinander, sondern via Interprozess Kommunikation mit dem PASTE Rahmenwerk, welches sich um die Datenübertragung zwischen den Systemen kümmert.

Statt einer vollständigen Dienstbeschreibung, verwenden die Anwendungsinstanzen beim Verbindungsaufbau einen Transportdienstidentifikator. Es wird also die verkürzte Variante des Kommunikationsassoziationsaufbaus verwendet, wie er in Kapitel 4.5.4 beschrieben ist. Damit kann demnach das Minimum, also der mindestens anfallende zusätzliche Aufwand, beim Assoziationsaufbau bestimmt werden. Client und Server fordern dabei jeweils einen zuverlässigen Transportdienst an, welcher in allen Szenarien durch das TCP Protokoll erbracht wird.

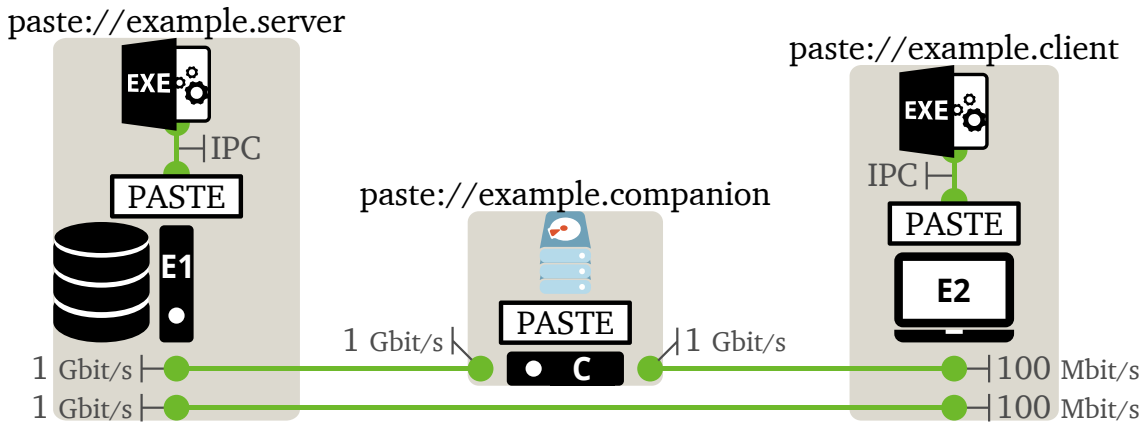


Abbildung 5.4: *Evaluationsszenario: Datenzwischenspeicher Aufbau.*

Das PASTE Rahmenwerk ist auf allen drei Systemen bereits gestartet und sowohl der Server als auch der Companion sind initial in einem Empfangsbereiten Zustand. Wenn der Client gestartet wird, fordert dieser per CONNECT Primitive den Aufbau einer Kommunikationsassoziation zum Server an. Nach der Etablierung der Assoziation, beginnt der Server die Datenübertragung zum Client. In allen Szenarien wurden genau 500 MB Nutzdaten vom Server an den Client übertragen. Anschließend beenden sich beide Anwendungsinstanzen. Für die Übertragung werden beim Einsatz eines Companions PASTE Sequenznummern verwendet, um die Reihenfolgetreue des angeforderten Transportdienstes zu gewährleisten.

Der Server versendet grundsätzlich 1000 Byte große Dateneinheiten. Diese überschreiten also nicht die Größe einer MSS und müssen nicht fragmentiert werden. Durch das PASTE Rahmenwerk werden die Anwendungsdateneinheiten grundsätzlich erhalten und in der gleichen Granularität in der sie versendet wurden an das Zielsystem gereicht. Für das TCP/Socket-API Szenario, wird regelmäßig geprüft wie viel Bytes lesbar sind und diese in 1000 Byte Datenblöcken auf der Empfängerseite aus dem Datenstrom entnommen. Im Falle des PASTE Rahmenwerks kommt zusätzlich zu den 1000 Byte der PASTE Header hinzu, sowie die GoodCompany Signalisierungsnachrichten, welche in allen Szenarien *inband* über die TCP Verbindungen übertragen werden.

Der zusätzliche Aufwand durch TCP und IP Header ist damit sowohl für die Berkeley Socket-API als auch die PASTE-API identisch und wird nicht betrachtet. Vielmehr wird die Datenübertragung per Socket-API als Basis für alle weiteren Betrachtungen herangezogen.

Abbildung 5.5 zeigt die Client Seite und stellt die Datenempfangsrate in Mbit/s dar. Hierzu wurde die pro Sekunde übertragene Datenmenge gemessen. Die Zeit in Sekunden ist auf der X-Achse des Diagramms abgetragen, während auf der Y-Achse die jeweils übertragene Datenmenge in Mbit abgetragen ist. Dabei ist zu beachten, dass für die Y-Achse

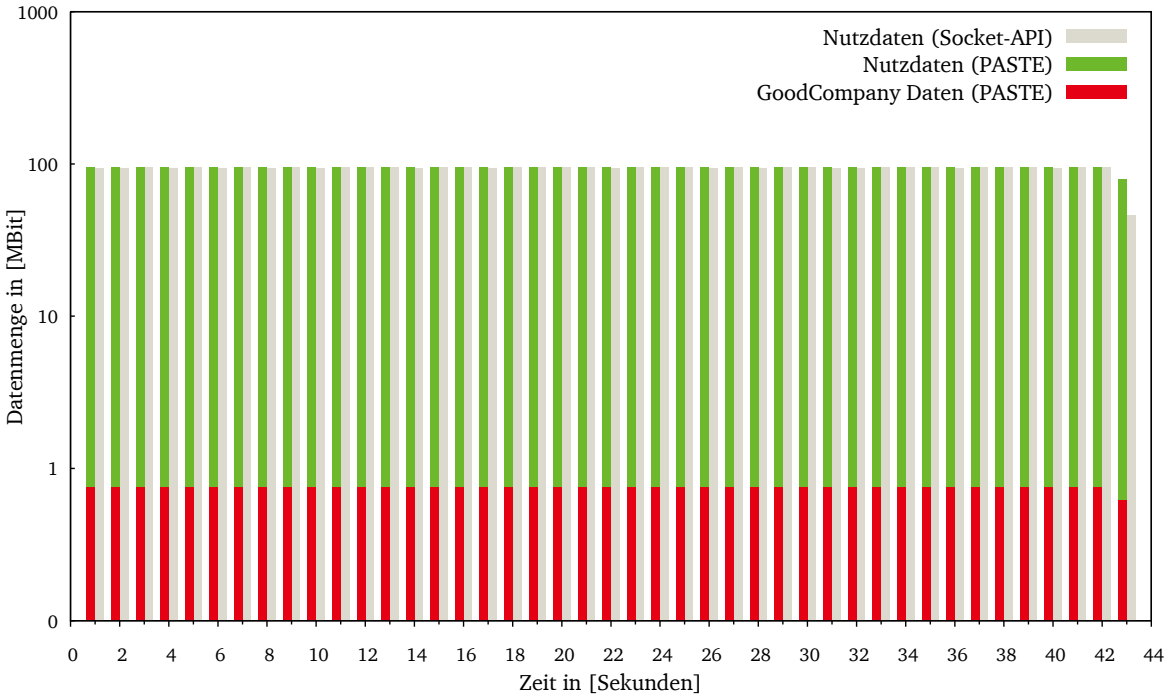


Abbildung 5.5: *Datenempfangsrate & -dauer mit Datenzwischenspeicher.*

eine logarithmische und für die X-Achse eine lineare Skalierung gewählt wurde. Pro Messintervall (also einmal pro Sekunde) ist die per Socket-API übertragene Datenmenge (in beige) gegenüber der mit Hilfe des PASTE Rahmenwerkes übertragenen Datenmenge abgetragen. Diese ist aufgeteilt in Nutzdaten (grün) und *GoodCompany* Signalisierungsdaten (rot). In beiden Fällen wird eine Auslastung von nahezu 100 Mbit/s erreicht, wobei der durch das PASTE Rahmenwerk erzeugte Zusatzaufwand konstant unter 1 Mbit/s also unter 1% liegt. Da im Falle des PASTE Rahmenwerks mehr Daten übertragen werden, ist der Balken im letzten Intervall in Sekunde 43 größer als im Falle der Socket-API. In beiden Fällen ist er kleiner als 100 Mbit, da keine vollen 100 Mbit mehr zu übertragen sind.

Auf der Client Seite ergibt sich dadurch für das PASTE Rahmenwerk eine leicht gestiegene Datenübertragungszeit ohne Vorteile gegenüber der Socket-API. Der PASTE Client profitiert in diesem Szenario also nicht vom Einsatz des Companions. Anders sieht es auf der Server Seite aus, welche in Abbildung 5.6 zu sehen ist. Statt der empfangenen Daten sind in diesem Diagramm die vom jeweils vom Server versendeten Daten abgetragen. Limitiert durch die Anbindung des Clients mit 100 Mbit/s, kann der Socket-API Server in diesem Szenario seine Anbindung von 1 GBit/s nicht ausnutzen. Die erreichte Datenrate liegt im Durchschnitt ebenfalls nur bei 100 Mbit/s. Im PASTE Fall fordert der Server daher nach 1 Sekunde einen Companion mit Zwischenspeicherdienst an. Nachdem dieser in die

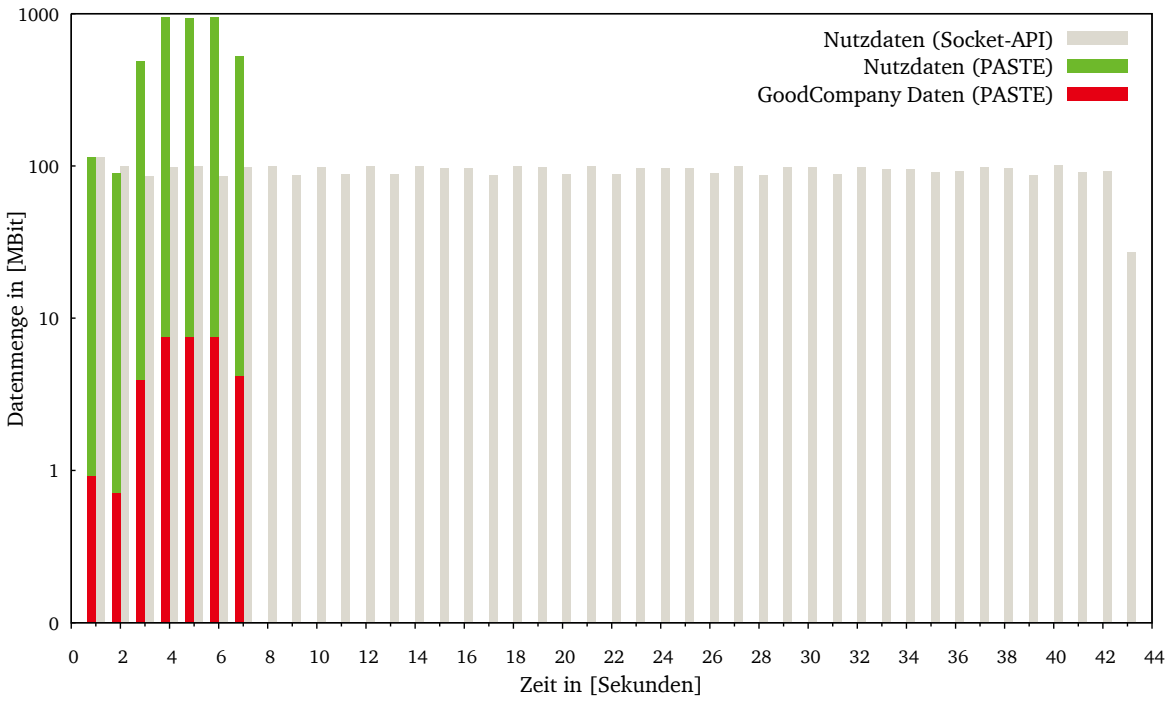


Abbildung 5.6: *Datensenderate & -dauer mit Datenzwischenspeicher.*

Kommunikationsassoziation eingefügt wurde, werden nun die Daten vom PASTE Server an den Companion übertragen. Dieser überträgt die Daten dann weiter an den Client. Dadurch kann der PASTE Server den Datentransfer bereits nach 7 Sekunden abschließen, gegenüber dem Socket-API Server, welcher fast 43 Sekunden benötigt. Auch durch den PASTE Server kann somit durch Einsatz des Companions der zur Verfügung stehende Anschluss ausgelastet werden.

Tabelle 5.2: *Gemessene Werte für das Szenario: PASTE Rahmenwerk mit Cache Companion*

Zeit	**Socket-API**	**PASTE**
Erste Dateneinheit	0,7612 ms	0,8756 ms
Senden	42,262890 s	6,561505 s
Empfangen	42,488655 s	42,828640 s
Datenmenge		
Gesendet	500 MB	504,000080 MB
Empfangen	500 MB	504,000016 MB

Tabelle 5.2 zeigt die exakten gemessenen Werte. Die insgesamt zusätzlich über die verwendeten TCP Verbindungen übertragenen Daten des PASTE Rahmenwerks durch

das *GoodCompany* Protokoll betragen 4,000080 MB, was einen Mehraufwand von ca. 0,8% bedeutet. Die vom PASTE Client empfangenen Daten sind etwas geringer, da weniger Signalisierungsnachrichten zwischen Companion und PASTE Client ausgetauscht werden. Zudem wurde die benötigte Dauer zur Übertragung der ersten Dateneinheit gemessen. Diese beträgt im Falle der Socket-API 0,7612 ms und im Falle des PASTE Rahmenwerks 0,8756 ms. Dies entspricht einem Anstieg um etwas mehr als 0,11 ms, was ungefähr einer halben Paketumlaufzeit (RTT) entspricht. Diese liegt zwischen Client und Server über den Zeitraum der Datenübertragung gemessen im Durchschnitt bei 0,261 ms. Der zusätzliche Aufwand wird in diesem Fall verursacht durch das initiale Übertragen des Assoziationszustandes beim Anfordern der Kommunikationsassoziation. Da die verkürzte Variante des Kommunikationsassoziationsaufbaus verwendet wird, wie in Abbildung 4.5.4 dargestellt, wird dazu vom Client eine `association_req` (mit einer Größe von 40 Byte) und eine `new_channel_ind` Nachricht (mit einer Größe von 20 Byte) verschickt. Da die Transportverbindung bereits parallel zur `association_req` Nachricht etabliert werden kann und direkt nach der Etablierung die `new_channel_ind` Nachricht verschickt wird, kann der Server direkt nach Erhalt der `new_channel_ind` Nachricht mit einer `new_channel_conf` Nachricht (mit einer Größe von 4 Byte) gefolgt von dem ersten Datenpaket antworten. Dies entspricht einer Verzögerung von weniger als einer Paketumlaufzeit, da der Client in diesem Fall zusätzlich zum TCP Verbindungsaufbau nur einmalig 60 Byte übertragen muss bevor der Server mit dem ersten Datenpaket + 4 Byte zusätzlichem *Overhead* antwortet.

Die Unterschiede zwischen Sende- und Empfangsdauer bei der Verwendung der Socket-API lassen sich durch leicht schwankende Paketverarbeitungszeiten auf dem Client und dem Server erklären. Diese sind auf Empfängerseite mitunter größer, da eingehende Pakete nicht sofort sondern erst nach kurzer Wartezeit verarbeitet werden. Dies passiert sofern andere Prozesse beim Eintreffen eines Paketes gerade auf den Speicher oder die Festplatten zugreifen und somit I/O Last produzieren. Dies lässt sich nie vollständig verhindern, da im Betriebssystem zu jedem Zeitpunkt diverse Hintergrundprozesse laufen, selbst wenn keine weitere Anwendung ausgeführt wird.

5.3.3 Szenario 2: PASTE Rahmenwerk mit Compress Companion

In Abbildung 5.7 ist der Evaluationsaufbau analog zum 1. Szenario dargestellt. Statt des Zwischenspeicherdienstes kommt in diesem Szenario aber ein Datenkompressionsdienst zum Einsatz. Dazu müssen Companiondienstkomponenten auf der Client Seite und dem Companion selbst instanziiert werden. Der restliche Aufbau ist identisch.

In diesem Szenario fordert der PASTE Client direkt nach dem Assoziationsaufbau einen Companion zur Datenkompression an, um die zu empfangende Datenmenge zu reduzieren. Abbildung 5.8 zeigt analog zum ersten Szenario die Client Seite des Datentransfers.

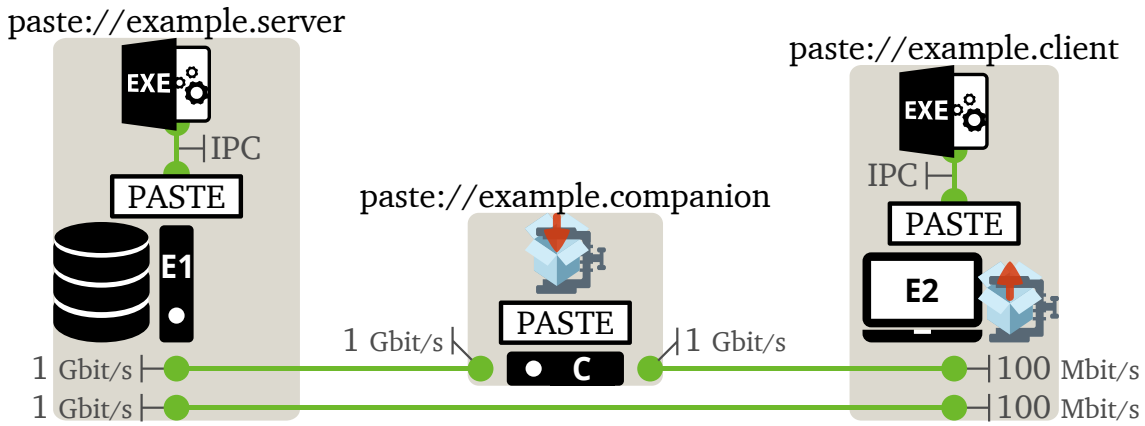


Abbildung 5.7: *Evaluationsszenario: Datenkompression Aufbau.*

Erneut wird eine Auslastung von nahezu 100 Mbit/s erreicht. Die zusätzliche Datenverarbeitung in Form von Kompression beziehungsweise Dekompression der versendeten Daten hat also keinen messbaren Einfluss auf die erzielte Datenrate. Da insgesamt aber weniger Daten zwischen Companion und PASTE Client übertragen werden müssen, hat der PASTE Client gegenüber dem Socket-API Client sämtliche Daten bereits nach 28 statt 43 Sekunden erhalten. Da der Companiondienst allerdings nur den Nutzdatenteil der Dateneinheiten komprimiert steigt der relative Anteil an Signalisierungsdaten auf im Durchschnitt 1.2 Mbit/s was 1.2% entspricht

Auf der Server Seite zeigt sich ein ähnliches Bild wie in Abbildung 5.9 zu sehen ist. Zwar wird die erreichbare Datenrate des PASTE Servers noch immer durch den Client limitiert, dadurch dass der Companionen allerdings insgesamt eine geringere Datenmenge an den PASTE Client übertragen muss, kann auch der PASTE Server eine höhere Datenrate erreichen. Diese liegt in diesem Szenario im Durchschnitt bei 148 Mbit/s. Dadurch kann auch der PASTE Server den Datentransfer bereits in 28 Sekunden abschließen, gegenüber der Socket-API welche erneut 43 Sekunden benötigt.

In diesem Szenario profitieren also sowohl der PASTE Client als auch der PASTE Server vom Companion Einsatz. Tabelle 5.3 zeigt erneut die exakten gemessenen Werte. Die insgesamt zusätzlich übertragenen Daten des PASTE Rahmenwerks betragen hier 4,000004 MB. Dies ist etwas geringer als im vorherigen Szenario, da hier der Companion Einsatz durch den PASTE Client angestoßen wird. Für die benötigte Dauer zur Übertragung der ersten Dateneinheit ergeben sich ähnliche Werte wie im ersten Szenario, was die dort angestellten Überlegungen bestätigt. In beiden Fällen wird die erste Dateneinheit noch direkt zwischen den Endsystemen ausgetauscht, bevor die Companion Signalisierung abgeschlossen ist. Die durch den Client empfangenen Daten wurden jedoch auf 316,120956 MB reduziert, was einer Kompression der übertragenen Datenmenge auf knapp 63% ihrer ursprünglichen Größe entspricht. Die verwendete Implementierung benutzt zu

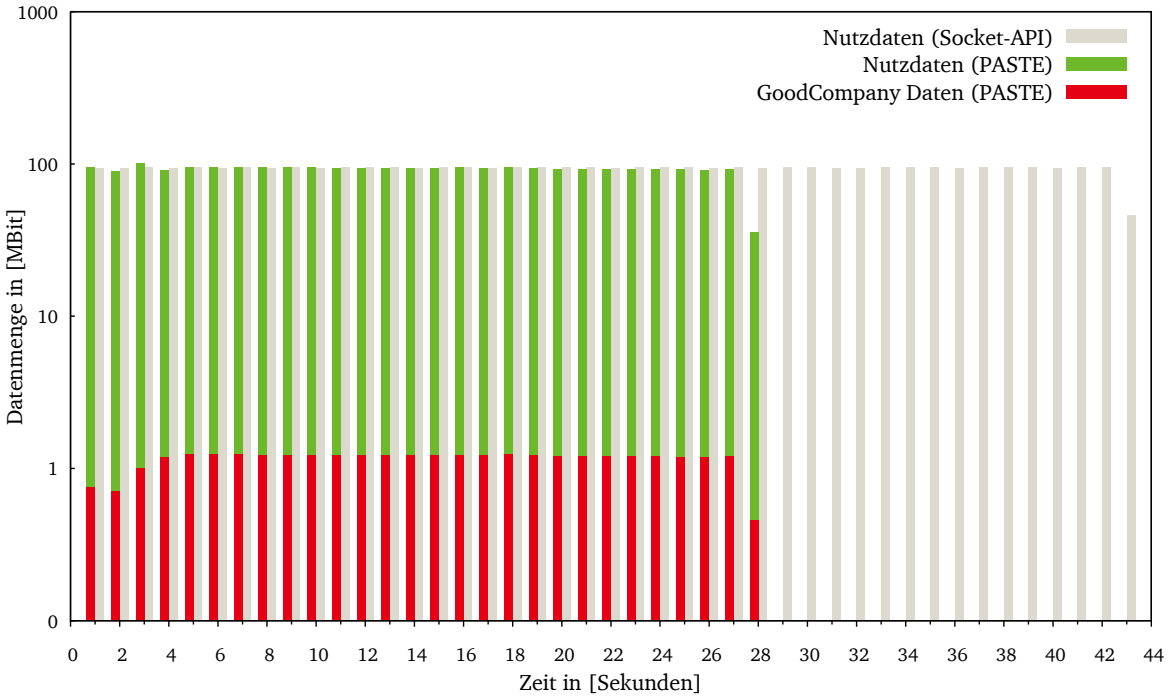


Abbildung 5.8: *Datenempfangsrate & -dauer mit Datenkompression.*

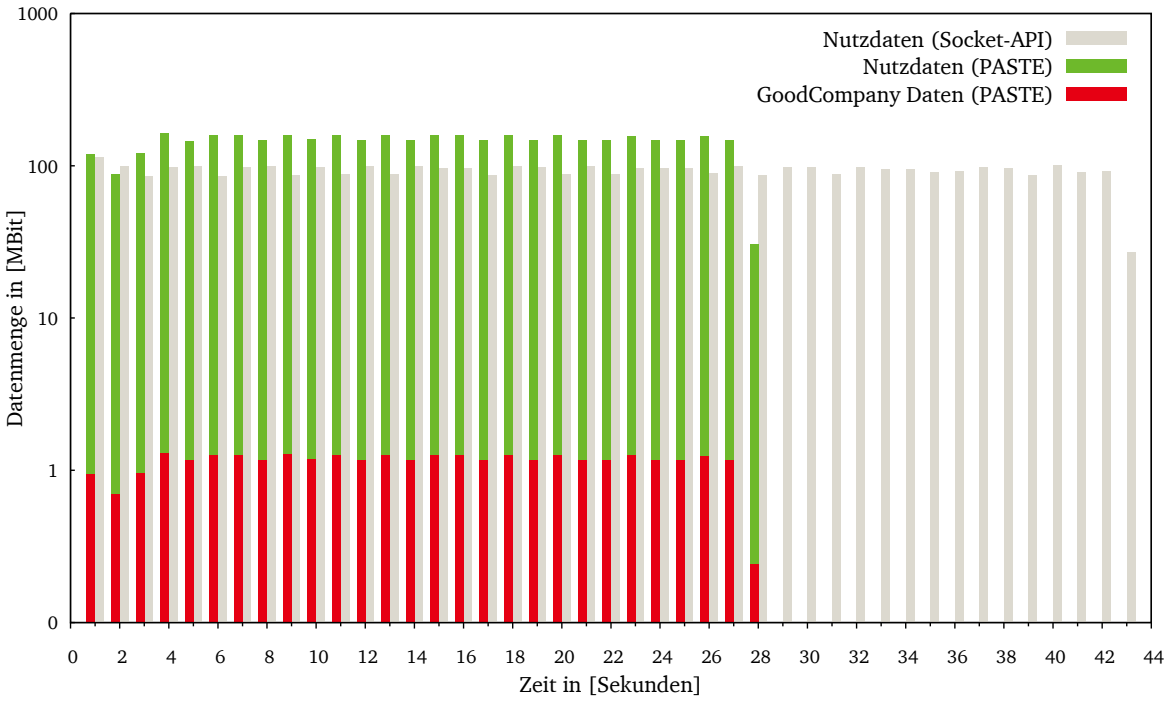


Abbildung 5.9: *Datensenderate & -dauer mit Datenkompression.*

Tabelle 5.3: *Gemessene Werte für das Szenario: PASTE Rahmenwerk mit Compress Companion*

Zeit	**Socket-API**	**PASTE**
Erste Dateneinheit	0,7612 ms	0,8704 ms
Senden	42,262890 s	27,191907 s
Empfangen	42,488655 s	27,382554 s
Datenmenge		
Gesendet	500 MB	504,000004 MB
Empfangen	500 MB	316,120956 MB

diesem Zweck eine einfach GZip Komprimierung der Nutzdaten pro beim Companion eingehendem Paket.

5.3.4 Szenario 3: PASTE Rahmenwerk mit Cache & Compress Companion

Für das letzte in Abbildung 5.10 dargestellte Szenario wurde ein Companion verwendet, welcher beide vorherigen Companiondienste gleichzeitig anbietet. Analog zu Szenario 1 und Szenario 2 fordern Endsystem E1 einen Zwischenspeicherdienst und Endsystem E2 einen Datenkompressionsdienst beim Companion Knoten an.

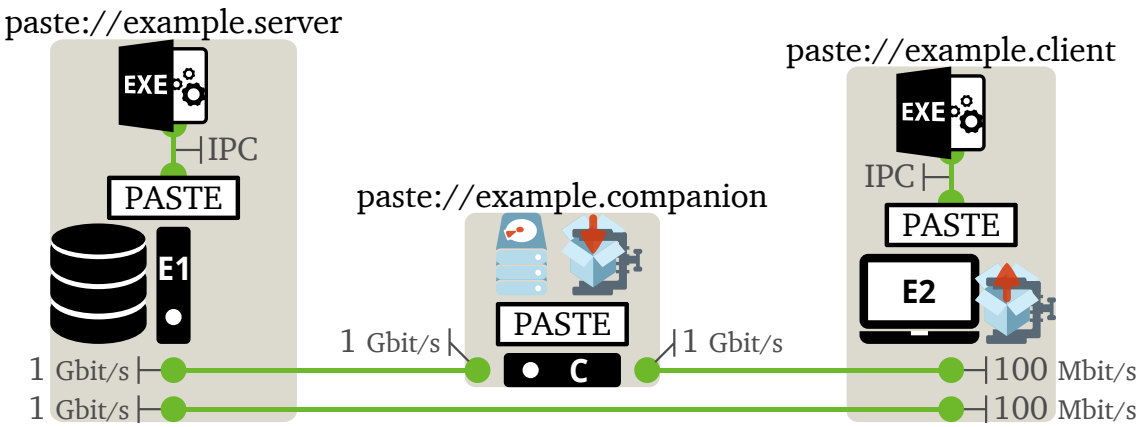


Abbildung 5.10: *Evaluationsszenario: Datenzwischenspeicher und -kompression Aufbau.*

Wie zu erwarten ist zeigt Abbildung 5.11, dass der PASTE Server durch den Zwischenspeicherdienst eine ähnliche Datenrate und damit kürzere Übertragungszeit wie zuvor erreicht. Dabei ist keine messbarer Verbesserung durch den zusätzlichen Einsatz des Kompressionsdienstes auf der PASTE Server Seite feststellbar. Dies liegt daran, dass trotz

der Kompression die Anbindung des PASTE Clients immer noch der limitierende Faktor ist, wie in den beiden vorherigen Szenarien auch.

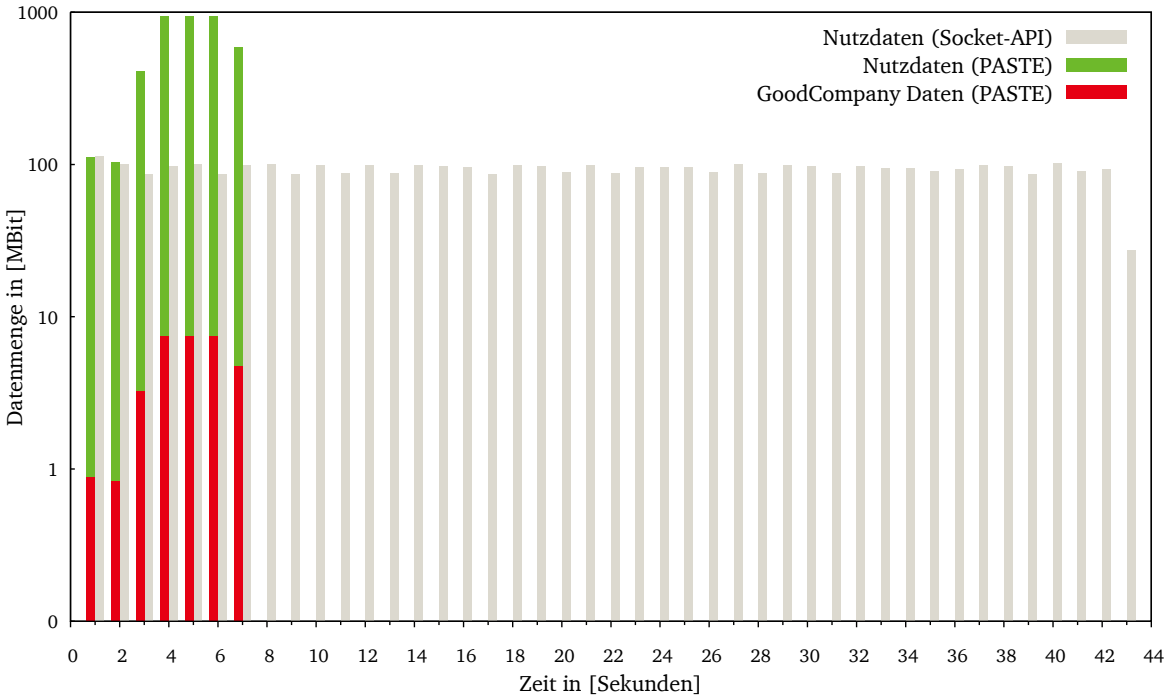


Abbildung 5.11: *Datensenderate/-dauer mit Datenzwischenspeicher & -kompression.*

Ein entsprechendes Bild zeigt sich auf der Clientseite, wie in Abbildung 5.12 zu sehen ist. Zwar erhält der Companion in diesem Szenario die zu übertragenden Daten schneller, jedoch benötigt die Übertragung sämtlicher Daten vom Companion zum PASTE Client genauso lange wie im 2. Szenario.

Der Vollständigkeit halber sind auch für das letzte Szenario noch einmal die gemessen Werte in Tabelle 5.4 aufgeführt. In allen drei Szenarien ließ sich für die verwendete Hardware keine signifikante zusätzliche Latenz durch die Interprozess Kommunikation zwischen Anwendung und PASTE Rahmenwerk feststellen. In einer auf Performanz ausgelegten Implementierung des PASTE Rahmenwerkes, sollte die Interprozess Kommunikation dennoch durch shared-memory ersetzt werden, um zwischen Anwendungen, PASTE Rahmenwerk und dem Netzwerkstapel des Betriebssystems keine Kopien der zu versendenden ADUs zu benötigen.

Die durch den Assoziationsaufbau zusätzlich eingeführte Latenz ist abhängig von der übergebenen Transportdienstbeschreibung. Ist diese größer, müssen initial einmalig mehr Daten übertragen werden, bevor eine Kommunikationsassoziation aufgebaut werden kann. Wie in Abschnitt 5.2 gezeigt wurde, lassen sich jedoch selbst größere Transportdienstbeschreibung innerhalb eines TCP-Segments versenden. Sofern keine Parameter

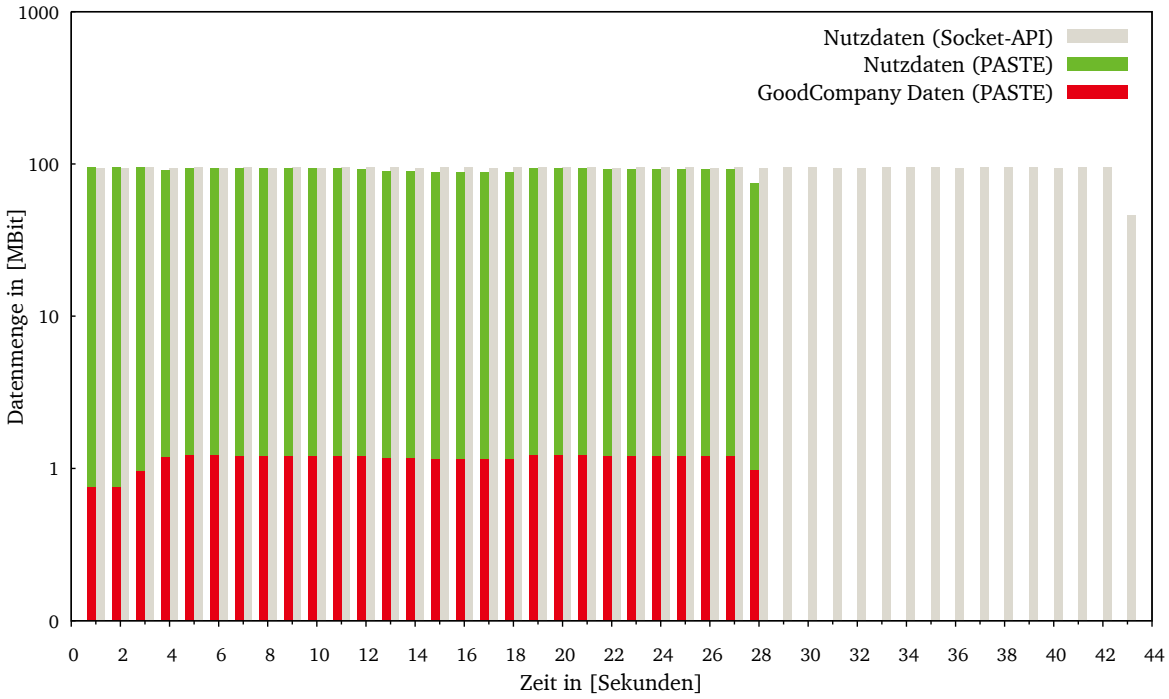


Abbildung 5.12: *Datenempfangsrate/-dauer mit Datenzwischenspeicher & -kompression.*

Tabelle 5.4: *Gemessene Werte für das Szenario: PASTE Rahmenwerk mit Cache & Compress Companion*

Zeit	**Socket-API**	**PASTE**
Erste Dateneinheit	0,7612 ms	0,8746 ms
Senden	42,262890 s	6,630826 s
Empfangen	42,488655 s	27,808577 s
Datenmenge		
Gesendet	500 MB	504,000084 MB
Empfangen	500 MB	316,121020 MB

zwischen den Endsystemen ausgehandelt werden müssen, beträgt die zusätzliche Latenz in der Regel weniger als eine halbe Paketumlaufzeit. Trotz des durch das PASTE Rahmenwerk eingeführten Zusatzaufwandes, konnte für die drei Szenarien ein deutlicher Leistungsgewinn für die Parameter *Datenübertragungsdauer*, *übertragene Datenmenge* und *Auslastung der verfügbaren Datenrate* erzielt werden, so dass die zusätzlich zu übertragenden Signalisierungsdaten und die höhere Latenz beim Assoziationsaufbau je nach Szenario und Client beziehungsweise Server Seite nicht ins Gewicht fallen.

5.4 Zusammenfassung

In diesem Kapitel wurde zunächst die Anwendbarkeit der PASTE-API für andere Kommunikationsparadigmen anhand verschiedener Arbeiten aufgezeigt. Dabei wurde die Nutzbarkeit der PASTE-API für *Content Centric Networks*, *publish-subscribe* basierte Netzwerke und innerhalb eines Demonstrators für das NENA Rahmenwerk dargelegt. Da die PASTE-API Protokoll-agnostische Dienstbeschreibungen verwendet und die Verfügbaren Dienstprimitive vom Anwendungsdienst mit beliebigen Transportdiensten kombiniert werden können, ist die PASTE-API flexibel zur Umsetzung verschiedenster Kommunikationsmuster und -paradigmen geeignet. Durch den Einsatz von URIs und von Anwendungen frei spezifizierbaren Parametern für angebotene Anwendungsdienste kann die PASTE-API flexibel an die Bedürfnisse einer Anwendung angepasst werden. Zudem wurde ein Vergleich zwischen minimalen Beispielimplementierungen jeweils eines Clients und eines Servers mit Hilfe der PASTE-API und Berkeley Socket-API durchgeführt.

Anschließend wurde eine Abschätzung für die durchschnittliche Größe von Transportdienstbeschreibungen angestellt. Zuletzt wurde mit Hilfe einer prototypischen Implementierung des PASTE Rahmenwerkes ein einfacher Companion Anwendungsfall für drei verschiedene Szenarien evaluiert. Dabei konnte basierend auf der Abschätzung der Transportdienstbeschreibungen und der gemessenen Datenübertragungsraten, -zeiten und -mengen gezeigt werden, dass sich für alle drei Parameter ein Leistungszuwachs erzielen lässt, der jeweils über dem durch das PASTE Rahmenwerk eingeführten Zusatzaufwand für die einzelnen Größen liegt.

Kapitel 6

Zusammenfassung

Eingangs wurde motiviert, inwiefern im heutigen Internet Innovationen vor allem auf den unteren Schichten des ISO/OSI Referenzmodells und oberhalb der Transportschicht beziehungsweise innerhalb der Anwendungsschicht stattfinden. Gleichzeitig sind Innovationen auf den Schichten 3 und 4 kaum möglich, unter anderem durch die weite Verbreitung der Berkeley Socket-API als De-Facto-Standard für die Kommunikation innerhalb von verteilten Anwendungen und die damit verbunden starre Protokollwahl auf der Anwendungsebene zur Designzeit einer Anwendung.

Zur Lösung dieses Problems schlägt diese Arbeit die PASTE-API vor, eine Protokollagnostische Schnittstelle zwischen Anwendung und Netzwerk. Die PASTE-API ermöglicht es Anwendungsentwicklern Datentransportdienste anzufordern, welche erst zur Laufzeit der Anwendung auf ein passendes Transportprotokoll abgebildet werden. Zusätzlich bietet sie Anwendungsentwicklern Mechanismen zur flexiblen Umsetzung und Nutzung unterschiedlichster Kommunikationsmuster und -paradigmen an und fördert damit die automatische Nutzung von im Netzwerk vorhandenen Protokollen und Ressourcen ohne zusätzliches Wissen vom Anwendungsentwickler zu benötigen. Damit werden auch für *Internet Service Provider* Anreize geschaffen neue Protokolle auszubringen, um Anwendungen besser zu unterstützen und gleichzeitig ihre Infrastruktur durch die Wahl passender Protokolle effizienter ausnutzen zu können.

Ein hierarchisches Transportdienstbeschreibungssystem für die PASTE-API wurde entwickelt, um Anwendungsentwicklern die Möglichkeit zu geben gewünschte Datentransportdiensteigenschaften zu formulieren und vom Netzwerk anzufordern. Gleichzeitig wurde das System so entworfen, dass es für häufig genutzte Datentransportdienste einen Möglichst geringen Zusatzaufwand gegenüber der heute verwendeten Berkeley Socket-API beim Anfordern von Kommunikationsassoziationen einführt.

Zur Umsetzung der PASTE-API wurde das Protocol Agnostic Services & Transport Enrichment (PASTE) Rahmenwerk entworfen. Es ermöglicht die Realisierung Protokollagnostischer Transportdienste. Außerdem übernimmt es Aufgaben der Sitzungsschicht, welche heute häufig auf der Anwendungsebene umgesetzt werden. Insbesondere die Verwaltung von pausierbaren Kommunikationsassoziationen für die Anwendung. Zusätzlich kümmert sich das PASTE Rahmenwerk um die Namensauflösung. Schließlich werden erweiterte Transportdienste durch die Nutzung von Zwischensystemen im Netz durch das PASTE Rahmenwerk ermöglicht. Dazu wurden *Companions* eingeführt, bei denen es sich um im Netz verortet Zwischensysteme handelt auf denen das PASTE Rahmenwerk für Endsysteme verschiedene Netzwerkdienste anbietet. Im Rahmen der Evaluierung wurde gezeigt, wie Anwendungen von diesen Diensten profitieren können und die Funktionalität des PASTE Rahmenwerkes und der PASTE-API gezeigt.

6.1 Ausblick und weiterführende Arbeiten

Sowohl die PASTE-API als auch das PASTE Rahmenwerk bieten viel Spielraum für weiterführende Arbeiten. Sinnvolle Ergänzungen der PASTE-API umfassen:

- Eine Integration von Kommunikationsmustern direkt in die PASTE-API. In der in dieser Arbeit vorgestellten Form ist die PASTE-API auf die Kommunikation zwischen jeweils zwei Endpunkten ausgelegt. In Zukunft sollte die PASTE-API auch 1-zu-n und n-zu-n Kommunikation zwischen Gruppen von Endsystemen ermöglichen. Dazu ist mindestens eine Erweiterung zur Gruppenverwaltung auf der Anwendungsebene nötig. Des Weiteren muss untersucht werden, inwiefern Kommunikationsmuster wie beispielsweise *Multicast* oder *Concast* unterhalb der PASTE-API transparent realisiert werden können, und welche Mechanismen über die PASTE-API an die Anwendungsschicht durchgereicht werden müssen. Im Zusammenhang mit dem PASTE Rahmenwerk ist zudem interessant, inwiefern *Companions* hierbei zur Unterstützung im Netz eingesetzt werden können.

- Die für URIs spezifizierbaren Parameter sind im Rahmen dieser Arbeit sehr einfach gehalten, da es im Web Kontext bereits viele jedoch auch sehr komplexe Beschreibungssprachen gibt. Unter anderem existieren mit RAML (RESTful API Modeling Language) [The] und der OpenAPI Specification [Lin] zwei Lösungen zur Spezifizierung von Schnittstellen für Webdienste. Beide bieten nicht nur die Möglichkeit Typen zu definieren. Zusätzlich können auch Ressourcen und auf diesen operierende Methoden modelliert werden, sowie Anfragen an einen Dienst und auf die Anfrage zu erwartende Antworten definieren. Beide sind auf die Verwendung

mit HTTP ausgelegt, es sollte aber untersucht werden, welche der Konzepte sich verallgemeinern und in die PASTE-API integrieren lassen.

- Eine Standardisierung sämtlicher Transportdiensteigenschaften müsste durchgeführt werden, mit dem Ziel einen vollständigen Transportdiensteigenschaftenkatalog zu erstellen. In der Praxis wird dieser nie vollständig sein, jedoch existieren einige Ansatzpunkte um einen solchen Katalog zu erstellen. Eine Basis könnten gängige *Quality of Service* Parameter sein, welche durch Architekturen wie *Integrated Services* und *Differentiated Services* angeboten werden. Ein anderer Ansatzpunkt ist das Sammeln von Diensteigenschaften heute existierende Protokolle. In diesem Zusammenhang existiert mit der *IETF Working Group on Transport Services (taps)* [FTK16; WTK16] bereits eine Initiative, die es sich zum Ziel gesetzt hat für existierende von der IETF spezifizierte Transportprotokolle eine Liste von Transportdiensten und Transportdiensteigenschaften zusammenzutragen.

Auch für das PASTE Rahmenwerk bieten sich Erweiterungsmöglichkeiten an:

- Neuere Entwicklungen in den Bereichen *Software Defined Networking (SDN)* und *Network Function Virtualization (NFV)* bieten die Möglichkeit zukünftige Netze flexibler zu gestalten und konfigurieren. Auch Dienste im Netzwerk sollen sich darüber einfacher anbieten lassen. NFV zielt dabei aber bisher vor allem auf Dienste ab, welche vom Netzwerkbetreiber im Netz benötigt werden. Beispielsweise *Loadbalancer* oder *Firewalls*. Es sollte untersucht werden, inwiefern sich NFV und SDN einsetzten lassen um Companiondienste im Netzwerk umzusetzen. Es existieren bereits Bestrebungen unterschiedlichste Technologien und Schnittstellen innerhalb von Netzwerken zu öffnen und zu standardisieren, wie zum Beispiel in „Open Network Interfaces for Carrier Networks“ [Pan+16]. Deshalb sollte untersucht werden, ob von Anwendungen angeforderte Transportdienste sich besser umsetzten lassen, indem das PASTE Rahmenwerk über beispielsweise SDN realisiert werden. Einfache Companiondienste lassen sich möglicherweise direkt mithilfe von SDN Anwendungen umsetzen, deren Instanziierung vom PASTE Rahmenwerk angestoßen werden. Dadurch würde es als Brücke zwischen Anwendung und dem Netzwerk fungieren und den Anwendungsentwicklern mehr Kontrolle über die Konfiguration des Netzwerks durch eine Anwendung anhand von Dienstanforderungen geben. Des Weiteren könnte es sinnvoll sein, Teile des *GoodCompany* Protokolls beziehungsweise der dadurch verursachten Signalisierungsdaten in die *Control Plane* zu verlagern. Insbesondere die (regelmäßige) Kommunikation zwischen einzelnen PASTE Instanzen untereinander zum Austausch von Zustandsinformationen lässt sich mithilfe von SDN möglicherweise so realisieren, dass die Daten nur im Netzwerk verteilt werden, sofern es nicht anderweitig ausgelastet ist und somit die

eigentlichen Datenübertragungen weniger gestört werden. Ebenso sind dedizierte Signalisierungskanäle denkbar, welche sich bei Bedarf flexibel durch das PASTE Rahmenwerk im Netzwerk auf- und abbauen lassen.

- Es existieren diverse Arbeiten im Bereich des mobilen Internets, welche sich mit *Vertical Handoff* Lösungen [Che+06; CLC08; Che+05] befassen. Darunter versteht man die schichtenübergreifende Signalisierung von bevorstehenden Verbindungsabbrüchen oder Netzdomänenwechseln auf einem Knoten. Diese Lösungen könnten in das PASTE Rahmenwerk integriert werden, um automatische *RESUME* und *suspend* Anfragen beziehungsweise Ereignisse auszulösen und effizienter auf Verbindungsabbrüche zu reagieren.

- In dieser Arbeit wurde nicht betrachtet, inwiefern sich der Zugriff auf Companions und die Nutzung von Companiondiensten regulieren und absichern lässt. Ein vielversprechender Ansatz dazu existiert mit „Multi Context TLS“ [Nay+15]. Es könnte eingesetzt werden, damit Companions sich gegenüber Endsystemen authentifizieren können und um die Kommunikation zwischen Endsystemen über Companions hinweg abzusichern.

Glossar

Anwendung Eine (verteilte) Anwendung besteht aus einer oder mehreren Anwendungsinstanzen, welche auf Endsystemen ausgeführt werden und einen Anwendungsdienst erbringen.

Anwendungsdienst Anwendungsdienste sind alle durch eine Anwendung angebotenen Netzwerkdienste. Sie befinden sich oberhalb der Schnittstelle zwischen Anwendung und Netzwerk und werden durch die konkreten Anwendungsinstanzen einer Anwendung realisiert.

Anwendungsinstanz Anwendungsinstanzen sind auf Endsystemen ausgeführte Prozesse. Zwischen diesen findet der eigentliche Datenaustausch über mindestens eine auf dem Endsystem vorhandene Netzwerkschnittstelle statt.

Anwendungsprozess siehe Anwendungsinstanz

Dienstanbieter Ein Dienstanbieter stellt einen Anwendungsdienst bereit. Dienstanbieter können dabei Organisationen oder Firmen aber auch Privatpersonen sein.

Endsystem Endsysteme sind alle netzwerkfähigen Geräte (beispielsweise Smartphones, Tablets, PCs, Laptops, Server), welche über eine Kommunikationsbeziehung miteinander Daten austauschen. Dabei ist es unerheblich ob es sich dabei um physikalische oder virtuelle Geräte handelt oder ob diese sich am Netzrand oder im Kern eines Netzes befinden. Verschiedene Systeme können immer nur ein Endsystem in Bezug auf eine oder mehrere Kommunikationsbeziehungen sein.

Internetdienstanbieter Ein Internetdienstanbieter stellt seinen Kunden einen Internetanschluss und damit einen Zugang zum Internet zur Verfügung. (Auch ISP, Internet Service Provider)

Kommunikationsassoziation Kommunikationsassoziation sind Kommunikationsbeziehungen zwischen zwei Anwendungsinstanzen. Diese tauschen über eine beliebige und nicht konstante Menge von Zwischensystemen Daten aus. Der Datenaustausch wird dabei über einen definierten von der Anwendung gewählten Transportdienst umgesetzt, der für die Dauer der Kommunikationsassoziation fest mit dieser verbunden ist. Kommunikationsassoziation existieren unabhängig von Protokollen und unabhängig davon, ob gerade aktiv Daten ausgetauscht werden oder nicht. Die Lebensdauer wird nur durch die Anwendungsinstanzen, welche die Kommunikationsassoziationen aufbauen und beenden, bestimmt.

Kommunikationsbeziehung Eine Kommunikationsbeziehung ist im weitesten Sinne der Vorgang des Datenaustausches zwischen zwei oder mehr beteiligten Parteien. Bei den beteiligten Parteien handelt es sich um konkrete Anwendungsinstanzen einer oder verschiedener Anwendungen.

Kommunikationsdomäne Innerhalb einer Kommunikationsdomäne sind alle End- und Zwischensysteme über einheitliche Adressen untereinander erreichbar. Jedes System kann Teil einer oder auch mehrere Kommunikationsdomänen sein. Heute existierende Kommunikationsdomänen sind beispielsweise IPv4 oder IPv6 Domänen.

Kommunikationsmuster Ein Kommunikationsmuster legt fest wie zwei oder mehr Anwendungsinstanzen Informationen untereinander austauschen. Zum Beispiel über einen *Bus*, bei dem jede Nachricht an alle Anwendungsinstanzen weitergeleitete wird, oder indirekt über einen Nachrichten Broker, wie er beispielsweise beim *publish-subscribe* Muster zum Einsatz kommt.

Kommunikationsparadigma Ein Kommunikationsparadigma definiert grundlegende Elemente einer Kommunikation, wie beteiligte Subjekte, ausgetauschte Objekte und Art der Kommunikation. Im Internet existiert die Ende-zu-Ende Kommunikation, bei der Daten in Form von Datagrammen oder eines Datenstroms zwischen Client und Server ausgetauscht werden. Ein alternatives Kommunikationsparadigma stellt die inhaltsbasierte Kommunikation da, bei der statt fester Endsysteme Inhalte in Form von Datenblöcken direkt adressiert werden.

Nachricht Eine Nachricht enthält die zwischen Anwendungsinstanzen ausgetauschten Nutzdaten. Sie wird als Paket verschickt.

Paket Ein Paket enthält eine Nachricht und die notwendigen Kontrollinformationen (PCI, Protocol Control Information), um die Nachricht an den gewünschten Zielort weiterzuleiten.

Protokollstapel Als Protokollstapel wird eine Kombination von Protokollen der Schichten 1-4 bezeichnet, welche im Zusammenspiel einen bestimmten Transportdienst anbieten.

Transportbeziehung Eine Transportbeziehung ist ein bestehender Kanal zwischen zwei Systemen zum Datenaustausch über ein Transportprotokoll. Transportbeziehung bezeichnet dabei sowohl einen Kanal über verbindungsorientierte als auch über verbindungslose Protokolle.

Transportdienst Transportdienste unterscheiden sich in der Art und Weise, wie Daten durchs Netz transportiert werden. Transportdienste setzten verschiedene Transportdiensteigenschaften um, um eine bestimmte Dienstgüte für den Austausch von Daten zwischen Anwendungsinstanzen zu gewährleisten. Beispielsweise Transportdiensteigenschaften wie Vollständigkeit oder Fehlerfreiheit. Transportdienste sind definiert durch eine Beschreibung ihrer Eigenschaften, welche von konkreten Transportprotokollen abstrahiert.

Transportprotokoll Transportprotokolle setzen die durch Transportdienste abstrakt definierten Dienste um. Dazu werden Transportdienstbeschreibungen auf passende Transportprotokolle abgebildet und diese entsprechend der gewünschten Transportdiensteigenschaften parametrisiert.

Transportverbindung siehe Transportbeziehung

verteilte Anwendung siehe Anwendung

Zwischensystem Zwischensysteme sind alle Systeme die Teil eines Netzwerkes sind und Daten für eine Kommunikationsbeziehung transportieren, ohne ein Absender oder Empfänger der Daten zu sein. Beispielsweise Router und Brücken im Netzwerk, aber auch anderen netzwerkfähigen Geräte wie Smartphones oder PCs, welche bei der Umsetzung einer Kommunikationsbeziehungen helfen, an denen sie nicht als Endsystem beteiligt sind.

Akronyme

ADU Application Data Unit

API Application Programming Interfac

CCN Content Centric Networking

DCCP Datagram Congestion Control Protocol

DCTCP Data Center TCP

DNS Domain Name System

HTTP Hypertext Transfer Protocol

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

IPC Inter-process communication

ISP Internet Service Provider

LEDBAT Low Extra Delay Background Transport

MPTCP MultiPath TCP

MQTT Message Queue Telemetry Transport

MSS Maximum Segment Size

NDN Named Data Networking

NENA Netlet-based Node Architecture

NFV Network Function Virtualization

PASTE Protocol Agnostic Services & Transport Enrichment

PCI Protocol Control Information

PDU Protocol Data Unit

RPC Remote Procedure Call

RTT Round Trip Time

SCTP Stream Control Transmission Protocol

SDN Software Defined Networking

SMTP Simple Mail Transfer Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

UDP-Lite Lightweight User Datagram Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

Literatur

[Alv16] H. Alvestrand. *Overview: Real Time Protocols for Browser-based Applications*. Internet-Draft draft-ietf-rtcweb-overview-15. IETF Secretariat, Jan. 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-overview-15.txt.

[Ben+15] S. Bensley u. a. *Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters*. Internet-Draft draft-bensley-tcpm-dctcp-05. IETF Secretariat, Juli 2015. URL: http://www.ietf.org/internet-drafts/draft-bensley-tcpm-dctcp-05.txt.

[BFM05] T. Berners-Lee, R. Fielding und L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986 (INTERNET STANDARD). Updated by RFCs 6874, 7320. Internet Engineering Task Force, Jan. 2005. URL: http://www.ietf.org/rfc/rfc3986.txt.

[BMW12] H. Backhaus, D. Martin und H. Wippel. *A Network Pluralist's Approach to Online Video Stores*. Demonstrator. Juli 2012.

[BP12] M. Belshe und R. Peon. *SPDY Protocol*. Internet-Draft draft-mbelshe-httpbis-spdy-00. IETF Secretariat, Feb. 2012. URL: http://www.ietf.org/internet-drafts/draft-mbelshe-httpbis-spdy-00.txt.

[BPT15] M. Belshe, R. Peon und M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540 (Proposed Standard). Internet Engineering Task Force, Mai 2015. URL: http://www.ietf.org/rfc/rfc7540.txt.

[CBA14] R. Craven, R. Beverly und M. Allman. „A Middlebox-cooperative TCP for a Non End-to-end Internet". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. Chicago, Illinois, USA: ACM, 2014, S. 151–162. ISBN: 978-1-4503-2836-4. DOI: 10.1145/2619239.2626321. URL: http://doi.acm.org/10.1145/2619239.2626321.

[CC83] V. G. Cerf und E. Cain. „The DoD internet architecture model". In: *Computer Networks (1976)* 7.5 (1983), S. 307–318.

[Che+05] L.-J. Chen u. a. „Enhancing QoS Support for Vertical Handoffs Using Implicit/Explicit Handoff Notifications". In: *Proceedings of the Second International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks*. Washington, DC, USA: IEEE Computer Society, 2005, S. 37–. ISBN: 0-7695-2423-0. DOI: 10.1109/QSHINE.2005.22. URL: http://dx.doi.org/10.1109/QSHINE.2005.22.

[Che+06] L. J. Chen u. a. „USHA: a simple and practical seamless vertical handoff solution“. In: *IEEE Consumer Communications and Networking Conference*. Bd. 2. 2006.

[Cis12a] Cisco Systems Inc. *Cisco Visual Networking Index: Forecast and Methodology, 2011–2016*. Mai 2012. URL: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[CLC08] Y.-S. Choi, K.-W. Lee und Y.-Z. Cho. „Information Networking. Towards Ubiquitous Networking and Services: International Conference, ICOIN 2007, Estoril, Portugal, January 23-25, 2007. Revised Selected Papers“. In: Hrsg. von T. Vazão, M. M. Freire und I. Chong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Kap. TCP with Explicit Handoff Notification for a Seamless Vertical Handoff, S. 831–840. ISBN: 978-3-540-89524-4. DOI: 10.1007/978-3-540-89524-4_82. URL: http://dx.doi.org/10.1007/978-3-540-89524-4_82.

[Det+13] G. Detal u. a. „Revealing Middlebox Interference with Tracebox“. In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Barcelona, Spain: ACM, 2013, S. 1–8. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504757. URL: http://doi.acm.org/10.1145/2504730.2504757.

[Dut+07] R. Dutta u. a. „The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet“. In: *Communications, 2007. ICC '07. IEEE International Conference on*. Juni 2007, S. 1899–1904. DOI: 10.1109/ICC.2007.316.

[Fie00] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. AAI9980887. Diss. 2000. ISBN: 0-599-87118-0.

[For+13] A. Ford u. a. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824 (Experimental). Internet Engineering Task Force, Jan. 2013. URL: http://www.ietf.org/rfc/rfc6824.txt.

[FTK16] G. Fairhurst, B. Trammell und M. Kühlewind. *Services provided by IETF transport protocols and congestion control mechanisms*. Internet-Draft draft-ietf-taps-transports-10. http://www.ietf.org/internet-drafts/draft-ietf-taps-transports-10.txt. IETF Secretariat, März 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-taps-transports-10.txt.

[Fun12] I. Funke. „Realisierung des Content-Centric Networking-Ansatzes in der Netlet-based Node Architecture“. Betreuer: Helge Backhaus, Denis Martin, Hans Wippel. Bachelorarbeit. Karlsruher Institut für Technologie, 2012.

[Ham+16] R. Hamilton u. a. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*. Internet-Draft draft-tsvwg-quic-protocol-02. IETF Secretariat, Jan. 2016. URL: http://www.ietf.org/internet-drafts/draft-tsvwg-quic-protocol-02.txt.

[Jac+09a] V. Jacobson u. a. „Networking Named Content“. In: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '09. Rome, Italy: ACM, 2009, S. 1–12. ISBN: 978-1-60558-636-6. DOI: 10.1145/1658939.1658941. URL: http://doi.acm.org/10.1145/1658939.1658941.

[Jac+09b] V. Jacobson u. a. „VoCCN: Voice-over Content-centric Networks“. In: *Proceedings of the 2009 Workshop on Re-architecting the Internet*. ReArch '09. Rome, Italy: ACM, 2009, S. 1–6. ISBN: 978-1-60558-749-3. DOI: 10.1145/1658978.1658980. URL: http://doi.acm.org/10.1145/1658978.1658980.

[KHF06] E. Kohler, M. Handley und S. Floyd. *Datagram Congestion Control Protocol (DCCP)*. RFC 4340 (Proposed Standard). Updated by RFCs 5595, 5596, 6335, 6773. Internet Engineering Task Force, März 2006. URL: http://www.ietf.org/rfc/rfc4340.txt.

[Lab+10] C. Labovitz u. a. „Internet Inter-domain Traffic“. In: *SIGCOMM Comput. Commun. Rev.* 40.4 (Aug. 2010), S. 75–86. ISSN: 0146-4833. DOI: 10.1145/1851275.1851194. URL: http://doi.acm.org/10.1145/1851275.1851194.

[Lin] Linux Foundation Collaborative. *OpenAPI Specification*. Accessed: 2016-05-18. URL: http://openapis.org/specification/.

[Mai+09] G. Maier u. a. „On Dominant Characteristics of Residential Broadband Internet Traffic“. In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '09. Chicago, Illinois, USA: ACM, 2009, S. 90–102. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644904. URL: http://doi.acm.org/10.1145/1644893.1644904.

[MVZ11] D. Martin, L. Völker und M. Zitterbart. „A Flexible Framework for Future Internet Design, Assessment, and Operation“. In: *Computer Networks* 55.4 (Feb. 2011), S. 910–918. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.12.015.

[MWB11] D. Martin, H. Wippel und H. Backhaus. „A Future-Proof Application-to-Network Interface“. In: *Proceedings of the 2011 Second International Conference on the Network of the Future (NoF 2011)*. IEEE, Nov. 2011.

[Nay+15] D. Naylor u. a. „Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS“. In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), S. 199–212. ISSN: 0146-4833. DOI: 10.1145/2829988.2787482. URL: http://doi.acm.org/10.1145/2829988.2787482.

[Pan+16] A. Panda u. a. „Open Network Interfaces for Carrier Networks“. In: *SIGCOMM Comput. Commun. Rev.* 46.1 (Jan. 2016), S. 5–11. ISSN: 0146-4833. DOI: 10.1145/2875951.2875953. URL: http://doi.acm.org/10.1145/2875951.2875953.

[PGS10] L. Popa, A. Ghodsi und I. Stoica. „HTTP As the Narrow Waist of the Future Internet“. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 6:1–6:6. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868453. URL: http://doi.acm.org/10.1145/1868447.1868453.

[Pra13] P. A. Prats. „Implementation of MQTT protocol inside NENA“. Betreuer: Helge Backhaus, Denis Martin. Bachelorarbeit. Karlsruher Institut für Technologie, 2013.

[Rai+12] C. Raiciu u. a. „How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP“. In: *Proceedings of the 9th conference on Networked Systems Design and Implementation (NSDI)*. NSDI'12. San Jose, CA: USENIX Association, Apr. 2012.

[San12] Sandvine Incorporated ULC. *Global Internet Phenomena Report 2H 2012*. Nov. 2012. URL: http://www.sandvine.com/downloads/documents/Phenomena_2H_2012/Sandvine_Global_Internet_Phenomena_Report_2H_2012.pdf.

[SN16] A. Stanford-Clark und A. Nipper. *Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1*. Standard. International Organization for Standardization, Feb. 2016.

[Ste+04] R. Stewart u. a. *Stream Control Transmission Protocol (SCTP) Partial Reliability Extension*. RFC 3758 (Proposed Standard). Internet Engineering Task Force, Mai 2004. URL: http://www.ietf.org/rfc/rfc3758.txt.

[Ste07] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960 (Proposed Standard). Updated by RFCs 6096, 6335, 7053. Internet Engineering Task Force, Sep. 2007. URL: http://www.ietf.org/rfc/rfc4960.txt.

[The] The RAML Workgroup. *RAML - RESTful API Modeling Language*. Accessed: 2016-05-18. URL: http://raml.org/.

[Til] O. Tilmans. *Tracebox - A Middlebox Detection Tool*. Accessed: 2016-05-18. URL: http://www.tracebox.org/.

[TP08] J. Touch und V. Pingali. „The RNA Metaprotocol“. In: *Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on*. Aug. 2008, S. 1–6. DOI: 10.1109/ICCCN.2008.ECP.46.

[Uni16] Universal Binary JSON Specification. „Universal Binary JSON“. available at http://www.ubjson.org/. 2016. URL: http://www.ubjson.org/.

[W3C01] W3C Recommendation. „Web Services Description Language (WSDL) 1.1“. available at http://www.w3.org/TR/wsdl.html. 2001. URL: http://www.w3.org/TR/wsdl.html.

[W3C04a] W3C Recommendation. „OWL Web Ontology Language Semantics and Abstract Syntax“. available at http://www.w3.org/TR/owl-semantics/. 2004. URL: http://www.w3.org/TR/owl-semantics/.

[W3C04b] W3C Recommendation. „Resource Description Framework (RDF): Concepts and Abstract Syntax“. available at http://www.w3.org/TR/rdf-concepts/. 2004. URL: http://www.w3.org/TR/rdf-concepts/.

[WTK16] M. Welzl, M. Tuexen und N. Khademi. *On the Usage of Transport Service Features Provided by IETF Transport Protocols*. Internet-Draft draft-ietf-taps-transports-usage-00. http://www.ietf.org/internet-drafts/draft-ietf-taps-transports-usage-00.txt. IETF Secretariat, Jan. 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-taps-transports-usage-00.txt.

[WY12] D. Wing und A. Yourtchenko. *Happy Eyeballs: Success with Dual-Stack Hosts*. RFC 6555. http://www.rfc-editor.org/rfc/rfc6555.txt. RFC Editor, Apr. 2012. URL: http://www.rfc-editor.org/rfc/rfc6555.txt.