Gleiche-zu-Gleiche-Protokolle auf Basis asynchroner Nachrichtenkommunikation mit prototypischer Implementierung eines Pastry-Netzwerks

Studienarbeit

Cand. Inform. Alexander Liebrich

Tag der Anmeldung:	01. 10. 2005
Tag der Abgabe:	31. 12. 2005

Betreuer:	Prof. Walter F. Tichy
Betreuender Mitarbeiter:	Dipl. Inform. Marc Schanne

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die verwendeten Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, im Januar 2006 .....

Unterschrift (Alexander Liebrich)

Kurzfassung

Der in verschiedenen EU-Forschungsprojekten am Forschungsbereich Software Engineering (SE) im FZI Karlsruhe entwickelte Echtzeit Nachrichtendienst (EventChannelNetwork) bietet im Umfeld verteilter, sicherheits- und geschäftskritischer Systeme eine zuverlässige Infrastruktur für Nachrichten-Kommunikation mit Publiziere/Abonniere-Logik. Basierend auf dem Konzept von logischen Nachrichtenkanäle werden diese zur Gruppierung der Nachrichten und Definition von Anforderungen an die verarbeitenden Kontrollfäden genutzt. Über die Rundruflogik dieser physikalischen Netzwerke muss für herkömmliche Implementierungen immer zuerst eine Punkt-zu-Punkt-Verbindungen aufgebaut werden und hierauf wird wieder die Viele-zu-Viele-Kommunikation des P2P-Netzes implementiert. Durch die direkte Anbindung des Nachrichtendienstes kann dies optimiert werden. Die vorliegende Studienarbeit untersucht diese Möglichkeiten anhand von Anforderungen bestehender P2P-Programmbibliotheken und definiert eine Abbildung auf den Nachrichtendienst. Zur Verifikation des Konzepts wird eine offene Implementierung der P2P-API Pastry als Prototyp entwickelt.

Der resultierende Kommunikationsaufwand reduziert sich, unter Beibehaltung der Pastry-Protokollkonformität, um die für die Punkt-zu-Punkt-basierten Weiterleitung benötigten O(log(n)) Nachrichten (mit n Knoten im Netzwerk) und ist in der EventChannelNetwork-basierten Implementierung unabhängig von der Größe des Netzwerks. In dieser Ausarbeitung wird die Eignung des Nachrichtendienstes für verteilte, dezentral organisierte Applikationen untersucht. Dabei stellt die Ausfallsicherheit einen wichtigen Aspekt dar. Da die Datenstrukturen für die Weiterleitung nicht mehr, bzw. stark reduziert und nur für bestimmte Teilnehmer des Netzwerks benötigt werden, werden die Anforderungen an die Rechenleistung und Speicherkapazität verringert. Diese Ressourcen stehen entweder der Applikationsebene zur Verfügung, oder können für eine erweiterte gegenseitige Überwachung der Knoten untereinander ausgenutzt werden.

Dadurch steht unter Benutztung des EventChannelNetwork als Pastry-Kommunikationsschnittstelle ein weitgehend plattformunabhängiges Gleiche-zu-Gleiche-Netzwerk zur Verfügung.

Inhaltsverzeichnis

1 Motivation zur Realisierung einer P2P-Bibliothek basierend auf einem Echtzeit-Nachrichtendienst

Das von Gordon Moore postulierte so genannte Moorsche Gesetz basiert auf der Beobachtung des Ingenieurs Moore, dass sich die Computerleistung bezüglich Rechen- und Speicherkapazität alle 18 Monate verdoppelt[23]. Analog lässt sich diese Beobachtung in den letzten Jahren auch im Telekommunikationsbereich erkennen. Ebenso wie die rasante Steigerung der Datenraten stehen immer schneller neue Kommunikations- und Verknüpfungsmedien zur Verfügung und lösen veraltete Netze ab. Heutige Konzepte basieren meist auf dem einer zentralen Einheit, dem Dienstgeber, dessen Ressourcen unter den anfragenden Dienstnehmern, meist gleichmäßig, aufgeteilt werden. Es ist augenscheinlich, dass mit steigenden Nutzerzahlen eine Dienstleistung nur noch eingeschränkt, bzw. gar nicht mehr erfolgen kann. Resultierend muss die Leistung der Dienstgeberseite immer weiter gesteigert und neue Methoden zur Lastaufteilung beispielsweise nach dem Haupt- und Satellitenrechner-Modell entworfen werden. Diese Herangehensweise muss meist individuell gelöst werden und ist damit nur unter hohem Aufwand und mit großen Kosten zu verwirklichen. Außerdem beinhalten diese Lösungen durch ihre zentrale Organisation immer noch eine Skalierbarkeitsgrenze bezüglich ihrer Klienten. Nicht nur durch „böswillige" Angriffe, sondern auch durch zu hohe Last kommt es bei namhaften Dienstleistern immer wieder zu Ausfällen. Ansätze, welche die Ressourcen ihrer Klienten mitbenutzen und auf diese Art das Netz nicht nur um seine Teilnehmerzahl, sondern auch um dessen Leistungsfähigkeit erweitern, sind damit vielversprechender.

Diese Klasse der Überlagerungsnetze nennt man Gleiche-zu-Gleiche-Netze (Peer2Peer, P2P), bei denen die Unterscheidung zwischen Dienstgeber und -nehmer verblasst. Eine Betrachtung einiger dieser Gleiche-zu-Gleiche Netzwerk befindet sich in Abschnitt 2. Die meisten verfügbaren Systeme dieser Gattung basieren auf dem Prinzip der Verteilten Hash-Tabelle. Dieses dezentrale Verwaltungsprinzip bildet die Ausgangsbasis für das, in dieser Arbeit dargestellte Gleiche-zu-Gleiche System. Unterschiedliche Ansätze und allgemeine Zusammenhänge haben sich herausgebildet, welche in Absatz 2.2 behandelt werden.

Aus den anfangs ad-hoc-programmierten verteilten Anwendungen sind neue Forschungsbereiche entstanden, da man die Möglichkeiten der Skalierbarkeit, Ausfallsicherheit und für Dienstleister mögliche Kosteneinsparungen erkannte. Gleiche-zu-Gleiche-Netze erfreuen sich so einer großen Verbreitung und finden nicht nur im Bereich der Forschung immer neue Verwendungsmöglichkeiten. Aktuelle Bestrebungen versuchen einen Standard für die Gleiche-zu-Gleiche-Kommunikation zu entwickeln. Hierzu gehören u.a. JXTA oder Pastry. Diese beiden Programmbibliotheken (API) werden im Abschnitt 2.3 über konkrete verteilte Hash-Tabellen vorgestellt.

Gleiche-zu-Gleiche Netzwerke werden als Überlagerungsnetze (Overlays) bezeichnet, da sie üblicherweise auf dem IP-adressierten Internet aufbauen. Die in dieser Studienarbeit angestrebte Lösung fußt auf einem, am Forschungszentrum für Informatik, Karlsruhe (FZI) entwickeltem asynchronen Nachrichtendienst, dem EventChannelNetwork (ECN, siehe Abschnitt 3.2). Das EventChannelNetwork arbeitet blockierungsfrei nach dem Rundrufprinzip und besitzt damit schon eine Eigenschaft, welche von sonstigen P2P-Systemen erst emuliert werden muss.

In dieser Ausarbeitung soll nun gezeigt werden, wie eine Gleiche-zu-Gleiche-API, in diesem Fall Pastry[27], auf Basis einer asynchronen Rundrufkommunikation realisiert werden kann. Eine Betrachtung genereller Unterschiede der beiden Kommunikationsformen werden in Abschnitt 3 erfasst. Aufbauend auf diesen Erkenntnissen entsteht eine grobe Planung (Abschnitt 4), welche im darauf folgenden Abschnitt 5 in den Details eines prototypischen Vorschlags für eine Implementierung mündet.

Abschließend soll ein einfaches Beispiel betrachtet werden, an welches sich noch ein abrundender Ausblick in Abschnitt 6 mit möglichen Anwendungen und Erweiterungen anknüpft.

1.1 Pastry: eine mögliche API

Pastry ist eine konkrete P2P-Bibliothek und ein Quasi-Standard, der die Wiederverwendung bestehender Applikationen ermöglicht. Der Transport von Meldungen innerhalb des Netzes findet

über das Teilmodul „Sockets" der Pastry-Bibliothek statt und basiert auf einer Punkt-zu-Punkt-Kommunikation. Jeder Knoten besitzt eine eindeutige Identität, was der Abstraktion, bzw. Lösung der IP-basierten Adressierung dient. Ein Pastry-internes Weiterleitungsprotokoll (Dynamik Source Routing) steuert dezentral den Fluss der Nachrichten über mehrere Knoten hinweg. Erhält ein Knoten eine, mit einer Knoten-ID adressierten Meldung, welche ihn nicht selbst betrifft, so kann er diese auf dem kürzesten Pfad zum Zielknoten weiterleiten. Die Zahl der Weiterleitungsschritte ist bei einer Netzgröße von N Knoten auf $O(logN)$ begrenzt. Jeder Pastry-Knoten überwacht eine bestimmte Anzahl seiner nächsten Nachbarn innerhalb des Knoten-ID-Raums und beachtet auch die Entfernungen (bzgl. der physikalischen Verbindung) zu diesen Knoten (Dynamic Proximity-aware Source Routing, DPSR).

1.2 P2P über RMI – ein Turm zu Babel

Die Kommunikation mittels entfernter Methodenaufrufe (RMI) erfolgt mit direkten Punkt-zu-Punkt Verbindungen. Eine Anfrage per RMI ist an eine eindeutige Objektreferenz gekoppelt und wirkt auf den aufrufenden Kontrollfaden blockierend. Diese Tatsache macht eine Anfrage in parallelen Prozessen nötig und bedingt somit erhöhte Ressourcenanforderungen. Da in eingebetteten Systemen Echtzeitgarantien von essentieller Bedeutung sind, muss, um bestimmte Antwort- und Bearbeitungszeiten garantieren zu können, eine Prioritätssteuerung vorhanden sein.

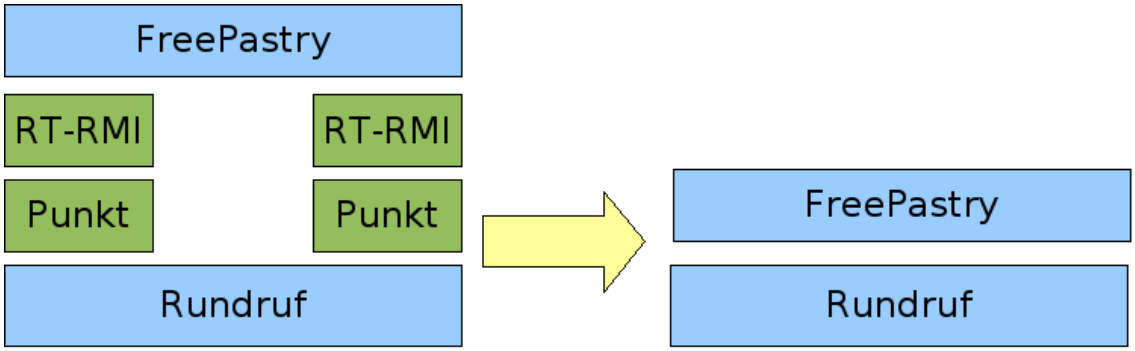

Abbildung 1: Vergleich der Protokollstapel zwischen RMI- und ECN-Pastry

Da RMI einen Punkt-zu-Punkt orientierten Transport verwendet, wird zusätzlich ein Routing-Verfahren nötig, welches wiederum Ressourcen-intensiv und komplex in seiner Implementierung ist. Für die RMI-basierte Kommunikation ist im Bereich der eingebetteten Systeme eine weitere Zwischenschicht nötig, da hier meist Rundruf-Netze (CAN, Feldbusse) eingesetzt werden und somit zunächst eine Emulation der Punkt-zu-Punkt Verbindungen in einem Viele-zu-Viele Netz erforderlich ist.

1.3 EventChannelNetwork als Basis einer P2P-Bibliothek

Der Publiziere/Abonniere Nachrichtendienst (RealTime-EventChannelNetwork, RT-ECN) in HIJA benutzt anonyme Kommunikation, was der Skalierbarkeit entgegenkommt. Eine Übertragung erfolgt mittels Rundruf und macht ein aufwendiges Weiterleiten der Meldungen von Knoten zu Knoten unnötig. Die verwendete asynchrone Kommunikation blockiert einen, auf eine Antwort wartenden Knoten nicht und kann sich deshalb in der Zwischenzeit um die Abarbeitung weiterer Aufgaben kümmern. Beliebig viele physikalische Verbindungen sind in logischen Kanälen zusammengefasst. Diese Kanäle werden nach Abonniere-/Publiziere-Logik (dynamisch) erzeugt und verwaltet. Erst beim Überbrücken zweier solcher logischer Kanäle ist ein Weiterleitungsmechanismus nötig. Da der RT-ECN speziell für Echtzeitanforderungen konzipiert ist, sind die Kanäle mit einer Priorität behaftet, welche es erlaubt, bestimmte Nachrichten bevorzugt zu behandeln. Der RT-ECN existiert in einer Version mit harten und flexiblen Echtzeitfähigkeiten und bietet

in der flexiblen Ausgabe eine Fehlerbehandlung durch Java Exceptions auf Anwendungsebene bei Verletzung von Echtzeitgarantien. Die dem RT-ECN eigene Prioritätsverwaltung regelt den Transport von Nachrichten, sowie deren Bearbeitung in Form von Aktivitäten und ermöglicht somit garantierte Antwort- und Bearbeitungszeiten

1.4 Mögliche Verwendungsszenarien

Im Hinblick auf eingebettete Systeme, die zuverlässig ihren Dienst verrichten sollen, oder „lebenskritische" Prozesse steuern, ist die Robustheit dieser Gleiche-zu-Gleiche-Netzwerke von Bedeutung. Da diese Netzwerke auf vorhandenen Protokollen aufbauen, bzw. diese überlagern sind sie in der Lage auch unterschiedlichste Systeme untereinander zu verbinden. Die Skalierbarkeit durch plattformunabhängiges Hinzufügen neuer Komponenten gestaltet sich einfacher, als eine Erweiterung, die eine Umstellung der zugrundeliegenden Rechen- und Transport-Systeme mit sich bringt. Die asynchrone und damit blockierungsfreie Kommunikation stellt eine Grundlage für echtzeitkritische Anwendungen dar. Erst durch die zusätzliche, dem EventChannelNetwork eigene Prioritätssteuerung können garantierte Antwortzeiten realisiert werden. Der Nachrichtendienst arbeitet zudem nach dem Rundrufprinzip und eignet sich dadurch besonders für den Einsatz in diesem Aufgabenbereich, da hier meist Protokolle nach diesem Kommunikationsschema zum Einsatz kommen.

Für allgegenwärtige und mobile Systeme bieten sich die Gleiche-zu-Gleiche-Netzwerke an, da sie unter ständig wechselnden Verbindungen zwar Leistungseinbußen aufzeigen, sich jedoch bei geeigneter Konzeption, als sehr robust erweisen. Erreicht wird dies durch eine gegenseite Überdeckung im Netzwerk vorhandener Ressourcen. Die für eine dynamische Weiterleitung nötigen Datenstrukturen müssen mit geringem Aufwand konsistent gehalten werden und auch unter schnell wechselnder Konnektivität eine kontinuierliche Verbindung sicherstellen.

2 Gleiche-zu-Gleiche Netzwerke

Da Gleiche-zu-Gleiche Netzwerke zu der Klasse der Überlagerungsnetze gehören und in der Praxis meist auf einer IP-Adressierte Punkt-zu-Punkt-Kommunikation aufbauen, realisieren eine anwendungsspezifische Adressierung. Ein Gleiche-zu-Gleiche (Peer-to-peer, P2P) Rechnernetzwerk bezeichnet ein Netzwerk, welches auf den Ressourcen (Rechen- und Transferleistung) seiner Teilnehmer basiert, anstatt diese in Form einiger weniger Dienstgeber zu konzentrieren. P2P-Netzwerke kommen typischerweise zum Einsatz, wenn weitestgehend „ad-hoc"-Verbindungen vorliegen. Ein reines Gleiche-zu-Gleiche-System besitzt keine Unterscheidung zwischen Dienstgeber und Nehmer, und besteht aus gleichartigen Teilnehmer, die zeitlich parallel als Geber und Nehmer von Diensten handeln. Dieses Netzwerkmodel unterscheidet sich vom herkömmlichen Dienstgeber-Dienstnehmer-Modell, bei dem die Kommunikation gewöhnlicherweise über einige wenige zentrale Dienstgeber stattfindet. Einige Netzwerke, wie Napster[8], OpenNap[9] oder IRC@find, nutzen eine Dienstgeber/Dienstnehmer-Struktur nur für bestimmte Aufgaben, beispielsweise der Indizierung mit anschließender Suche, und die P2P-Struktur für die übrigen Funktionalitäten. Netzwerke, wie Gnutella oder Freenet benutzen die P2P-Struktur für alle von ihnen benötigten Funktionen, obwohl Gnutella so genannte Verzeichnisdienstgeber nutzt, um Knoten über ihre Kollegen zu informieren. Aus dieser Tatsache heraus lassen sich drei Varianten dieser Netzwerke unterscheiden:

- Reine Gleiche-zu-Gleiche Netzwerke: Einheiten sind Dienstgeber und Dienstnehmer. Es gibt keine zentrale, das Netzwerk verwaltende Einheit und auch keine zentrale Weiterleitungseinheit.
- Hybrid P2P: Besitzt einen zentralen Dienstgeber, welcher die Teilnehmer und die Ressourcen indiziert und diese Informationen auf Anfrage zur Verfügung stellt. Die einzelnen Teilnehmer sind zuständig für die Speicherung und Verwaltung der im Netzwerk enthaltenen Ressourcen, da diese nicht auf dem Server zu finden sind.

- gemischte P2P: Besitzen Eigenschaften der reinen und hybriden Gleiche-zu-Gleiche Netzwerke. Ein allseits bekanntes Beispiel hierfür ist das Domänenbenennungssystem (Domain-Name-System, DNS).

2.1 Aspekte der Gleiche-zu-Gleiche Überlagerungsnetzwerke

In den folgenden Unterabschnitten 2.1.1 bis 2.1.3 werden allgemeine Eigenschaften der Gleiche-zu-Gleiche Netzwerke betrachtet und verhelfen so zu einem Überblick der im weiteren verwendeten Begriffe und Sachverhalte. Wie zuvor schon angedeutet ist ein wichtiges Ziel der Gleiche-zu-Gleiche Netzwerke, dass alle Dienstnehmer ihre Ressourcen bezüglich Transferbandbreite, Speicherplatz und Rechenleistung) anbieten. Auf die Weise wird, wenn neue Einheiten hinzukommen und eine Erweiterung des Systems verlangen, die totale Kapazität des Systems mitvergrößert. Somit steht einer unbegrenzten Skalierbarkeit nichts mehr im Wege. Dies ist bei einer unsymmetrischen Dienstgeber-/Dienstnehmer-Struktur, bei der die Ressourcen auf die anfragenden Teilnehmer aufgeteilt wird, nicht der Fall. Auf diese Weise kann, durch eine geschickte Maximierung des Datenflusses über alle, einen Dienst benutzenden Knoten hinweg, als auch unter diesen untereinander, (beispielsweise eine Echtzeitvideoübertragung) ohne Beachtung der zu erwarteten Nutzerzahlen implementiert werden. Hier wird sozusagen eine Einer-zu-Viele-Kommunikation emuliert. Auch ein Verteilen großer Datenmengen, (beispielsweise einer Linux-Distribution) welche oft mehrere GByte umfasst, kann mit dieser Technik effizient und ohne hohe Kosten für eine entsprechende Dienstgeber-Infrastruktur realisiert werden (vgl. BitTorrent[3]), wobei in diesem Fall keine (weichen) Echtzeitanforderungen, sondern die Konsistenz der übertragenen Daten von oberster Priorität ist. Dieses Prinzip bietet eine Vielzahl an Möglichkeiten für verschiedenste Anwendungen, welche hier aber nicht weiter ausgeführt werden sollen. Statt dessen folgt im Anschluss eine Betrachtung der frühen Ursprünge bis hin zum heutigen Stand der Forschung an.

2.1.1 Perspektiven in der Informationswissenschaft

Die zuvor spezifizierte Kategorie der „Reinen" Gleiche-zu-Gleiche Netzwerke stellt nur eine kleine Gruppe von Anwendungen dar, da die meisten Anwendungen die ein oder andere Komponente, welche eine Unterscheidung zwischen Dienstgeber und Nehmer zulässt, enthält. Zu dieser Kategorie gehören die schon in die Jahre gekommenen Anwendungen Usenet[32] (1979), sowie FidoNet[] (1984). Viele Peer2Peer Systeme nutzen eine Hierarchisierung der Knoten, was meist eine sternförmige Netzwerktopologie zur Folge hat. Hierzu gehört auch das Projekt JXTA, welches in Abschnitt 2.3.1 genauer betrachtet wird.

Im Bereich der Informationswissenschaften beschäftigt man sich in den letzten Jahren verstärkt mit Systemen und Anwendungen, welche nach dem Gleiche-zu-Gleiche Prinzip konzipiert sind. Es gibt immer noch offene Fragen, beispielsweise in Bezug auf die Sicherheit in solchen verteilten Systemen. Eine Auswahl der Projekte, die zu Forschungszwecken ins Leben gerufen wurden sind im Folgenden aufgelistet:

- Chord[4, 33] wurde vornehmlich zu Forschungszwecken ins Leben gerufen.
- PAST[18, 28] baut auf der Pastry-Transportschicht (Middleware) durch persistente Speichere- und Lade-Funktionen eine verteilte Hash-Tabelle.
- P-Grid[10] benutzt im Unterschied zu den zuvor genannten Gleiche-zu-Gleiche-Systemen eine andere Organisationsform (siehe Abschnitt 2.1.3). Dieses Projekt zielt in nicht in erster Linie auf eine verteilte Datenhaltung, als vielmehr auf eine allgemeine Zwischenschicht für verteilte Anwendungen.
- CoopNet[5] wiederum versucht einen Mittelweg zwischen dem Dienstgeber/Dienstnehmer- und dem Gleiche-zu-Gleiche-Prinzip zu vereinen. Auf diese Weise soll die Abhängigkeit eines System von den Ressourcen seiner Teilnehmer entkoppelt, und nur bei Bedarf genutzt werden können.

2.1.2 Bedrohungen für P2P-Netze

Wie zuvor schon angedeutet, sind Fragen bezüglich der Sicherheit solcher Netze noch ein aktueller Bestand der Forschung. Da nicht, wie im zentralisierten Dienstgeber-/Dienstnehmer-Fall, diese zentralen Komponenten angegriffen werden können, entfällt diese Art der Bedrohung. Es existieren andere Probleme, welche im folgenden dargestellt werden sollen.

P2P-Netzwerke leiden zum einen unter Problemen technischer Natur, aber auch unter solchen, die auf ihren Benutzern beruhen. Zu den Problemen, die durch die Teilnehmer eines Peer2Peer-Systems verursacht werden, gehören „vergiftende" Angriffe. Eine Vergiftung des Netzwerks geschieht durch Einschleusen eines Datums, dessen Schlüssel zuvor schon mit einer anderen Ressource assoziiert wurde. Durch wiederholte Weitergabe des modifizierten Datums wird sukzessive das Original verdrängt, bis es schließlich nicht mehr zu finden ist. Weitere Probleme treten zwischen den Nutzern auf, etwa durch Beleidigung, oder Identitätsdiebstahl. Diese Probleme, wie auch Überflutungsangriffe oder eine Verseuchung mit Viren oder anderer Schad-Software, existieren ebenso, wie in der Dienstgeber/Dienstnehmer-Architektur, jedoch stellt sich eine Überwachung eines verteilten Systems als ungleich schwieriger dar.

Der gleiche Effekt, wie bei dem zuvor erwähnten vergiftenden Angriff tritt auf, wenn eine inkonsistente Adressierung vorliegt. Beispielsweise wird durch eine zu kleine Schlüsselmenge, welche auf eine größere Menge unterschiedlicher Daten abgebildet werden soll, eine eindeutige Zuordnung unmöglich. In Abschnitt 2.2.3 und 4.3.4 werden diese Belange wegen ihrer zentralen Bedeutung genauer betrachtet. Den meisten der hier genannten Probleme gemeinsam ist, dass zu einer Anfrage unterschiedliche Aussagen existieren, was auf das exemplarische Problem der „Byzantinischen Generäle", oder auch „Byzantinischen Fehlertoleranz" führt.

Diese Probleme gilt es auf technischer Seite auszuschließen, denn bei welcher Anzahl der dem Netzwerk schadenden Teilnehmern das System zusammenbricht, hängt start von dessen Konzeption ab. Deswegen schließt sich nun eine genauere Betrachtung der Funktionsweise dieser Gleiche-zu-Gleiche-Netzwerke an.

2.1.3 Organisationsprinzipien der P2P-Netzwerke

Bisher wurden lediglich die äußerlichen Merkmale und Eigenschaften dieser verteilten Systeme beleuchtet, weshalb nun auf die internen Aspekte, bezüglich Aufbau und Funktionsweise eingegangen wird. So lassen sich prinzipiell zwei Organisationsformen von P2P-Netzen unterscheiden: das ist zum einen die verteilte Hash-Tabelle (Distributed-Hashtable, DHT), zum anderen das dezentrale Objektlokalisations- und Weiterleitungs-System (DOLR) [19]. Der Vollständigkeit halber ist das DLOR Verfahren in diesem Abschnitt erwähnt, jedoch wird es im weiteren Verlauf nicht benötigt.

Verteile Hash-Tabelle: Eine DHT bietet die gleichen Funktionalitäten wie eine herkömmliche Hash-Tabelle, indem sie für einen Schlüssel eine Abbildung auf einen Wert liefert. Die Schnittstelle implementiert eine einfache Speichere-und-Lade Funktionalität, wobei sich die Werte in lebendigen Knoten des Überlagerungsnetzes befinden.

Distributed Objekt Localisation and Routing - DOLR: Bei diesem Verfahren wird durch eine Abstraktion ein dezentraler Verzeichnisdienst zur Verfügung gestellt. Jede Objekt-Kopie oder Endpunkt (bezüglich einer Anfrage) besitzt einen eindeutigen Objektschlüssel, welcher sich an jeder beliebigen Stelle des Systems befinden kann. Anwendungen veröffentlichen die Präsenz eines Endpunktes, indem sie ihren Standort mitteilen. Eine mit einem Objektschlüssel adressierte Nachricht kann somit an diesen Standort weitergeleitet werden. Der zugrunde liegende verteilte Verzeichnisdienst kann mit Hilfe eines annotierten Baums realisiert werden, welcher die einzelnen Objektschlüssel enthält. Prinzipiell gibt es aber auch andere Implementierungsmöglichkeiten. Ein Projekt welches auf dieser Klasse der Gleiche-zu-Gleiche-Systeme beruht ist das zuvor schon erwähnte P-Grid[]. Dieses Verfahren beruht auf einer Baumstruktur, wohingegen SkipNet[25] mit Hilfe so genannter Skip-Listen ein Überlagerungsnetzwerk aufbaut.

Da diese Verfahren in sich schnell ändernden Umgebungen eingesetzt werden sollen, ist deren Effizienz weitestgehend von dem Aufwand für Reorganisation und Konsistenzhaltung des Systems abhängig. Ebenso sind Verfahren notwendig, die das Auffinden eines Datums im System, unter Berücksichtigung der Tatsache, dass hier kein statisches System vorliegt, von Bedeutung. Hierzu ist es nötig die Funktionsweise dieser Organisationsformen zu betrachten, was im folgenden Abschnitt erfolgen soll.

2.2 Die Verteilte Hash-Tabelle

Verteile Hash-Tabellen (Distributed Hashtables, DHT) sind Grundlage für die hier vorgestellten Überlagerungsnetze. Typische Strukturen, Verfahren und allgemeine Parameter werden beleuchtet, da sich diese in den Betrachtungen über konkrete Vertreter dieser verteilten Systeme wiederfinden lassen.

Eine Hash-Tabelle bietet die Möglichkeit zu einem eindeutigen Schlüssel einen Wert zu speichern, der sich zu einem späteren Zeitpunkt, genau mit diesem Schlüssel, wieder lesen lässt. Verteilte Hash-Tabellen gehören zu der Klasse der dezentralisierten, verteilten Systeme, welche den Schlüsselraum einer herkömmlichen Hash-Tabelle über mehrere Teileinheiten verteilen. Als weiteren Unterschied müssen die Anfragen mit Hilfe eines Schlüssels effizient an die den Schlüssel enthaltende Einheit weitergeleitet werden.

Im folgenden Abschnitt sollen allgemeine Hintergründe und Funktionsweise dieses Systems erläutert werden. Die hier vorgestellten Sachverhalte bilden die theoretische Basis der in Gleiche-zu-Gleiche Netzwerke verwendeten Algorithmen.

2.2.1 Hintergründe

Motiviert durch Gleiche-zu-Gleiche-Systeme wie Napster[8], Gnutella[1] oder Freenet[7, 16], welche den Vorteil der verteilten Ressourcen ausnutzen, wurde die Forschung auf diesem Bereich der verteilten Systeme ausgeweitet. Dabei wurden einige grundlegende Zusammenhänge festgestellt, welche hier dargelegt werden.

Gleiche-zu-Gleiche Systeme unterscheiden sich in der Art, wie Daten der einzelnen Einheiten gefunden werden können. Napster benutzte einen Zentralen Index-Server, welcher eine Liste aller Daten einer Einheit anforderte und diese für Suchanfragen zur Verfügung stellte. Diese zentrale Komponente des sonst dezentral verwirklichten Dienstes macht das System anfällig für Angriffe und Ausfälle und stellt somit auch kein reines Gleiche-zu-Gleiche-System dar. Gnutella und ähnliche Netzwerke verwenden ein überflutendendes Anfrage-Modell. Zuzammenfassend lässt sich sagen, dass eine Suchanfrage in einem Rundruf an alle Einheiten bzw. Knoten resultieren würde. Während eine zentrale ausfallbehaftete Einheit, wie bei Napster, vermieden wurde, war diese Möglichkeit bei weitem weniger effizient als das Napster System. Freenet schließlich war vollkommen verteilt konzipiert, benutzte jedoch einen heuristisch schlüsselbasiertes Weiterleitungsverfahren, bei dem jede Datei mit einem Schlüssel verknüpft ist und ähnliche Schlüssel dazu tendierten, sich auf ähnlichen Knotensätzen zu verteilen (Schubladensystem,Cluster). Anfragen konnten so, mittels weniger Schritte durch das Netzwerk zu einem Knotensatz geleitet werden, jedoch konnte nicht sichergestellt werden, dass das angeforderte Datum auch vorhanden und aufgefunden werden konnte.

Verteilte Hash-Tabellen nutzen heutezutage ein weiter strukuriertes schlüsselbasiertes Weiterleitungverfahren, um zum einen die Dezentralisation von Gnutella und Freent, als auch die Effizienz und garantierte Antworten, ähnlich Napster, liefern zu können. Ein Nachteil verteilter Hash-Tabellen ist deren Möglichkeit lediglich nach exakten Anfrageschlüsseln suchen zu können, anstelle von „Schlüssel"-Wörtern. Diese Funktionalität kann auf der DHT aufbauend in der Applikationsebene realisiert werden. Die ersten verteilten Hash-Tabellen (Chord[4, 33], Pastry und Tapestry[13]) wurden zur selben Zeit, etwa 2001, veröffentlicht und dienen seither als Grundlage für die aktuelle Forschung in diesem Fachgebiet. Außerhalb des akademischen Bereichs wurden verteilte Hash-Tabellen in „BitTorrent"[3] und „Coral Content Distribution Network"[6] verwendet.

2.2.2 Eigenschaften

Die Entscheidung, ob eine verteilte Hash-Tabelle, oder eine abgeschwächte Variante oder Abwandlung vorliegt lässt sich anhand folgender Eigenschaften bestimmen. Die Hauptcharakteristik einer DHT ist ihre dezentrale Verwaltung ohne jegliche zentrale Komponente, die einer extremen Skalierbarkeit schon im Wege stehen würde. Diese massive Skalierbarkeit ist auch die zweite wichtige Eigenschaft dieses verteilten Systems, bei dem ein permanentes Hinzutreten und Verschwinden keine Leistungs-, bzw. Funktionalitätseinbußen mit sich bringen darf. Der Aufwand für die Kontrolle, bzw. Konsistenzwahrung des Systems muss so gering wie möglich gehalten werden. Erreichen lässt sich dies durch geschickte Aufteilung der Kontrolle eines einzelnen Knotens über ein paar wenige andere, meist O(log n) (bei n Knoten innerhalb des Systems). Warum sich diese Größe als geschickt erweist, wird im Abschnitt 2.2.4 erläutert.

Erweiterte Eigenschaften betreffen die Sicherheit gegenüber bösartigen Teilnehmern, oder die Wahrung der Anonymität seiner Nutzer. Hierzu sei kurz auf die Klasse der anonymen Gleiche-zu-Gleiche Netze verwiesen, auf welche hier nicht weiter eingegangen werden soll. Ebenso, wie bei herkömmlichen verteilten Systemen, sind Aspekte, wie Lastverteilung, Konsistenz und Performance, zu beachten und gehören zu den Anforderungen einer verteilten Hash-Tabelle.

2.2.3 Struktur

Verteite Hash-Tabellen lassen sich am einfachsten in zwei Teilen verstehen. Das Schlüsselaufteilungsschema verteilt die Schlüssel über Knoten, während das Überlagerungsnetzwerk die einzelnen Knoten verbindet und so das Auffinden einer zum Schlüssel gehörigen Ressource bewerkstelligt. Diese Betrachtungsweise wurde von Naor und Wieder[24], sowie Manku[22, 2] vorgeschlagen.

Die Schlüsselaufteilung Die meisten verteilten Hash-Tabellen nutzen eine konsistente Hashfunktion um eine Menge von Schlüsseln auf Knoten abzubilden. Diese Technik benutzt eine Funktion $delta(k1, k2)$, welche eine abstrakte Entfernung zwischen den Schlüsseln k1 und k2 definiert. Ein eindeutiger Schlüssel, auch Identität (ID) genannt, wird mit einem einzelnen Knoten verknüpft. Ein Knoten mit ID i besitzt auch alle Schlüssel, die sich laut *delta*-Funktion am nächsten zu i befinden.

Die DHT-Implementierung Chord[4, 33] behandelt die Schlüssel als Punkte auf einem Kreis. Der Abstand $delta(k1, k2)$ zweier Schlüssel entspricht der im Uhrzeigersinn zurückgelegten Entfernung auf dem Kreis. Sind nun i1 und i2 zwei aufeinander folgende Schlüssel, so enthält der Knoten mit ID i2 alle Schlüssel, welche sich zwischen i1 und i2 befinden.

Die Schlüssellokalisierung Da die Entfernungsfunktion $delta(k1, k2)$ eine Relation zwischen Knoten definiert, lässt sich Aufgrund dieser Ordnung ein optimaler Knoten aus der Menge der lokal überwachten bestimmen. Auf diese dezentrale Art lässt sich jeder Schlüssel nach einer bestimmten Anzahl von besuchten Knoten auffinden.

2.2.4 Allgemeine Parameter

Netzwerktopologie Da Jeder Knoten eine bestimmte Menge (meist $O(log_n)$) anderer Knoten überwachen muss, ist eine bestimmte Strukturierung dieser Knoten nötig. Beispielsweise würde sich eine Einteilung in direkte Nachbarknoten und Knoten, die nur über Mittelsknoten erreichbar sind (Weiterleitungstabelle) eignen. Auf diese Weise wird eine Topologie zwischen den Knoten definiert.

Entfernungssystem Allen VHTs gemeinsam ist die essentielle Eigenschaft, dass ein Knoten entweder einen Schlüssel besitzt, oder einen weiteren Teilnehmer kennt, welcher sich „näher" am geforderten Schlüssel k befindet. Hierbei wird die zuvor definierte $delta(k1, k2)$-Funktion verwendet.

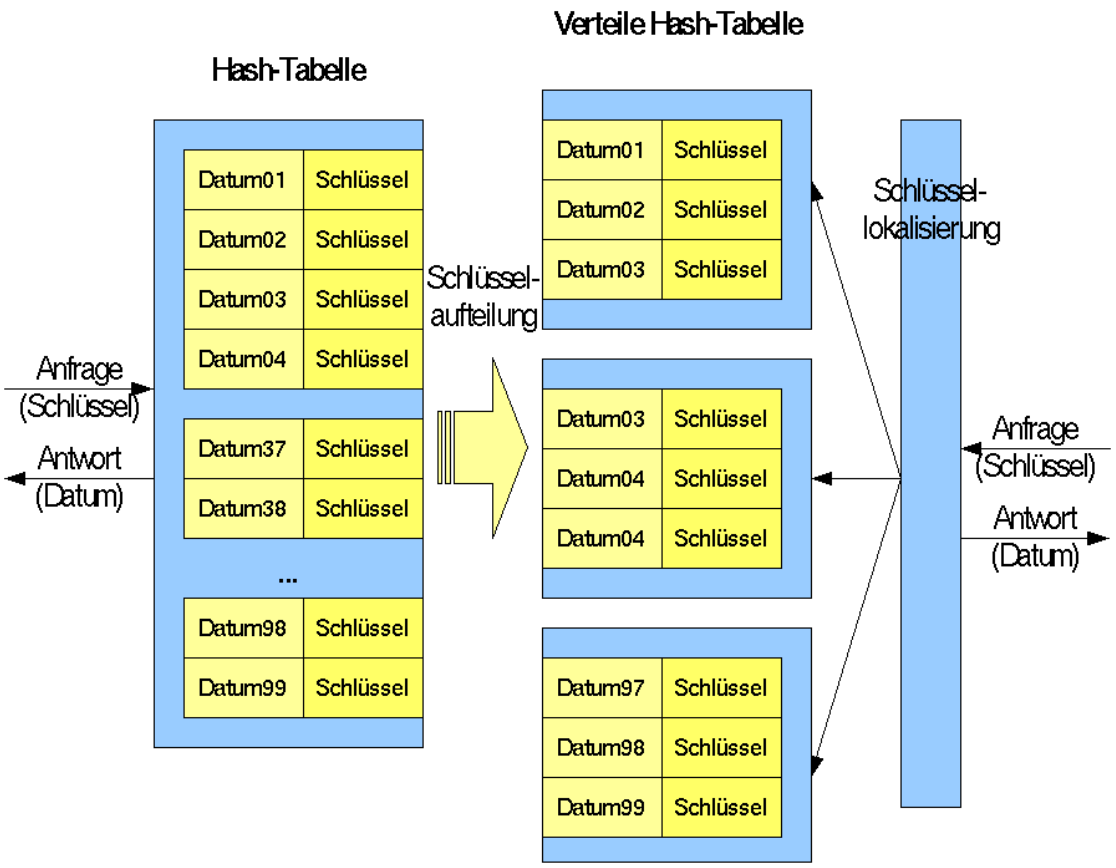

Abbildung 2: Struktur einer verteilten Hash-Tabelle

Sind diese Eigenschaften sichergestellt, so ist es nur noch ein kleiner Schritt, eine Nachricht an seinen adressierten Endpunkt dezentral weiterzuleiten. Nach dem Gready-Algorithmus[35] wird in jedem Schritt die optimale Lösung für das Weiterleitungsproblem, sprich die minimale Entfernung zu Zielschlüssel k, gesucht. Der Algorithmus bricht ab, sobald kein näherer Nachbar gefunden wird. Nach der zuvor definierten Eigenschaft muss die Nachricht nun bei dem, den Schlüssel k enthaltenen Knoten angekommen sein. Diese Art der Weiterleitung wird auch als schlüsselbasiertes Routing (key-based routing) bezeichnet.

Größe der Weiterleitungstabelle und Pfadlänge Über die grundlegende Richtigkeit des Weiterleitungsalgorithmus hinaus, bestimmen zwei Parameter die Leistungsfähigkeit des Systems. Je nach deren interner Topologie sind unterschiedliche Pfadlängen und Tabellengrößen notwendig. Natürlich sollte die Zahl der zu überwachenden Knoten, als auch die Anzahl der Weiterleitungsschritte, welche maßgeblich den Aufwand einer Reorganisation bei neuen oder verschwundenen Knoten beeinflusst, möglichst gering sein. Jedoch kann eine zu klein gewählte Nachbarschaftsgröße die Ausfallsicherheit der verteilten Hash-Tabelle untergraben.

Es ist also ein Kompromiss, wie die Parameter gewählt werden. Die gebräuchlichsten sind folgend aufgelistet:

Knotengrad	Pfadlänge
O(1)	$O(log(n))$
$O(log(n))$	$O(log(n)/loglog(n))$
$O(log(n))$	$O(log(n))$
$O(log(n^{1/2}))$	$O(log(1))$

Die dritte Variante ist die am häufigsten verwendete, da sie meist die ausgewogenste Konstellation in Bezug auf Tabellengröße und Weiterleitungsschritten darstellt. Ein weiterer Grund ist die verbesserte Flexibilität bei der Wahl der zu überwachenden Nachbarknoten. In Abschnitt 3.4 wird gezeigt, dass auch das vierte Verhältnis seine Berechtigung hat.

Um die Größe der Tabelle zu minimieren, werden Intervalle von Schlüsseln eingeführt. Besitzt ein Knoten einen angefragten Schlüssel, so kann er das Objekt bereitstellen. Anderenfalls muss ein mit einem Knoten assoziiertes Schlüsselintervall ausgewählt werden, welches näher am geforderten Schlüssel liegt.

2.3 Konkrete verteilte Hash-Tabellen

Ausgehend von den zuvor vorgestellten allgemeinen Eigenschaften wird im Folgenden ein solches System mit JXTA näher betrachtet werden. Da der als Basis dienende Nachrichtendienst in der Programmiersprache Java realisiert wurde, sind die hier vorgestellten konkreten Gleiche-zu-Gleiche Netzwerke ausgewählt worden, da für diese eine JAVA-Implementierung existiert. Da für eine prototypische Implementierung eine Pastry-API realisiert werden soll, werden in diesem Abschnitt lediglich Hintergründe, Erweiterungen und Implementierungen von Pastry vorgestellt. Eine tiefgehendere Betrachtung dieses Gleiche-zu-Gleiche Systems befindet sich dann in Abschnitt 3.1.

2.3.1 JXTA

Das von Sun Microsystems im Jahre 2001 ins Leben gerufene Open-Source Projekt JXTA definiert lediglich die XML-Protokolle und bietet somit eine plattform- und netzwerkunabhängige Spezifikation eines Gleiche-zu-Gleiche Netzwerks. Der angedachte Einsatzbereich soll vom Mobiltelefon bis hin zu normalen Standard-PCs reichen. Implementierungen liegen für verschiedene Programmiersprachen und Umgebungen vor, von denen JXTA2SE, als Java- und jxta-c, als C-Variante die wichtigsten beiden darstellen. Aufgrund seiner Portabilität und seiner ausgefeilten Struktur ist es eines, der wohl am weitestentwickelten Gleiche-zu-Gleiche-Systeme und soll hier in über-

blickender Art dargestellt werden. JXTA besteht aus üblichen Grundkomponenten, wie Knoten, Verbindungen und Nachrichten, die teilweise in verschiedenen Abwandlungen vorhanden sind.

Peer-Kategorien Das System unterscheidet vier Arten von Gleichartigen, so gennante „Edge Peers", welche sich, bedingt durch eine geringe Verbindungsbandbreite bildlich gesehen am Rand des Netzwerks befinden. Die andere Kategorie wird als „Super Peers" bezeichnet und lässt sich wiederum in „Rendevous Peers" und „Relay Peers" einteilen.

Ein Rendevous-Knoten ist ein spezialisierter Knoten, welcher sich um die Verwaltung der Knoten kümmert und somit die Erreichbarkeits- und Weiterleitungsinformationen besitzt. Eine vermittelnde Rendevous-Einheit wird immer benötigt, sobald sich zwei „Edge Peers" in unterschiedlichen Subnetzwerken befinden. Das sogenannte Relay-Peer erlaubt die Kommunikation durch Firewalls und Netzwerkaddressübersetzungssysteme (NAT) hinweg mit dem JXTA Netzwerk in Verbindung zu treten.

Peer Gruppen Peer Gruppen (Peer Groups) regeln den Bereich, in dem Nachrichten verbreitet werden. Jeder Teilnehmer des Netzwerks ist zumindest Teil der „NetPeerGroup" und kann zu beliebig vielen weiteren Gruppen gehören. Eine Gruppe benötigt mindestens einen gemeinsamen Treffpunkt-Knoten (Rendevous Peer).

Advertisements Ein Advertisement (Angebot) ist ein XML-Dokument, welches beliebige Ressourcen beschreiben kann.

Pipes Pipes (Rohre) sind virtuelle Kommunikationskanäle, welche einer Übermittlung von Meldungen und Daten dienen. Die Kanäle arbeiten asynchron und nur in einer Richtung, ebenso verfügen sie über keine Konsistenzprüfung und liegen in drei Varianten vor, von denen die „Unicast Pipe" die eben beschriebenen Eigenschaften erfasst. Eine spezielle „Unicast Secure Pipe" bietet zusätzlich eine gesicherte Variante, wohingegen eine „Propagate Pipe" eine multicast, d.h. eine 1-zu-N-Kommunikation, zur Verfügung stellt.

Protkolle JXTA definiert für jede, vom Überlagerungsnetz benötige Funktion ein spezielles Protokoll:

- Peer Resolver: Zum Senden und Empfangen generischer Anfragen.
- Peer Information: Liefert auf Anfrage Informationen über ein Peer.
- Rendezvous: Dient der Verbindung mit einem „Rendevous Peer".
- Peer Membership: Regelt Gruppenzugehörigkeit.
- Pipe Binding: Verknüpft so genannte Pipes (Nachrichtenkanäle) mit einem Ziel (endpoint).
- Endpoint Routing: Erzeugt zwischen zwei Peers eine Weiterleitung.

MyJXTA - Eine Anwendung Diese Applikation bietet Möglichkeiten zum Austauch von Dateien und Konferenzen in Chat-Räumen oder in Bild und Ton an.

2.3.2 Pastry

Pastry gehört zu den reinen Gleiche-zu-Gleiche Netzwerken, denn es verfügt über keinerlei zentrale Komponente und basiert auf den vorgestellten Prinzipen der verteilten Hash-Tabellen. Ein effizienter und robuster Weiterleitungsalgorithmus regte eine Vielzahl, meist wissenschaftliche Erweiterungen oder Anwendungen an. Das von Microsoft ins Leben gerufene Pastry-Projekt definiert die Anforderungen, des in dieser Arbeit zu realisierenden P2P-Systems. Aus diesem Grund sei auf den folgenden Abschnitt 3.1 verwiesen, in dem die Funktionsweisen und Strukturen beschrieben

werden. An dieser Stelle folgt eine Zusammenstellung von Projekten, welche Pastry implementieren oder darauf aufbauen.

Erweiterungen der Pastry-Transportschicht Aufbauend auf einer Transportschicht, welche sich um die Schlüsselaufteilung mit anschliessender Lokalisierung kümmert, existieren erweiternte Protokolle, die hier vorgestellt werden.

- Scribe[30] Ein Nachrichtensystem mit Publiziere-Abonniere-Prinzip.
- PAST[28] Diese Erweiterung macht Pastry zu einer verteilten Hash-Tabellen im formalen Sinn und lässt sich für temporäre und persistente Speicherung innerhalb der Pastry-Middleware einsetzen.
- Squirrel[21] Ebenso wie PAST handelt es sich um einen verteilten Datenspeicher, welcher aber als eine Proxy-Lösung konzipiert ist.
- SplitStream [14, 15] Dieses Protokoll maximiert den Fluß von Ressourcen mit Hilfe einer Rundruf-Emulation. Auf diese Weise soll die Transferbandbreite durch Kooperation verbessert werden.

Pastry Implementierungen

- FreePastry Als Ausgangspunkt für die folgenden Überlegungen wird die quelloffenne Java-Implementierung gewählt. FreePastry benutzt für seine Kommunikation einzelne TCP-basierte Punkt-zu-Punkt Verbindungen. Eine Ausnahme stellt hier die Entfernungsmessung im physikalischen Netzwerk dar, welche mittels UDP basiertem Rundruf durchgeführt wird. Da die FreePastry-Programmbibliothek die Schnittstellen für dieses Projekt definiert, wird in 4.1 eine genaue Betrachtung des Systems folgen.
- OverQoS[34] Diese Realisierung ist speziell für weiche Echtzeitanforderungen konzipiert und zielt, ähnlich wie die SplipStream-Protokollerweiterung auf Anforderungen an die Qualität eines Dienstes.
- Overcast[20] Ebenso wie das zuvor genannte Projekt wird in diesem Fall speziell auf eine Einer-zu-Viele Kommunikation wert gelegt und sucht ebenfalls eine Flussmaximierung über seine Teilnehmer wie in SlipStream.

Auf Pastry aufbauende Projekte Als konkrete Erweiterungen der Pastry-Middleware sind weitere Projekte entstanden, die im Weiteren kurz angesprochen werden.

- PASTA (Computer Laboratory, University of Cambridge) Stellt ein verteiltes Dateisystem, ähnlich zu PAST zur Verfügung, mit dem Unterschied, das sich die Daten mittels eines dezentralen, hierarchischen Namensraum ansprechen lassen[11].
- Das Herold Projekt (Microsoft Research, Redmond) Ein auf die größe des Internet skalierbare Anwendung, die mit Hilfe einer Publiziere-/Abonniere-Logik einen Benachrichtigungsdienst anbietet[?].
- Pastiche[17] Dieses an der Universität von Michigan entstandene Projekt ermöglicht ein Backupsystem mit Lastverteilung.
- Das DPSR Projekt stellt eine Verbindung zum Bereich der mobilen, sich schnell ändernden Netzwerke her und ist an der Purdue Universität, West Lafayette entstanden.
- XDM Dieses Projekt der Telecom Italia stellt eine verteilte XML-Datenbank zur Verfügung. Neben Suchanfragen an das sich selbst replizierende Sytem, wird ebenso eine Publiziere-/Abonniere-Schnittstelle angeboten.

3 Voraussetzungen einer Gleiche-zu-Gleiche API auf dem EventChannelNetwork

Da Pastry als P2P-Plattform für eine Implementierung mit dem EventChannelNetwork vorgesehen ist, werden im Folgenden die Struktur der Knoten, die verwendeten Lokalisierungsalgorithmen, sowie die verwendeten Protokolle beschrieben. Da die Schnittstelle des EventChannelNetwork als Basis für die Neuimplementierung für Pastry vorgesehen ist schließt sich eine Beschreibung des asynchronen Nachrichtendienstes an.

3.1 Pastry

Pastry bietet eine Transportschicht für Gleiche-zu-Gleiche Anwendungen, die erst durch zusätzliche Protokolle (siehe Abschnitt 2.3.2) bestimmte höhere Aufgaben, wie beispielsweise Lade-/Speicher-Funktionalitäten anbieten kann. Im Folgenden wird diese Transportschicht beschrieben, die sich um die Bekanntschaften der Knoten untereinander (siehe Abschnitt 3.1.1) kümmert. Ebenso findet in dieser Schicht die Weiterleitung der Nachrichten (Abschnitt 3.1.2) über die resultierende Netzwerktopologie hinweg statt. Da Pastry für ein dynamisches Beitreten und wieder Verlassen von Knoten konzipiert wurde, sind in Abschnitt 3.1.3 - 3.1.4 diese Fälle, zusammen mit ihren Auswirkungen auf den Zustand der beteiligten Knoten beschrieben.

Ein Knoten im Pastry-Netz wird standardmäßig durch einen 128bit langen Schlüssel eindeutig adressiert. Dieser Schlüssel (bzw. die Knoten-Id) wird einem Knoten beim Betreten des Netzes in zirkulärer Weise zwischen 0 und $2^{128} - 1$ zugewiesen. Die Zuteilung erfolgt zufällig, so dass Knoten mit aufeinander folgenden Schlüsseln möglichst gleichmäßig über das zugrundeliegende physikalische Netzwerk verteilt sein sollen. Durch eine Splittung des Schlüssels in 2^b Teileinheiten lassen sich auf einfache Weise Intervalle im Schlüsselbereich darstellen. Diese Tatsache wird für Platzersparnis ebenso, wie für die Nachrichtenweiterleitung genutzt. Erhält ein Knoten eine Nachricht, so sendet er diese an den Knoten weiter, der mind. eine Schlüsselstelle mit der Zieladresse mehr gemeinsam hat. Somit lässt sich eine durchschnittliche Anzahl der Weiterleitungsschritte von $O(Log_{2^b} N)$ erreichen und bedingt durch die Anzahl der Stellen, die zu einem Teilschlüssel zusammengefaßt werden, eine verbesserte Skalierbarkeit des Systems. Die Pastry-Weiterleitungstabelle ist in drei Kategorien unterteilt, die die Netzwerktopologie mitbestimmt.

3.1.1 Der Pastry-Knoten

Der Zustand eines Pastry-Knotens wird in drei Listen erfasst. Diese Listen enthalten die Knoten, die vom lokalen überwacht werden. Jeder Knoteneintrag ist mit Verbindungsparametern verknüpft. Wichtig ist bei der folgender Beschreibung, zwischen einer tatsächlichen physikalischen Entfernung und einer überlagernden Distanz innerhalb des Schlüsselraums zu differenzieren. Bei einer Betrachtung der in Abbildung 3 enthaltenen Knoten von oben nach unten verringert sich der physikalische Abstand zum betrachteten Knoten. In umgekehrter Richtung erfogt eine Annäherung an den numerischen Wert eigenen des Knotenschlüssels.

Die folgende Auflistung der Teiltabellen findet in der Reihenfolge, wie sie in Abbildung 3 gewählt wurde, statt.

- Leafset - Die Menge der Blattknoten

  Der Abschnitt der Tabelle, der die am weitesten über das Netzwerk verteilten Knoten enthält, wird als Menge der Blattknoten (Leafset) bezeichnet und wird mit den Knoten befüllt, die zur eigenen Knoten-ID die geringste Differenz aufweisen. Dabei werden die Schlüssel in ab- (SMALLER) und aufsteigender (LARGER) Reihenfolge in die 2 Blattknotenlisten eingefügt. Die typische Anzahl der in der Liste enthaltenen Blattknoten ist 2^b oder 2^{b+1}. Eine Erklärung für diese Zahlen wird im anschließenden Absatz gegeben.

- Routeset - Die Weiterleitungsknoten:

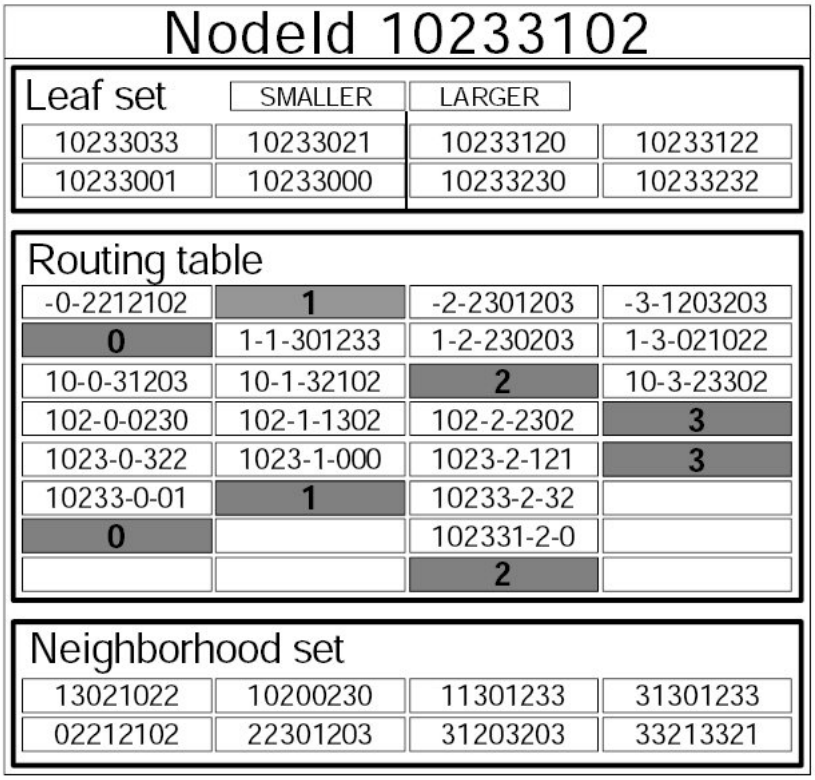

Abbildung 3: Die Tabellen eines Pastry-Knotens[26] hier für b=2,L=8

Die Weiterleitungstabelle enthält $Log_{2^b} N$ Zeilen mit jeweils $(2^b - 1)$ Einträgen. Durch einen Vergleich des lokalen Knotenschlüssels mit einem entfernten, ergibt sich eine Anzahl gemeinsamer Schlüsselstellen, die die Plazierung innerhalb dieser Weiterleitungstabelle bestimmt. Das bedeutet, dass eine entfernte Knoten-ID, die 3 gemeinsame Stellen mit der lokalen ID besitzt in der 3. Zeile der Tabelle abgelegt wird. Die Weiterleitungstabelle nutzt die Eigenschaften der partitionierbaren Schlüssel aus und spannt somit ein Intervall von Knoten-IDs auf, die über den enthaltenen Knoten erreicht werden können.

- Neighborhood set - Die Nächsten Nachbarn

  Diese Knoten befinden sich in unmittelbarer (physikalischer) Nähe des betrachteten Knotens und wurden aufgrund ihrer kurzen Antwortzeiten (Pingzeiten) in diese Liste aufgenommen. Wegen der in Pastry geltenden Annahme, der gleichmäßig über das Netz verteilten Schlüssel, befinden sich in dieser Tabelle die am weitest gestreuten Schlüsselwerte.

3.1.2 Das Routing

Die Weiterleitung von Nachrichten innerhalb der einzelnen Knoten erfolgt mit einem Algorithmus, welcher in Abbildung 4 zu sehen ist. Auf eine abstrakte Beschreibung des Verfahrens folgt eine Erläuterung des in der Abbildung gezeigten Verfahrens.

Im Mittelpunkt dieses Algorithmus steht eine Entfernungs- und eine Auswahlfunktion. Erstere berechnet die Distanz zwischen zwei Schlüsseln. Anzumerken für die folgende Betrachtung des Algorithmus ist, dass es sich bei den genannten „Entfernungen" immer um die Entfernung im Schlüsselraum handelt. In Pastry ist diese Entfernung auf Basis der gemeinsamen Schlüsselstellen definiert. Die Auswahlfunktion wählt aus einer Menge von Schlüsseln denjenigen aus, der die meisten gemeinsamen Stellen mit dem in der Nachricht adressierten Zielschlüssel besitzt.

Durch dieses Lösungsprinzip gehört das Weiterleitungsverfahren zu den Gready-Algorithmen. In jedem Weiterleitungsschritt wird untersucht, ob die Nachricht für den Knoten selbst, oder für einen im Blattwerk enthaltenen Knoten bestimmt ist. Ist dies nicht der Fall, d.h. der weiterleitende Knoten kennt den Adressierten nicht, sucht er innerhalb der Weiterleitungstabelle den Knoten aus, dessen Schlüssel möglichst viele gemeinsame Stellen besitzt.

Damit die Nachricht sich nicht wieder weiter von ihrem Ziel entfernt, muss die Anzahl der übereinstimmenden Schlüsselstellen des Weiterleitungsknotens mindestens der Anzahl der gemeinsamen Stellen zum lokalen entsprechen.

(1) if ($L_{(-L/2)} \leq D \leq L_{(L/2)}$){
(2) // D ist im Blattwerk enthalten
(3) leite D an L_i weiter, mit min$|D - L_i|$
(4) } else {
(5) // benutze die Weiterleitungstabelle
(6) l = sharedPrefix(D,A)
(7) if ($R_l^{D_l} \neq null$){
(8) forward D to $R_l^{D_l}$
(9) } else {
(10) leite an $T \in (L \bigcup R \bigcup M)$, welcher
(11) sharedPrefix(D,A)$\geq l$
(12) $|T - D| < |A - D|$
(13) }
(14) }

Abbildung 4: Der Pastry-Weiterleitungsalgorithmus[29]

Der zugrundeliegende Algorithmus ist in Abbildung 4 dargestellt. Dieser behandelt auf einem Knoten mit Schlüssel A jede, mit einem Schlüssel D versehene Nachricht. L_i bezeichnet einen Eintrag des Blattwerks L. Ein Eintrag in der Weiterleitungstabelle hat folgende Form: R_l^i, wobei $0 \leq i < 2^b$ dessen Zeilen und $0 \leq l < 128/b$ die Spalten indizieren. Die Entfernungsfunktion „sharedPrefix(ID1,ID2)" bestimmt die Anzahl der gemeinsamen Stellen des Knoten-Schlüssels. D_n stellt den Wert der n-ten Stelle innerhalb des Schlüssels dar.

Unter der Annahme, dass sich die Weiterleitungstabelle in einem konsistenten Zustand befindet, das heißt, das keine ausgefallenen Knoten enthalten sind, kann gezeigt werden, dass die erwartete Anzahl an Weiterleitungsschritten (hops) $log_{2^b}(N)$ ist. Dafür sind die drei Fälle des Algorithmus zu begutachten. Im besten Fall wird der angeforderte Schlüssel in der Menge der Blattknoten gefunden, was bedeutet, dass die Nachricht nur noch einen Schritt von ihrem Ziel entfernt ist (Zeile 1-3). Ist dies nicht der Fall, wird in den Zeilen 6 - 8 zunächst die Anzahl der gemeinsamen Schlüsselstellen und damit die Distanz im ID-Raum zum Zielknoten berechnet. Da nun bei jedem Schritt eine weitere übereinstimmende Stelle mit dem Zielschlüssel erreicht wird, reduziert sich sowohl der Anteil der noch ungleichen Stellen, als auch die Zahl der damit möglichen Weiterleitungsschritte. In Zeile 10-12 schliesslich wird der Fall behandelt, dass der Schlüssel nicht im Blattwerk enthalten ist, obwohl die Nachricht nur noch einen Schritt von seinem Ziel entfernt sein sollte. Geht man von konsistenten Tabellen aus, so wird der Schlüssel als nicht existent interpretiert.

Durch mehrere Ausfälle aufeinanderfolgender Knoten des Blattwerks kann sich der Pfad zum Ziel auf O(N) verlängern. Erst bei einem Ausfall von $\lfloor |L|/2 \rfloor$ Knoten mit aufeinanderfolgenden Schlüsseln bringt das System zum Erliegen. Aufgrund der Annahme, dass die Blattwerksknoten gleichmäßig über das zugrundeliegende Netz verstreut sind und eine geeignete Größe des Blattwerks gewählt wird, kann das eintreten dieses Falls minimiert werden. Experimentelle Untersuchungen haben gezeigt, dass diese mit $|L| = 2^b$ (b ist die Anzahl der vereinigten Schlüsselstellen) eine Ereigniswahrscheinlichkeit von 2% haben. Verdoppelt man die Größe der Tabelle auf $|L| = 2^{b+1}$, so sinkt diese Wahrscheinlichkeit auf 0.6%. Bevor der Ausfall eines Knotens im über-

nachsten Abschnitt näher beleuchtet wird, folgt zuvor eine Beschreibung des Knotenbeitritts und der Erweiterung des Pastry-Netzwerks.

3.1.3 Das Beitrittsprotokoll

Möchte ein neuer Knoten ein schon vorhandenes Pastry-Netz betreten, so kontaktiert er einen, vor dem Start der Applikation, schon bekannten Einstiegsknoten. Die erste Nachricht, mit der der Knoten den Initialisierungsknoten kontaktiert ist eine `NodeIdRequestMessage`, mit welcher die ID des Knotens für eine weitere Kommunikation ausgehandelt wird. Die Beantwortung geschieht mit Hilfe der `NodeIdResponseMessage`, welche den angeforderten Schlüssel enthält. Um die nun folgende Anzahl an Nachrichten zur Initialisierung der Knotentabellen zu minimieren und aufgrund der Annahme, dass der Einstiegsknoten in dessen (physikalischen) Nähe liegt, wird der neue Knoten mit dem Blattwerk (`LeafsetRequestMessage`,`LeafsetResponseMessage`) und den Weiterleitungstabellen (`RouteRowRequestMessage`,`RouteRowResponseMessage`) initialisiert.

Daraufhin kann das eigentliche Beitrittsprotokoll (JoinProtocol) gestartet werden, welches nach Abschluss den Knoten zu einem gleichberechtigtes Mitglied des Netzwerks macht. Auch nach einer Verbindungsunterbrechung muß dieser Vorgang wiederholt werden, damit die Konsistenz der Weiterleitungstabelle und des Netzes gewahrt bleibt.

Ein beitretender Knoten X versendet eine mit seinem Schlüssel S_1 versehene Beitrittsanforderung („Join") an den, ihm aus der zuvor beschriebenen Initialisierung schon bekannten Knoten A. Da A den neuen Knoten noch nicht kennt, bzw. in seinen Tabellen enthält, leitet er dessen Beitrittsanforderung über mehrere Mittelsknoten bis an den Knoten, dessen Schlüssel S_2 am nähesten zu S_1 liegt. Die während dieser Weiterleitung besuchten Knoten senden als Antwort ihre Statusinformationen, welche, wenn passend, die Tabellen des neuen Knotens aufbauen. Eventuelle Lücken werden durch zusätzliche Einzelanfragen beseitigt. Da der Knoten, mit Schlüssel S_2 der Schlüssel mit geringster Entfernung zu S_1 ist, dient das Blattwerk von Knoten S_2 dem neuen Knoten als Ausgangsblattwerk. Sind die internen Strukturen vollständig befüllt, sendet der neue Knoten seine Statusinformationen an alle Knoten, die er kennt und stellt sich somit vor. Auf diese Art wird auch das Überlagerungsnetzwerk über die Existenz eines neuen Knotens informiert und integriert den neuen Knoten in die Tabellen der schon bestehenden Knoten. Die Kosten einer Einfügeoperation betragen somit $O(log_{2^b} N)$ oder in Anzahl der Nachrichten: $3 * log_{2^b} N$.

3.1.4 Knotenausfälle

Ein Knoten wird als fehlerbehaftet bezeichnet, wenn ihn seine Blattwerksnachbarn (im ID-Raum) nicht mehr erreichen können. Um die Blattknotentabelle zu reparieren, wird bei einem weiteren Blattknoten dessen Liste der Blattknoten angefordert, um die enthaltenen Knoten in die eigene zu integrieren. Dabei wird ein Knoten aus der Blattliste (ab-, oder aufsteigend) ausgewählt, in der auch der ausgefallene Knoten enthalten war. Somit kann das Blattwerk als Ausfallkriterium für das Netzwerk angegeben werden, da es solange stabil ist, so lange nicht $|\frac{L}{2}|$ aufeinanderfolgende Knoten der Blattknotentabelle ausgefallen sind. Da der ausgefallene Knoten in L Blattwerkstabellen der übrigen Knoten überwacht wurde, sind nun Reparaturkosten in Form von L Nachrichten nötig, um das Netzwerk konsistent zu halten. Die Weiterleitungstabelle wird nur im Fall einer Weiterleitungsanfrage repariert. Die nächsten Nachbarn im Ping-Raum hingegen sind auf eine ständige gegenseitige Überwachung angewiesen und prüfen sich in regelmäßigen Abständen. Ist ein Knoten nicht mehr erreichbar, so wird der Schlüssel in die Tabelle eingefügt, welcher den nächstgeringsten Abstand im physikalischen Netzwerk besitzt.

3.2 Der Echtzeitnachrichtendienst

Durch einen steigenden Leistungsbedarf bei sicherheits- oder geschäftskritischen Systemen wird eine Aufteilung der Verarbeitung auf Komponenten notwendig. Damit eine Entwicklung komplexer und ausfallsicherer Applikationen möglich ist, wird versucht, objektorientierte Ansätze, die eine besseren Wiederverwendbarkeit und einfachere Wartung anbieten, in diesem Anwendungsbereich

nutzbar zu machen. Die für sicherheitskritische Systeme notwendigen Echtzeitgarantien erweitern sich in verteilten Systemen auf die verwendeten Netzwerke. In diesem Zusammenhang wird auch das EventChannelNetwork (ECN, asynchroner Nachrichtendienst) entwickelt, das im Hinblick auf den Einsatz in eingebetteten Systemen auch mit den dort vorkommenden Netzwerken umgehen kann. Diese Netzwerke arbeiten meist nach dem Rundruf- oder dem Feldbusprinzip. Das EventChannelNetwork ist in Java nach den RTSJ-Spezifikationen (Real-Time Specification for Java[12]) implementiert und erhält damit die Echtzeiteigenschaften der zugrundeliegenden Kommunikationsnetzwerke. Der Nachrichtendienst stellt eine asynchrone Publiziere/Abonniere-Kommunikation zur Verfügung. Dabei werden Nachrichten nach einem direkten "PushÜbertragungsmodell versendet. Die Asynchronität bedeutet, dass nicht auf einen Rückruf gewartet und damit Ressourcen belegt werden. Darüber hinaus findet eine Entkoppelung von Sender und Empfänger statt. Im weiteren wurde darauf geachtet, dass ein möglichst geringer Overhead durch die Verwaltung des asynchronen Nachrichtendienstes hervorgerufen wird.

Im Folgenden werden zunächst die dem EventChannelNetwork zugrundeliengen Strukturen und deren Interaktion beschrieben. Daran anschliessend werden generelle Betrachtungen zur asynchronen und rundrufbasierten Kommunikation dargelegt.

3.2.1 Der Aufbau des EventChannelNetwork

Die zentrale logische Komponente des Nachrichtendienstes stellt der Nachrichtenkanal (EventChannel) dar. Dieser Nachrichtenkanal ist mit Eigenschaften, wie zum Beispiel einer Priorität, versehen. Diese Eigenschaften regeln die Weiterverarbeitung der mit dem Nachrichtenkanal verknüpften Komponenten, wie Sender (Transmitter), Empfänger (Receiver) oder der Überbruckungseinheit (Gateway). Eine Darstellung des Nachrichtenkanals ist in Abbildung 5 zu sehen.

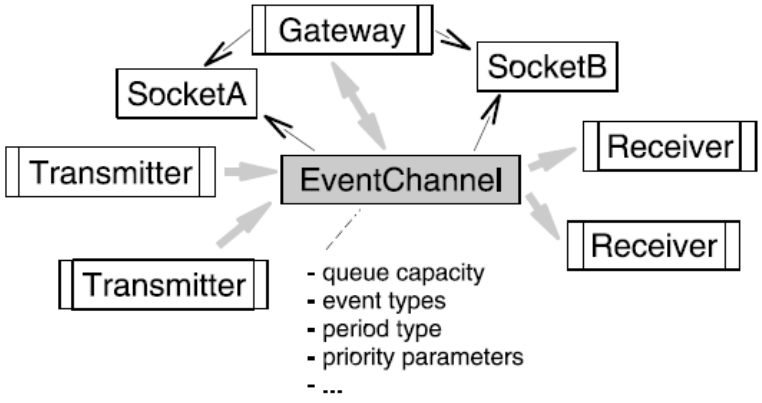

Abbildung 5: Der EventChannel[31]

Ein höchstprioritärer Kontrollfaden hat die Aufgabe, die asynchron entgegengenommenen Nachrichten in ein Warteschlangensystem einzufügen. Dieses Warteschlangensystem besteht aus einzelnen prioritätsbehafteten und die Reihenfolge erhaltenden Listen (FIFO, First-in-first-out). Damit das Warteschlangensystem keine Nachrichten wegen Überfüllung unterschlägt, ist entweder auf eine ausreichende Puffergröße, oder eine entsprechend schnelle Abarbeitung der enthaltenen Nachrichten notwendig. Für die genannte Weiterverarbeitung, der im Warteschlangensystem vorhandenen Nachrichten ist ein Aktivitätsverwalter (ActivityManager) zuständig. Dieser hat die Aufgabe, die zu behandelnden Anfragen an Kontrollfäden mit entsprechenden Systemattributen (Priorität) zuzuordnen. Auf diese Weise können definierte Behandlungs-, bzw. Antwortzeiten garantiert werden. In Abbildung 6 sind die beschriebenen Komponenten dargestellt.

Nachdem nun das EventChannelNetwork vorgestellt wurde, sollen die, für dieses System zugrundeliegenden Kommunikationsprinzipien untersucht werden. Diese Betrachtungen sind grund-

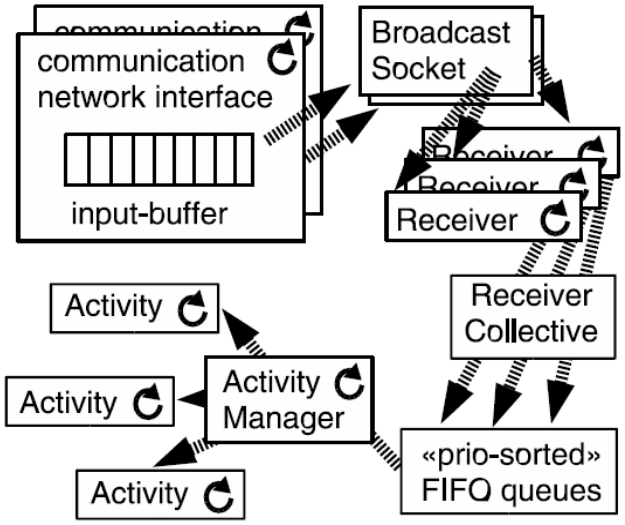

Abbildung 6: Interaktion beim Nachrichtenempfang[31]

legend für den Bau des Gleiche-zu-Gleiche-Netzwerks, wie es in dieser Studienarbeit beschrieben wird.

3.3 Die Asynchrone Kommunikation

Eine asynchrone Nachrichtenkommunikation hat gegenüber der synchronen Variante den Vorteil, dass diese blockierungsfrei arbeitet. Zwischen dem Versenden und Empfangen von Botschaften kann die beteiligte Ressource (Netzwerk, Rechner) für anderen Aufgaben genutzt werden. In der Anwendungsebene ist ein dienstnehmender Kommunikationsknoten meist an ein Anfrage-Antwort-Muster gebunden. Bevor diese Probleme in Angriff genommen werden, soll zunächst noch das Rundrufprinzip untersucht werden, welches nur unter geschickter Benutzung auch dessen Vorteile zur Geltung bringt.

Da durch eine asynchrone Kommunikation eine Entkoppelung von Sender und Empfänger stattfindet, kann auch von einer anonymen Kommunikation gesprochen werden. Diese besitzt den Vorteil, dass im Zusammenhang mit einer Anforderung die behandelnde Komponente nicht adressiert wird und damit unabhängig von deren möglichen Ausfall ist. Einer direkten Übernahme durch eine weitere Komponente, die den gleichen Dienst anbietet wird auf diese Art ermöglicht.

3.4 Betrachtung der Rundruf-Kommunikation

Mit Hilfe der Rundrufkommunikation werden mit einer Nachricht alle sich im Netzwerk befindlichen Einheiten angesprochen. Damit ist der Aufwand für die Verbreitung einer Information nicht abhängig von der Anzahl seiner Empfänger, wie es bei einzelnen Punkt-zu-Punkt Verbindungen der Fall ist. Ein Aufwand bei Ausfall einer mehrfach im Netzwerk vorhandenen Ressource ist abgesehen, von einer Behandlung auf Anwendungsebene, praktisch nicht existent. Ebenso reduziert sich der Aufwand auf Seiten des Senders, der für eine Eine-zu-N-Kommunikation einem Aufwand von O(1) entspricht. Auf Seiten der Empfänger ist, durch die, den Knoten nicht selbst betreffenden Nachrichten, mit einer Steigerung des Datenverkehrs zu rechnen. Hinzukommt, dass ankommende Meldungen geprüft werden müssen, ob diese überhaupt angefragt, bzw. für den Empfänger interessant sind. Das ECN bietet hierzu eine Differenzierung durch die Wahl unterschiedlicher logischer Kanäle an. Damit ist es möglich spezifische, dienstbezogene Protokolle in unterschiedlichen, nicht interferierenden Kanälen zu definieren. Um keine Überflutung des Netzwerks oder Kettenreaktionen auszulösen, muss eine klare Zuständigkeit für die Beantwortung von Anfragen

definiert werden. Auch dürfen Anfragen nicht in zu schneller Folge hintereinander in das Netz hineingesendet werden.

Um eine anonyme Anfrage, die nur eine anonyme Antwort benötigt nicht mehrfach zu behandeln, werden im folgenden verschiedene Ansätze, dies zu erreichen vorgestellt. Eine Möglichkeit stellt das „Schnüffeln" dar. Hierbei wird ein Nachrichtenkanal bei Erhalt einer Anfrage belauscht, ob nicht eine Beantwortung durch einen anderen Knoten statt gefunden hat. Dadurch entsteht jedoch die Gefahr einer Blockierung (Deadlock). Eine Verbesserung lässt mit Hilfe zufälliger Wartezeiten erreichen, nach denen eine Antwort gesendet wird. Wurde in der Zwischenzeit jedoch schon eine Beantwortung mitgelauscht, so wird ein wiederholtes Senden unterlassen. Eine randomisierter Ansatz ist für den Aspekt der garantierten Antwortzeiten kontraproduktiv und darüber hinaus ist diese Lösung ebenfalls mit der Gefahr einer Blockierung behaftet. Ein weiterer Ansatz bedient sich der Annahme, dass drei Knoten innerhalb eines Netzwerks unterschiedliche große Latenzen aufweissen. Eine Empfangs-, bzw. Bearbeitungsbestätigung kann zurück gesendet werden, anhand derer alle Knoten im Netz über die Ungültigkeit der Anfrage informiert sind und von einer zusätzlichen Behandlung der Nachricht absehen.

Darüber hinaus gibt es aber noch weitere Anfrage-Antwort-Muster, bei denen zu einer Anfrage eine bestimmte Anzahl von Antworten benötigt werden. Gerade im Bereich der sicherheitskritischen Systeme werden Mehrfachberechnungen als Ergebnisabsicherung verwendet. Eine Bestimmung, welche, und wieviele Komponenten für eine Behandlung nötig sind, ist je nach Anwendung unterschiedlich. Im Folgenden werden die Vor- und Nachteile einer Rundrufkommunikation noch zusammenfassend aufgelistet.

Vorteile des Rundruf Prinzips:

- Einfache Ressourceübernahme durch anonyme Kommunikation: Dies Minimiert Möglichkeit eines Totalausfalls im Vergleich zur einer-zu-einer-Kommunikation und ermöglicht eine unterbrechungslose Übernahme durch einen Standby-Knoten.
- Innerhalb des Rundrufbereichs ist keine Weiterleitung nötig: Dies bedingt einen Aufwand O(1) für eine gesendete Nachricht. Da alle, im Überlagerungsnetz befindlichen Knoten die Nachricht mithören können, erhalten diese zusätzliche Informationen. Ein prophylaktisches Zwischenspeichern einer erhaltenen Nachricht macht mit einer bestimmten Wahrscheinlichkeit gewisse Anfragen unnötig.
- Kleinere Knotentabellen: Dadurch, dass keine Weiterleitungsschritte über O(log(n)) Knoten verwaltet werden müssen, sinkt nicht nur der Speicherbedarf, sondern auch die für deren Verwaltung benötigten Rechenkapazitäten.

Nachteile des Rundruf Prinzips:

- Zuständigkeitsentscheidung Dienstgeber müssen zunächst über ihre Zuständigkeit entscheiden, um anschliessend die, für den Dienstnehmer interessanten Informationen herauszusuchen und zu versenden.
- Erhöhtes Nachrichtenaufkommen auf der Dienstnehmerseite: In Nachrichten enthaltene Informationen müssen auf Relevanz für den Dienstnehmer geprüft werden. Dadurch ergibt sich ein erhöhter Rechenaufwand für den Knoten.
- Skalierbarkeitsgrenze in Rundruf-Netzwerken: Einer-zu-Einer Kommunikation in einem Rundrufnetzwerk belastet das gesammte Netzwerk. Selbst, wenn das Netzwerk für die durchschnittliche Nachrichtengröße, mit einer durchschnittlichen Häufigkeit verhergesagte Netzwerklast ausreichend ist, so können sich die Ereignisse kummulieren.
- Protokollfehler durch Nachrichtenüberlagerung: Bei steigender Größe des Netzwerks sind auch immer mehr potentielle Ausfallpunkte im Netz enthalten. Durch unvorhergesehene Anfrage-Antwork-Muster kann ein Deadlock oder eine Kettenreaktion auftreten.

4 Planung

Im folgenden Abschnitt wird der Grobentwurf der zu entwickelnden Bibliothek entstehen. Als Anforderungen definierende API wurde FreePastry in der Version 1.4.2 vom September 2005 gewählt, mit dem Ziel, auf dieser API basierende Anwendungen wiederzuverwenden. Um die Anbindung des RT-ECN mit möglichst geringen Refaktorisierungsaufwand vollziehen zu können, ist im ersten Schritt eine genaue Betrachtung der vorhandenen Bibliothek von nöten. Als Beispiel einer existierenden Netzwerkanbindung wird die TCP-IP überlagernde Socket-Variante (siehe Abschnitt 4.2) der Bibliothek vorgestellt.

Auf diesen Tatsachen aufbauend, lassen sich in Abschnitt 4.3 Gemeinsamkeiten und Unterschiede der beiden Netzwerkanbindungen herausarbeiten sowie mögliche Adaptionswege diskutieren. Bevor dieser Abschnitt in einer Beschreibung der Umsetzung mündet, folgt zuvor noch eine Zusammenstellung der Funktionen, welche für eine prototypische Implementierung nötig sind.

4.1 Die FreePastry-Programmbibliothek

Die FreePastry-Programmbibliothek (FreePastry-API), wurde an der Rice University, Texas, Houston als eine quelloffene Implementierung der Pastry-Spezifikationen zu Versuchs und Forschungszwecken realisiert. Neben der Pastry-Transportschicht sind auch die in Abschnitt 2.3.2 vorgestellten Erweiterungen implementiert. Abbildung 7 soll einen Überblick über enthaltenen Bibliothekspakete vermitteln, welche im einzelnen folgend beschrieben werden.

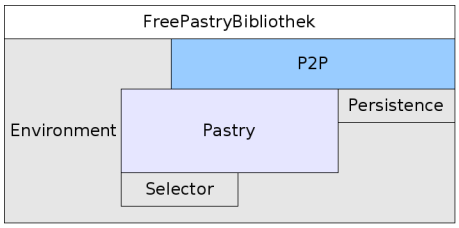

Abbildung 7: Die FreePastry-API

Environment (rice.environment) Dieses Paket stellt einige Werkzeuge bereit, wie beispielsweise ein Logging-Mechanismus, eine Zufalls- und eine Zeitquelle und einen Konfigurationshelfer, welcher durch das Unterpaket `rice.environment.parameters` bereitgestellt wird. Für eine Ablaufsteuerung steht ein Unterpaket `rice.environment.processor` zur Verfügung, welches eine absrakten Prozessor simuliert. Zwei parallele Prozessfäden kümmern sich unabhängig zum einen um die Abarbeitung blockierender Ein- und Ausgaben, zum anderen um die Ausführung sonstiger Arbeiten.

Selektor (rice.selector) Dieses Paket stellt einen erweitertes Observer-Muster bereit, welches einen bestimmten Nachrichtenempfänger (`MessageReceiver`) aufruft, sofern dieser unter einem entsprechenden `SelectionKey` registriert wurde. Eine weitere Aufgabe dieses Pakets ist das Socket-Lauschen nach neu eingetroffenen Nachrichten. In diesem Fall wird der zugehörige `MessageReceiver` aufgerufen, der die Nachricht dann weiterverarbeiten kann. Das Gegenstück zum Empfänger, dem `MessageDispacher` obliegt die lokale Weitergabe einer Nachricht. Mittels Aneinanderreihung mehrerer `MessageDispacher` und `MessageReceiver` lassen sind komplexe Zuständigkeitsketten definieren.

Persistence (rice.persistence) Implementiert verschiedene Varianten für Datenhaltungsfunktionen.

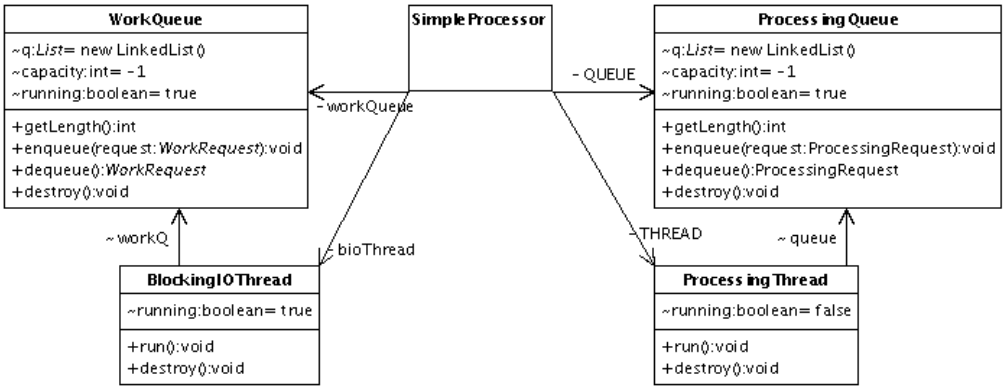

Abbildung 8: Die FreePastry - Prozessor Abstraktion

P2P (rice.p2p) Hier enthalten sind die im Abschnitt 2.3.2 genannten Erweiterungen, welche auf der Transportschicht aufbauen. Dies unterstützt die Unempfindlichkeit dieser speziellen Protokolle gegenüber Änderungen in `rice.pastry` , bzw. `rice.pastry.socket` .

Pastry (rice.pastry) Dieses Paket bildet die Transportschicht der P2P-Bibliothek. In Abb. 9 ist eine Ansicht der enthaltenen Unterpakete zu sehen. Die Anordnung der Pakete soll deren Abhängigkeiten untereinander darstellen und eine Reihenfolge der Erläuterungen vorgeben.

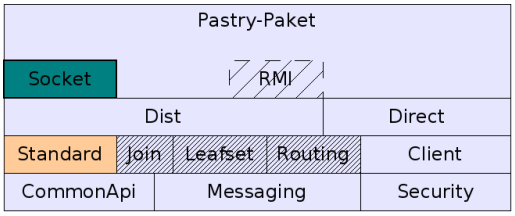

Abbildung 9: Die Pastry-Transportschicht: Eine Übersicht der enthaltenen Pakete

In der obersten Schicht befinden sich konkrete Implementierungen für die Anbindung spezieller Netzwerktypen. In den Paketen dieser Ebene findet die eigentliche Überlagerung des zugrundeliegenden Kommunikationssystems, im Fall des Pakets `rice.pastry.socket` handelt es sich um das Internet. Die schraffiert gekennzeichnete RMI ist seit Version 1.4.1 nicht mehr in der FreePastry-API enthalten und wird hier auch nur als Beispiel für eine weitere Netzwerkanbindung aufgeführt. Eine Schicht tiefer sind zwei Varianten verfügbar. Das Paket `rice.pastry.direct`dient lokalen Tests. Da sich ohne latenzbehafteter Netzwerkkommunikation keine Entfernungsfunktion auf den Pingzeiten definiert werden kann, steht eine Hilfsfunktion für Experimente zur Verfügung. Die abtrakte Implementierung im Paket `rice.pastry.dist` dient als Grundlage für netzwerkspezifische Realisierungen.

In der weiter abstrahierten Protokollschicht sind innerhalb der Pakete `Join`, `LeafSet`, `Routing` Strukturen und interne Funktionen auf diesen definiert, die für eine lokale Verwaltung der Knotenbeziehungen bestimmt sind. Im Zusammenspiel mit speziellen Nachrichtentypen, wie beispielseweise eine Blattwerksanfrage (`rice.pastry.leafset.RequestLeafSet`) wird auch die verteilte Verwaltung ermöglicht. Das Paket `rice.pastry.standard` definiert, ausgehend von den drei zuvor genannten Paketen bestimmte Standard- bzw. Grundprotokolle, welche sich zur Laufzeit an einem Knoten registriert werden. Diese beschreiben dann, zum Beispiel mit `StandardJoinProtocoll` den Verlauf eines Knotenbeitritts. Eine Variante des Beitrittsprotokoll ist mit `ConsistenJoinProtokoll` gegeben. Auf der gleichen Ebene ist das Paket `rice.pastry.client` zu finden. In diesem wird ein Endpunkt, bzw. eine Schnittstelle für die Anwendungsschicht definiert.

Die unterste dargestellte Schicht liefert mit `CommonAPI` diverse Abstraktionen, beispielsweise die eines Schlüssels (`ID`, `NodeId`), und einer zugehörigen Schlüssel-Fabrik-Klasse (`IDFactory`). Ebenso existiert das Paket `Messaging`, in dem zunächst eine abstrakte Nachrichtenklasse `Message` und darauf aufbauend das erweiterte Beobachtermuster, durch `MessageDispatch`und `MessageReceiver`, realisiert wird. Eine leere Klasse namens `Address` erlaubt es, bestimmte Interessensgruppen zu spezifizieren, indem ein Nachrichtenempfänger (`MessageReceiver`), zusammen mit einer Ableitung der Klasse `Address`, am `MessageDispatch`Object registriert werden.

Aufgrund der Tatsache, dass eine Schnittstelle zum RT-EventChannelNetwork auf die gleiche Art wie die IP-basierte Socket-Anbindung realisiert wird, erweist sich eine genauere Betrachtung des Teilpakets `rice.pasrty.socket` als hilfreiches Beispiel für das weitere Vorgehen.

4.2 Die Pastry-Socket-Kommunikationsschicht

Folgend wird die Socket-spezifische Implementierung innerhalb der FreePastry-Bibliothek vorgestellt, wobei das UML-Klassendiagramm in Abbildung 10 dazu dienen soll, einen Überblick über die enthaltenen Klassen zu geben, die diesem Abschnitt erläutert werden.

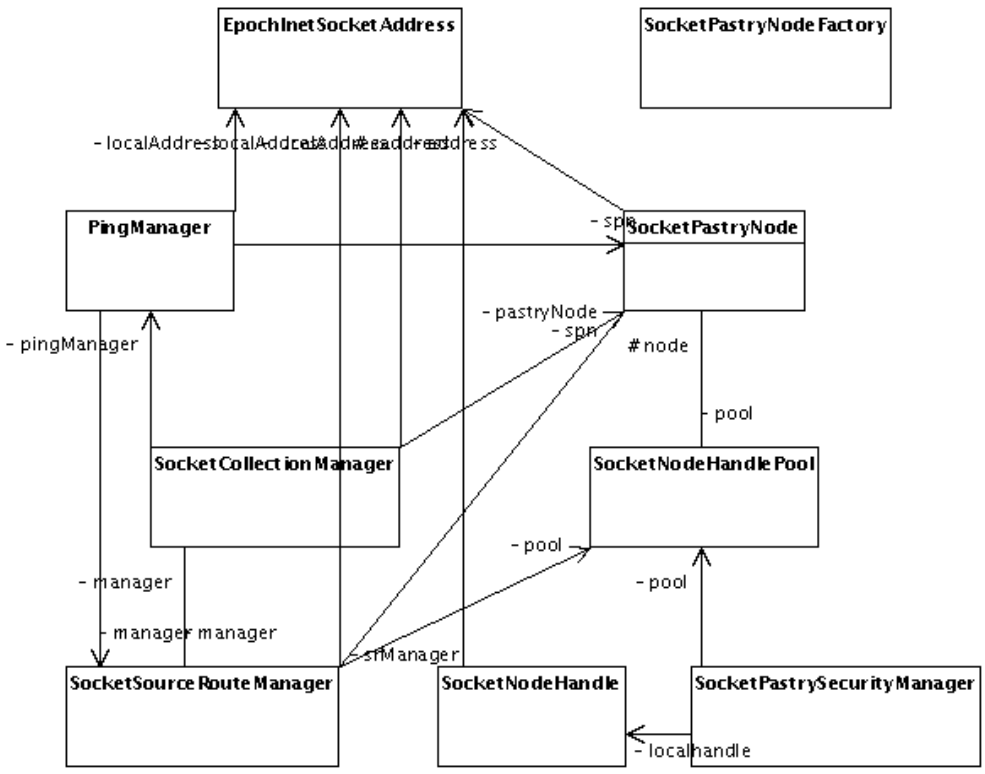

Abbildung 10: Die TCP-IP-Netzwerküberlagerung der FreePastry-Bibliothek

SocketPastryNodeFactory Diese Fabrikklasse stellt neben einer Erzeuger-Methode für Objekte der Klasse `SocketPastryNode` Methoden für einer synchrone Kommunikation nach dem Anfrage-Antwort-Muster. Diese werden benötigt, um vor eine Initialisierung, bzw. einem Beitritt eines neuen Knotens mit dem Pastry-Netzwerk zu kommunizieren.

SocketPastryNode Diese Klasse stellt einen konkreten Pastry-Knoten dar. Hier werden Beobachter-Muster (mit MessageDispatch und MessageReceiver) verknüft und das Blattwerk und die Weiterleitungstabelle aufbewahrt. Diese Klasse ist die zentrale Komponente des Pakets.

SocketNodeHandle Ein Platzhalter für entfernte Knoten wird von dieser Klasse bereitgestellt.

SocketNodeHandlePool Die Klasse besteht aus einer Menge von `SocketNodeHandle` Objekten, die die entfernten, vom `SocketPastryNode` überwachten Knoten darstellt.

SocketPastrySecurityManager Diese Klasse definiert nur einen abstrakten Sicherheitsverwalter, der mit Hilfe der Methode `verifyMessage` bestimmte Nachrichten zurückweisen kann. Ebenso stehen Methoden zur Prüfung der Absenderknoten und der Adresse zur Verfügung.

EpochInetSocketAddress Um das Tupel aus IP-Adresse und Port mit einem Knotenschlüssel zu verknüpfen existiert diese Klasse und bildet damit den Grundbaustein des Überlagerungsnetzwerks.

Die Verwaltung der Kommunikation ist in der linken Hälfte des Diagramms zu erkennen.

SocketSourceRouteManager Diese Verwalterklasse kümmert sich primär um die Weiterleitung von Nachrichten. Dies betrifft auch die Fälle, in denen es sich um lokal versendete Nachrichten handelt. Die Aufgaben des Sendens und des Empfangens werden an die Untergeordneten Klasse `SocketCollectionManager` dirigiert.

SocketCollectionManager Diese Klasse schliesslich verwaltet alle ausgehenden Punkt-zu-Punkt-Verbindungen und überwacht zudem eine eingehende Verbindung. Hier werden neue Verbindungen, welche aus einer entfernten Anfrage resultieren, initialisiert und wieder freigegeben, sobald eine Antwort erhalten, oder eine gewisse Zeit verstrichen ist. Die eingehende Verbindung wird mit dem SelectorManager registriert und aktualisiert so den `MessageDispatcher` , der wiederum die einzelnen `MessageReceiver` benachrichtigt.

PingManager Die `PingManager` Klasse verwaltet entfernte Ping-Anfragen und aktualisiert entsprechend die Tabellen (nächste Nachbarn, Weiterleitungstabelle) des lokalen Pastry-Knotens. Der Ping-Verwalter wird auch benutzt, um Ausfälle von Knoten zu detektieren.

Aufbauend auf dieser Basis-Transportschicht können die weiteren Protokolle (Past, SplitStream, Scribe) genutzt werden. Achtet man also auf eine saubere Schnittstelle, so lassen sich bestehende Anwendungen, ohne große Refaktorierungsarbeiten weiter benutzen. Im weiteren sollen die Unterschiede zwischen Pastry und dem EventChannelNetwork betrachtet werden. Hierbei auftretende Probleme werden untersucht und exemplarisch gelöst.

4.3 Unterschiede und Gemeinsamkeiten

Mit dem Wissen über den Aufbau und die Funktionsweise der FreePastry-Bibliothek ist es möglich, Unterschiede zwischen der Socket-basierten Variante und einer Realisierung auf der Basis des Nachrichtendienstes zu identifizieren. Während dieser Abschnitt Probleme und verschiedene Möglichkeiten derer Lösung darstellt, folgt im darauffolgenden eine Zusammenstellung der für eine prototypische Implementierung notwendigen Aufgaben.

In den folgenden Unterabschnitten werden Unterschiede bezüglich der Knotenanbingung an das zu Überlagernde Netzwerk und einer daraus resultierenden Netzwerkstruktur dargestellt. Eine kurze Darstellung der unterschiedlich organisierten Zuteilung der Rechenzeit leitet auf eine Betrachtung verschiedener Möglichkeiten bei der Schlüsselgenerierung.

4.3.1 Knotenverbindungen

Ein Vergleich, der Knotenanbindung an das zugrundeliegende Netzwerk wird durch eine Betrachtung der beiden Schaubilder 11 und 12 unterstützt. Zunächst soll die Variante der Socket-Anbindung betrachtet werden, bei eine vorbestimmte Menge von Punkt-zu-Punkt-Verbindungen dynamisch verwaltet werden.

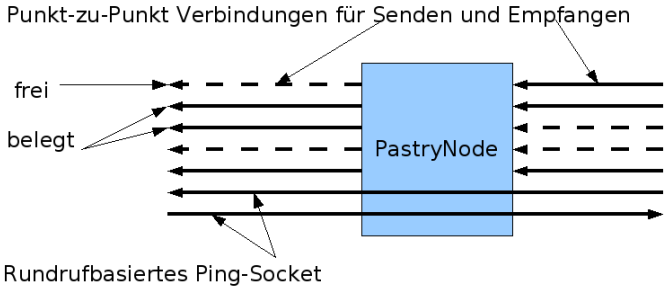

Abbildung 11: Knotenanbindung mit ECN und FreePastry-Sockets

Punkt-zu-Punkt-Basiert: Da, wie in Abbildung 11, ein Pastry-Knoten über mehrere separate Verbindungen für Ein- und Ausgabe verfügt, sowie die Tatsache, dass die Verbindungen durch eine dynamische Alloziierung und Freigabe verwaltet werden, erhöht sich, abgesehen vom Rechenaufwand, die Belastung des Netzwerks. Für ausgehende Verbindungen steht eine bestimmte Anzahl an Sockeln zur Verfügung. Die Klasse `SocketCollectionManager` hat nun die Aufgabe, diese zu verwalten und wartende Anfragen den freien Verbindungen zuzuweisen. Ebenso findet in dieser Klasse die Detektion ausgefallener Knoten statt, bei der, nach Ablauf einer bestimmten Antwortzeit, einen Knoten zunächst als „verdächtig" gekennzeichnet und zwecks einer Prüfung an den `PingManager` übergeben wird. Dieser `PingManager` prüft anschliessend, durch ansprechen des Knoten, ob nach einer gewissen Anzahl von Versuchen und einer maximalen Wartezeit eine Antwort vorliegt. Bleibt eine Antwort aus, wird eine Reparatur der Knotentabellen angestossen. Erst anschliessend kann die Anfrage im Verbindungsmanager ein zusätzliches mal weiter-, bzw. umgeleitet werden. An der Eingangsseite ordnet der `rice.selector.SelectorManager` eine erhaltene Nachricht mit Hilfe der `MessageDispatch`-Klasse den Interessenten (`MessageReceiver`) zu.

Nachrichtenkanalbasiert: In Abbildung 12 ist eine beispielhafte Darstellung für eine Verknüpfungmöglichkeit mit dem Nachrichtendienst.

Die logischen Kanäle im EventChannelNetwork dienen dem Senden und Empfangen von Nachrichten gleichzeitig. Genauso wie die Verwaltung der den Kanälen zugrundeliegenden physikalischen Verbindungen, wird auch die Zuteilung der zu sendenden und zu empfangenden Nachrichten von der EventChannelNetwork-API übernommen. Durch zusammenfassen mehrerer physikalischer Verbindungen in einem Kanal kann eine Redundanz- oder Gateway-Funktionalität realisiert werden, je nachdem, ob sich die physikalischen Verbindungen im gleichen Netzwerk, oder in unterschiedlichen Netzen befinden. Durch die logischen Kanäle wird ein Überlagerungsnetz zur Verfügung gestellt, dass unabhängig von der zugrundenliegenden Netzwerk-Hardware eine Adressierung auf Kanalbasis ermöglicht. Ein dynamisches Erzeugen neuer Kanäle ist mit einem gewissen Verwaltungsaufwand behaftet, wodurch eine solche Funktionalität nur in Systemen mit flexiblen Echtzeitanforderungen vorgesehen ist.

Im Fall der nachrichtenkanalbasierten Kommunikation ist mit weniger zusätzlichem Aufwand für die Verwaltung der Verbindungen zu rechnen, als wie es bei einer Punkt-zu-Punkt-basierten Pastry-Variante der Fall wäre.

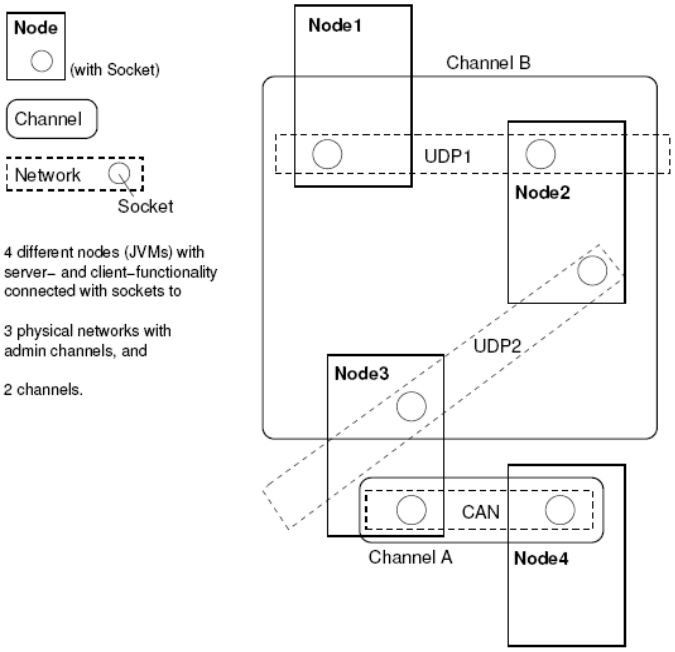

Abbildung 12: Eine abstrakte Darstellung des Echtzeit- Nachrichtendienstes, mit 2 logischen Kanälen und 2 physikalischen Netzwerken[31]

4.3.2 Resultierende Netzwerkstruktur

Aus den im vorigen Abschnitt beschriebenen Varianten der Anbindung der Pastry-Knoten an das zu überlagernde Netzwerk ergeben sich unterschiedliche Netzwerkstrukturen. Eine Betrachtung dieser Strukturen erfolgt, wie zuvor, für die Socket- und eine Event Channel Network-Anbindung.

Punkt-zu-Punkt-basiert: In Abbildung 13 ist eine Darstellung eines Pastry-Netzwerks aus einer subjektiven, das heißt, aus Sicht eines lokalen Knotens, zu sehen. Der zentral gelegene Knoten „Z" ist der lokale Knoten, Verbindungen stellen die über die Knotentabellen erreichbaren Knoten dar, wobei „B" die Knoten des Blattwerks, „W" die Knoten der Weiterleitungstabelle und „N" die Knoten, welche als nächste Nachbarn bezüglich ihrer Pingzeiten des Knotens darstellen.

Eine Anfrage wird nur die gestrichelte Linie dargestellt, welche über zwei Weiterleitungsknoten und einen, als Blattwerksknoten in W' eingetragenen Knoten B'. Mit Hilfe des sich auf dem Anfragepfad befindenden Knoten W konnte wird eine andere (subjektive) Ansicht des Netzwerks benutzt, welche aus komplexitätsgründen hier nicht dargestellt wird. W' und B' sind somit dem lokalen Knoten Z nur über den Weiterleitungsknoten W bekannt.

Nachrichtenkanalbasiert: Nun basiert der Nachrichtendienst bekannterweise auf dem Rundruf-Prinzip, wodurch sich innerhalb des vom Rundruf erreichbaren Netzwerks alle Knoten, ohne einen einzigen Weiterleitungsschritt erreichen lassen. Handelt es sich, wie in Abbildung 14, bei diesen Ringstrukturen um unterschiedliche physikalische Netze, so kann eine EventChannelNetwork eigene Weitergabe-, oder Gateway-Funktion verwendet werden. Diese Gateway-Knoten sind mit G gekennzeichnet dargestellt und verbinden die einzelnen Ringe zu einem Netz. Erst beim Übergang zwischen zwei logischen Nachtrichtenkanälen ist eine Weiterleitung nötig. Diese Form der Überbrückung ist grundsätzlich möglich, jedoch unter der Annahme, dass ein Nachrichtenkanal mit spezifischen Protokollen und Informationen verknüpft ist und eine Weitergabe einer Meldung mit Bearbeitung oder Transformation derselben einhergeht.

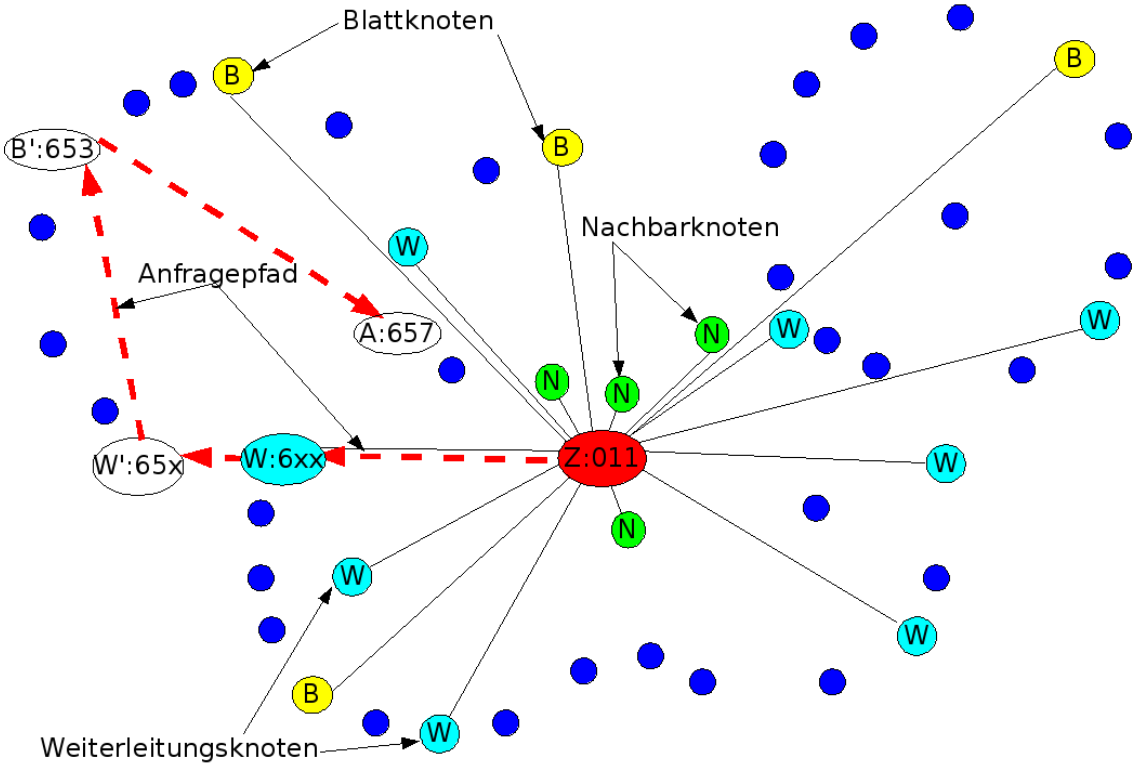

Abbildung 13: Die Pastry-Netzstruktur

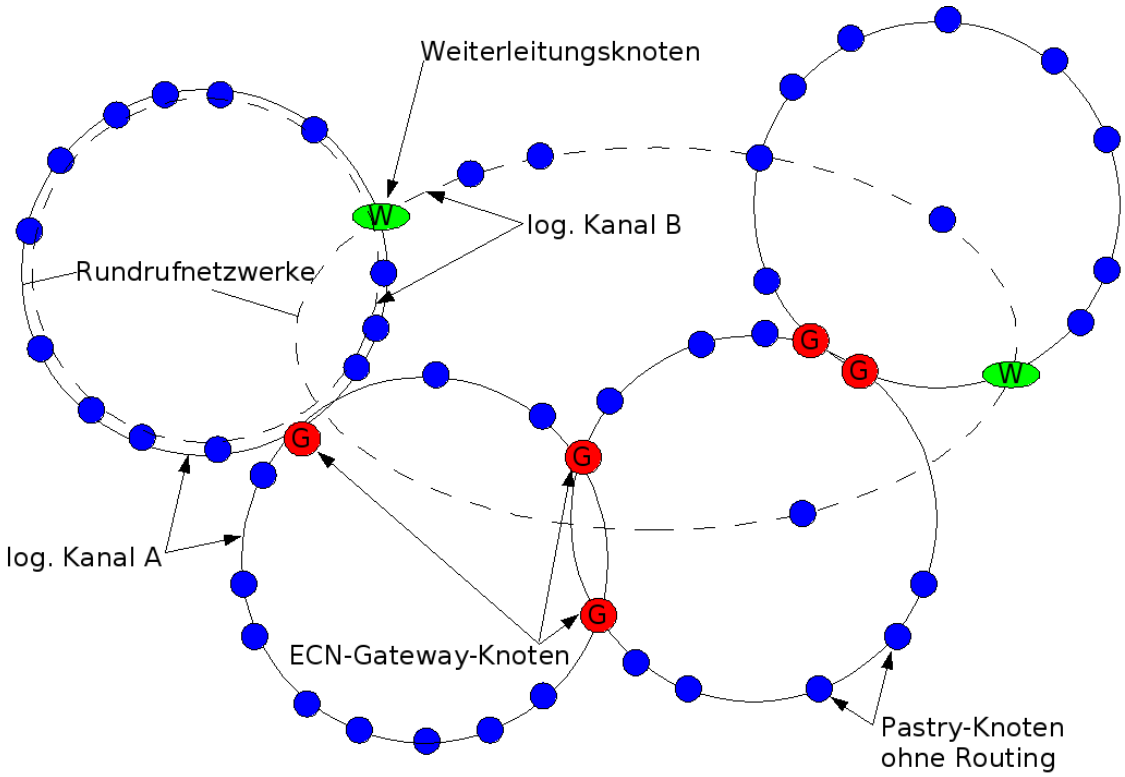

Abbildung 14: Mögliche Struktur eines Rundrufbasierten Pastry-Netzwerks

4.3.3 Pastry-Lokalitäts-Eigenschaften

Abbildung 15 zeigt eine Menge von Knoten, den Blattwerks-, Weiterleitungs- und Nachbarsknoten, in Bezug auf deren Schlüsselentfernung von lokalen Knoten und anschliessend auf ihre reale Entfernung.

Punkt-zu-Punkt-Basiert: Aus der vorigen Betrachtung ist zu Erkennen, dass das streng ID-basierte Weiterleiten nicht den idealsten Weg zwischen zwei Knoten wählt. In jedem Weiterleitungsschritt vergrößert sich die, über das Netzwerk zurückgelegte Strecke, bis sie schliesslich in der letzten Weiterleitung über den Blattwerksknoten ihr Maximum erreicht. Dies folgt Aufgrund der in Pastry geltenden Annahmen, dass die Nachbarknoten den größtmöglichen Schlüsselraum aufspannen und dass sich die Blattwerksknoten möglichst weit über das gesammte Pastry-Netzwerk verteilen. Es liegt also eine Antiproportionalität zwischen der Schlüsselentferung und der tatsächlichen Entfernung im zugrunde liegenden Netzwerk vor. Diese Netzwerktopologie besitzt nicht nur Nachteile, sondern verringert die Wahrscheinlichkeit einer, durch Knotenausfälle provozierte, Disjunktion des Netzwerks.

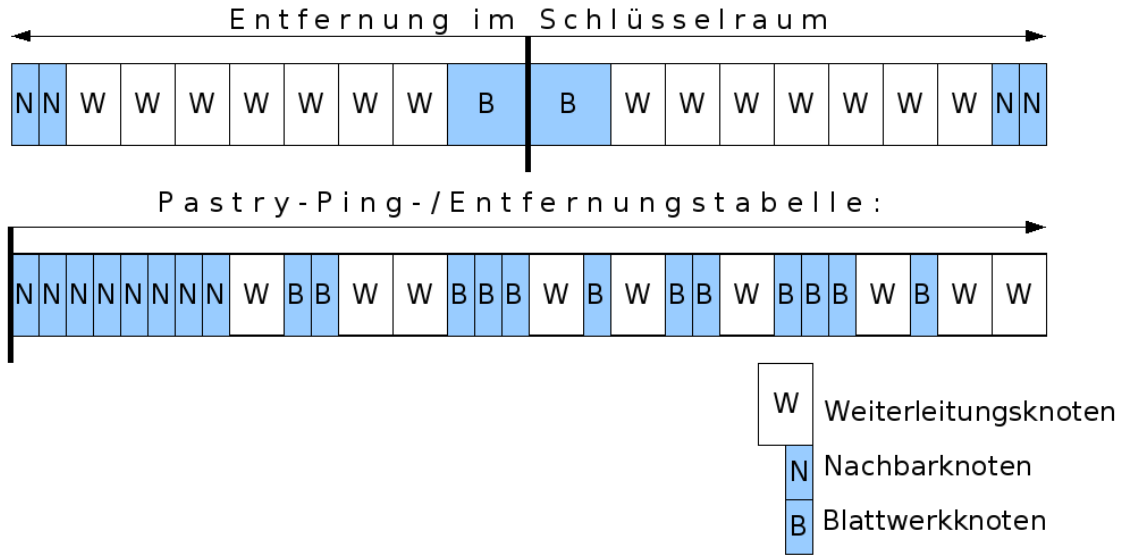

Abbildung 15: Schlüsselentfernung VS. Pingdistanz

Nachrichtenkanalbasiert: Es muss zwischen Pastry-Netzwerk innerhalb eines Kanals und eines Netzwerks, das sich über mehrere Kanäle erstreckt, unterschieden werden. Welche Auswirkung dies auf die Adressgenerierung oder dessen Aufbau hat wird im Folgenden untersucht.

4.3.4 Mögliche Adresshierarchien

Die Wahl einer geeigneten Adressierung innerhalb des Überlagerungsnetzwerks ist von großer Bedeutung. Nicht nur eine zu gering gewählte Schlüssellänge, sondern auch eine unbedachte Generierung des Schlüssel kann das Netzwerk untauglich machen. FreePastry versucht die in Abschnitt 4.3.3 dargestellten Probleme mit Hilfe eines pseudo-zufälligen Schlüssels zu umgehen, indem dessen Präfix aus dem Ergebnis der Hash-Funktion über IP-Adresse und Port erzeugt wird. Somit kann aufgrund eines gleichen Schlüsselpräfixes auf eine gemeinsame IP-Adresse geschlossen werden.

Die auf den Schlüsseladressen basierende Entfernungsfunktion bestimmt die Entscheidung, wohin eine Nachricht weiterzuleiten ist.

Da im Falle einer viele-zu-viele Kommunikation kein Weiterleiten notwendig ist und mit einem Aufwand von O(1), n Knoten erreicht werden, steht die Entfernung beim Wechsel in einen anderen

Nachrichtenkanal, einem mindestens doppelt so hohem Aufwand gegenüber. Dieser besteht aus den Teilaufwänden für das Übertragen über zwei Rundrufnetze hinweg mit einer zusätzlichen Weiterleitungslatenz.

Aus den dargestellten Gründen heraus werden im folgenden unterschiedliche Konstellationen aus Pastry-Netzen und logischen Kanälen betrachtet. Zugehörige Möglichkeiten der Adressierung und Schlüsselgenerierung schliessen diesen Abschnitt ab.

Ein Pastry-Netzwerk innerhalb eines Nachrichtenkanals Da innerhalb eines logischen Nachrichtenkanals das Rundruf-Prinzip verwendet wird, ist kein Weiterleitungsverfahren notwendig. In diesem Fall stellt die, in Pastry definierte gleichmäßige Verteilung der Schlüssel, kein Problem in Betracht auf die Lokalitätseigenschaften dar.

Zwei Überlagerungsnetze in zwei unterschiedlichen logischen Kanälen können weiterhin, jedoch nur auf Applikationsebene, verbunden werden und stellen somit auch zwei unterschiedliche Pastry-Netze dar. Auf diese Wiese wird eine maximale Trennung erreicht. Eine Interaktion der beiden Netzwerke ist damit auf Anwendungsebene zu definieren, was somit eine vereifachte Betrachtung der Teilsysteme möglich macht.

Ein Pastry-Netzwerk über mehrere EventChannels In diesem Fall liegt die Verwendung spezieller „Superknoten" nahe, die die Aufgabe haben, zwei unterschiedliche logische Kanäle zu einem Pastry-Netzwerk zusammenzufügen. Dazu legen sie kanalüberschreitende Weiterleitungstabellen an, welche jeweils dem gegenüberliegenden Subnetz für Anfragen zur Verfügung stehen. Da die in Pastry definierte gleichmäßige Schlüsselverteilung eine Entscheidung des Weiterleitungsproblems unmöglich macht, muss eine Möglichkeit gefunden werden, die Zugehörigkeit eines Knotens zu einem Kanal zu erkennen. Eine pauschale Weiterleitung aller Nachrichten kommt nicht in Frage, da das Netzwerk auf diese Weise schnell durch Kettenreaktionen überlastet wäre. Ein anderer Ansatz versucht, den Schlüssel, wie bei der IP/Port-basierten Schlüsselerzeugung, auf Grundlage des Namens des Nachrichtenkanals zu erzeugen. Dadurch unterscheiden sich die Stellen des Schlüssel, die den EventChannel wiederspiegeln nur dann, wenn es sich um unterschiedliche Kanäle handelt. Die übrigen Schlüsselstellen werden weiterhin zufällig generiert, um die Pastry-Spezifikationen zu erfüllen.

Möglichkeiten den Nachrichtenkanal im Schlüssel zu kodieren werden unterstützt durch Abbildung 16 dargestellt. Diese Abbildung lässt in der ersten Zeile die Schlüsselpartitionierung in 2^b lange Teilstücke, den Schlüsselstellen, erkennen.

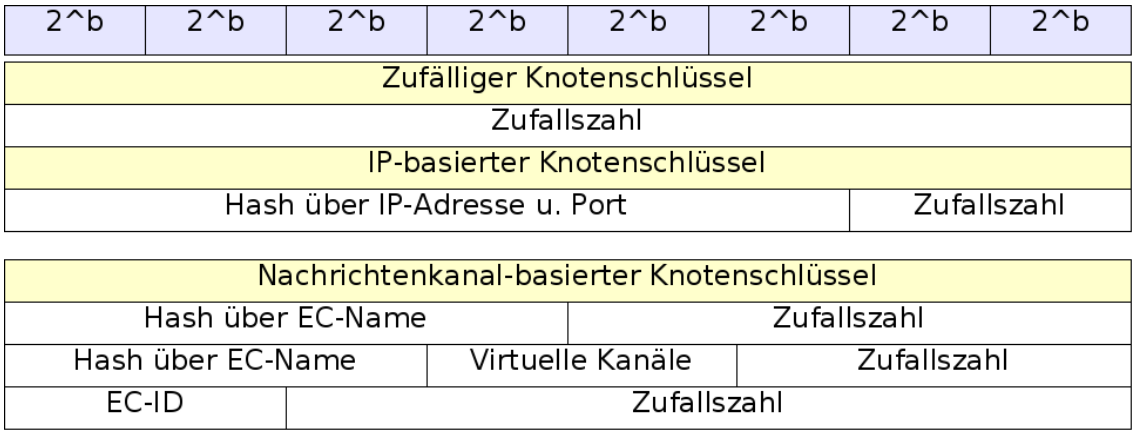

Abbildung 16: Verschieden generierte Pastry-Schlüssel

- Vollständig zufällige Schlüsselerzeugung
  Diese Schlüsselgenerierung wird beispielsweise im `rice.pastry.direct`-Paket der FreePastry-API benutzt, um für lokale Testzwecke ausreichend zufällige Schlüssel erzeugen zu können.

Ebenso steht einer Verwendung dieses Verfahrens für den Fall, dass sich das Pastry-Netzwerk lediglich über einen Nachrichtenkanal erstreckt, nichts im Wege.

- IP-basierte Schlüsselerzeugung
  Diese Variante stellt die in FreePastry normalerweise verwendete Schlüsselgenerierung dar.
- Nachrichtenkanal-Name im Knotenschlüssel enthalten
  Dieser Fall entspricht der für FreePastry vorgestellten IP-basierten Schlüsselgenerierung. Das Ergebnis der Hash-Funktion über den Namen des Nachrichtenkanals wird durch Auffüllen der restlichen Stellen mit einer Zufallszahl vervollständigt.
- Nachrichtenkanal-ID im Knotenschlüssel enthalten
  Die Kanäle des EventChannelNetwork besitzen nicht nur einen Namen, sondern auch eine 32-bit lange, eindeutige Identifikationsnummer. Anstelle des Hash-Werts wird diese als Präfix des Schlüssels gewählt. Dadurch entfällt zum einen die Notwendigkeit einer konsistenten Hash-Funktion, zum anderen wird der Platzbedarf an Stellen, die zum Kodieren des Kanals benötigt werden, auf ein Minimum reduziert.

Die folgende Adressierungsmöglichkeit ist in der Abbildung nicht dargestellt und wird auch im weiteren Verlauf dieser Ausarbeitung nicht weiter benötigt. Dabei handelt es sich um den Fall, den Namen des Nachrichtenkanals dem Knotenschlüssel nach dem Schema [Name des Nachrichtenkanals]://Knotenschlüssel zu spezifizieren. Auf diese Art kann eine weitere Dimension für das Gleiche-zu-Gleiche-Netzwerk erzeugt werden.

4.4 Weitere Probleme: Beispiel des Bekanntmachungsprotkolls

Exemplarisch wird im Folgenden der Pastry-Beitritt Prozess für den Fall eines Pastry-Netzwerks über einen einzelnen Nachrichtenkanal hinweg betrachtet. In diesem Zusammenhang werden ebenso die Auswirkungen auf die Tabellen der Blatt- und Nachbarknoten beschrieben.

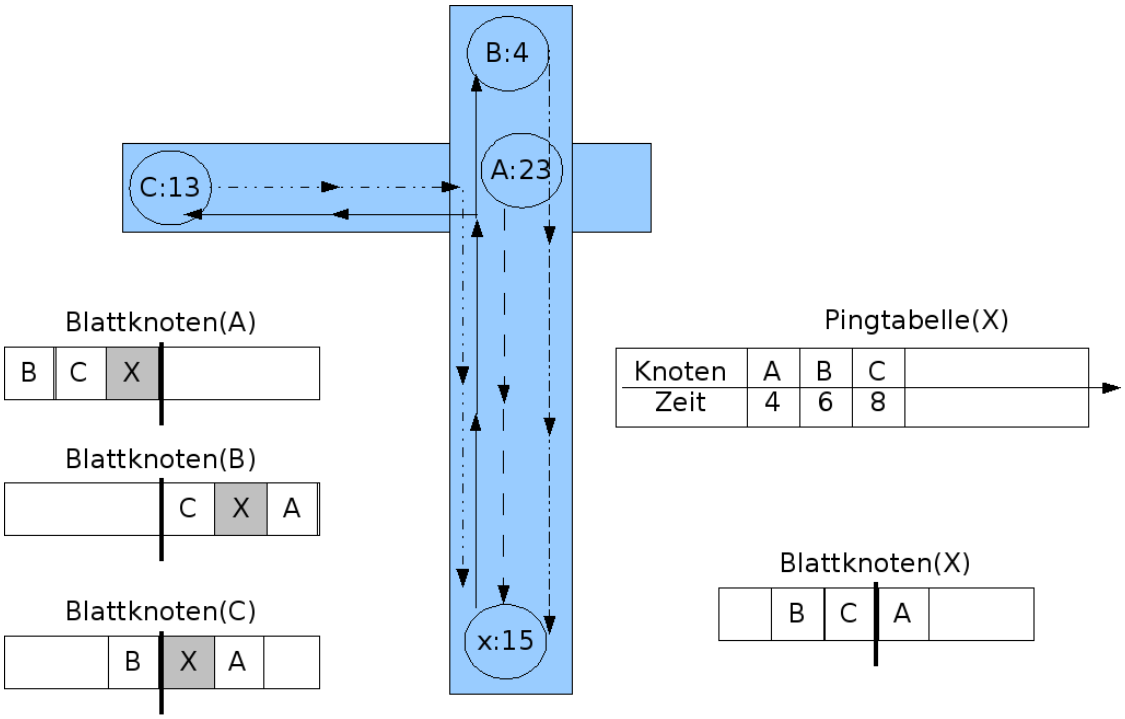

Abbildung 17: Beitritt eines neuen Pastry-Knotens X im ECN

4.4.1 Das Beispiel

Abbildung 17 veranschaulicht eine abgeschlossene Beitrittssituation des Knotens X zu dem, aus den Knoten A, B und C bestehendem Pastry-Netzwerk. Für den neu hinzukommenden Knoten ist neben der Tabelle der Blattknoten auch die Tabelle der Nachbarknoten dargestellt. Die eingezeichneten Pfeile symbolisieren den Verlauf einer Anfrage, wobei ein gestrichelter Pfeil den Verlauf einer Antwortnachricht kennzeichnet.

Folgend werden die einzelnen Schritte bis zu der in Abbildung 17 dargestellten Situation beschrieben.

1. Der Knoten X stellt eine Verbindung mit dem Nachrichtenkanal her.

2. Eine Beitrittsanfrage wird durch Versenden einer `NodeIDRequestMessage` im Nachrichtenkanal verteilt. Darauf wird eine Reihe von Nachrichten empfangen, welche in folgender Tabelle aufgelistet sind:

Zeitpunkt	Gesendet	Empfangen(von Knoten)
0	Schlüsselanfrage	-
4	-	Schlüsselbestätigung(A)
5	-	Blattknoten(A)
6	-	Nachbarknoten(A), Schlüsselbestätigung(B)
7	-	Blattknoten(B)
8	-	Nachbarknoten(B), Schlüsselbestätigung(C)
9	-	Blattknoten(C)
10	-	Nachbarknoten(C)

   Aus der Reihenfolge, mit denen die Antwortnachrichten eintreffen, lässt sich die Tabelle der Nachbarknoten erzeugen. Die in den Nachrichten enthaltenen Blattknoten werden in die entsprechende Tabelle des Knotens X eingefügt, sofern sich diese anhand ihres Schlüssels dazu eignen.

3. Eine Vorstellung bei den schon aktiven Knoten des Pastry-Netzwerks macht den Knoten X zu einem vollwertigen Mietglied des Systems. Die geschieht durch senden der lokalen Blattknotentabelle. Von diesem Zeitpunkt an kann der Knoten höherwertige Aufgaben erledigen.

4.4.2 Probleme

An diesem kleinen Beispiel sind noch nicht angesprochene Probleme zu erkennen:

- Eine Bestätigung des von Knoten X gewählten Schlüssels ist nicht von jedem Knoten des Netzwerk nötig:

  Handelt es sich, wie bei einer Anfrage, um eine einzige benötigte Antwort, dann ist zu prüfen, ob die Aufgabe nicht schon durch einen anderen, in Frage kommenden Knoten erledigt wurde. Mit dieser Vorgehensweise könnten in obigem Beispiel zwei Nachrichten eingespart werden.

- Blattknoten werden nur gesendet wenn der, zur Anfrage gehörende Schlüssel im Bereich der eigenen Blattknotentabelle liegt:

  In diesem Fall werden mehrere Antworten benötigt. Wird das Beispiel in größerem Umfang durchgespielt, wird ersichtlich, dass nicht alle der in Frage kommenden Knoten zum Zug kommen können. Eine Überlastung des Netzwerk wäre die Folge. Wenn also nur die, für die jeweiligen Blattknoten zuständigen Knoten ihre Blattknotentabelle versenden, beträgt die maximale Anzahl der zu erwartenden Nachrichten $|L|$ (die Größe der Blattknotentabelle). Damit ist der Aufwand eines Knotenbeitritts nicht, wie in Pastry, abhängig von der Größe des Gleiche-zu-Gleiche Netzwerks (vgl. $O(log(n))$). Dadurch, dass nicht jede Blattknotentabelle einzeln angefragt werden muss, reduziert sich die Anzahl an notwendigen Nachrichten, (gegenüber einer Punkt-zu-Punkt basierten Kommunikation,) um Faktor 2.

Daraus wird ersichtlich, dass eine verteilte Zuständigkeitskette unter den Knoten entwickelt werden muss. Bei dieser darf eine Beachtung möglicher Knotenausfälle nicht vergessen werden.

- Zuständigkeitsentscheidung: Der Knoten prüft, ob er die Anfrage überhaupt beantworten kann. Ist dies der Fall, dann wird geprüft, ob nicht zwischenzeitlich ein anderer Knoten die Bearbeitung übernommen hat. Länger dauernde Anfragebehandlungen sind vom bearbeitenden Knoten durch Senden einer Auftragsbestätigung zu quittieren, damit deren Abarbeitung nicht mehrfach ausgeführt wird.
- Ausfallbehandlung: Im Hinblick auf Redundanz sind Zuständigkeitsregeln nicht hinreichend. Für den Fall, dass ein Knoten während der Abarbeitung einer Anfrage ausfällt, muss diese Arbeit von einem anderen Knoten übernommen werden. Je nach Anwendungsfall ist zu entscheiden, ob ein Knoten eine Anfrage bearbeitet und anschliessend über ein Versenden der Antwort entscheidet, oder ob er zunächst bis zum Ablauf einer gewissen, vordefinierten Zeit warten soll, bevor er mit der Bearbeitung beginnt und die anschliessend die Antwort verschickt. Andere Anwendungsfälle benötigen eine Mehrfachberechnung, um die Korrektheit einer Antwort bestätigen zu können.

4.5 Nötige Erweiterungen zur Implementierung einer Pastry-API

Die Erkenntnisse der letzten Abschnitte werden im Folgenden zusammengetragen und aufgelistet. Zunächst werden die für eine einkanalige Pastry-Implementierung nötigen Erweiterungen dargestellt, im zweiten Abschnitt werden die Komponenten beschrieben, die für eine Weiterleitung über mehrere Nachrichtenkanäle hinweg dienen.

4.5.1 Einkanaliges Pastry-Netzwerk

Umstrukturierung der Protokolle

- Weiterleitungsprotokoll - StandardRouter:
  Da in einem einkanaligen Pastry-Netzwerk kein Weiterleitungsverfahren nötig ist, muss dieses Protokoll um diese Funktion reduziert werden.
- Bekanntmachungsprotokoll - StandardJoinProtocol:
  Hier sind die Änderungen, die in Abschnitt 4.4.1 und 4.4.2 vorgestellt wurden einzuarbeiten um den Aufwand an benötigten Nachrichten zu reduzieren.
- Blattknotenprotokoll - LeafsetProtocol:
  Dieses Protokoll muss zunächst die eigene Zusändigkeit überprüfen, indem getestet wird, ob der anfragende Knoten im Bereich des Blattwerks liegt.
- RouteSetProtocol:
  Dieses Protokoll wird aufgrund der nicht mehr benötigten Weiterleitungstabelle nicht mit dem Pastry-Knoten registriert.

Distanzfunktion Die Entfernungsfunktion muss ebenfalls an die Netzwerkstruktur angepasst werden. Innerhalb eines Kanals lassen sich lediglich Pingzeiten berechnen, nach denen sich die nächsten Nachbarn bestimmen lassen. Eine weitere Entscheidungsgrundlage für das Bearbeiten einer Anfrage stellen die (physikalischen) Entfernungen der Knoten untereinander dar. Damit möglichst kurze Antwortzeiten möglich sind, ist der Knoten für eine Bearbeitung der Anfrage zuständig, der sich am nächsten zum Anfrageknoten befindet. Eine Entfernungstabelle kann durch die Differenz zwischen Anfrage- und Antwortzeitpunkt ermittelt werden. Damit müssen keine zusätzlichen, das Netzwerk belastenden Nachrichten versendet, bzw. empfanden werden. Ebenso können durch Mitlauschen des Nachrichtenverkehrs auf dem Kanal, die Entfernungen zu weiteren Knoten zu bestimmt werden.

Zwischenspeichern von Nachrichten (Caching) Ein Nachrichtenpuffer nimmt eine bestimmte Anzahl letzter Nachrichten in eine Liste auf. Bevor ein Knoten nun eine Anfrage über den Nachrichtenkanal versendet, wird geprüft, ob sich nicht eine antwortende Nachricht im Zwischenspeicher befindet. Dabei muss beachtet werden, dass die Nachrichten immer nur eine gewisse Zeit gültig sind und danach zu Inkonsistenzen führen können. Dieser Zwischenspeicher besitzt eine bestimmte, festgelegte Größe. Ist der Zwischenspeicher voll, so muss bei einer neu angekommenden Nachricht die älteste enthaltene entfernt werden.

4.5.2 Pastry über mehrere Nachrichtenkanäle

Für die angestrebte Implementierung ist darüber hinaus die Möglichkeit vorgesehen, ein Pastry-Netzwerk über mehrere Nachrichtenkanäle hinweg zu betreiben. Diese Tatsache macht die folgenden Erweiterungen nötig.

Hierarchisierung der Knoten Im allgemeinen ist eine Hierarchisierung der Knoten in einem Gleiche-zu-Gleiche-Netzwerk nicht wünschenswert. Jedoch bietet sich eine solche Unterscheidung an, um auf unterschiedliche Rechenleistung und Speicherplatz flexibel reagieren zu können. Einfache Knoten stellen die Mehrzahl der Knoten des Pastry-Netzes dar. Eine Kommunikation innerhalb des EventChannels geschieht ohne Weiterleitung. Wird doch ein Schlüssel angefordert, der sich offensichtlich nicht im lokalen Kanal befindet, so wird dies nur durch einen Super-Knoten erkannt, da dieser auch über Knoten des Zielkanals informiert ist. Wenige „Super-Knoten" übernehmen somit einen größeren Schlüsselbereich und sind somit besser über den Zustand des Überlagerungsnetzwerks informiert. Gerade im Bereich der eingebetteten Systeme, stellt eine Reduzierung des Resourcenbedarfs und der Funktionalitäten auf ein nötiges Minimum, einen wichtigen Entwurfsaspekt dar. Hinsichtlich dessen können auch „Micro-Peers" überdacht werden, welche lediglich in eine Richtung mit dem Überlagerungsnetz in Kontakt treten. Diese könnten auch blind gegenüber dem Netzwerk sein, das heißt, dass sie keine Informationen über andere Knoten des Netzwerks besitzen.

Routing zwischen den Nachrichtenkanälen Die Superknoten sind in der Lage zwischen zwei EventChannels eine Nachrichtenweiterleitung zu initiieren. Dies setzt voraus, dass ein Superknoten über eine Weiterleitungstabelle samt zugehörigem Protokoll verfügt. Dieses Protokoll stellt eine Erweiterung des im vorigen Abschnitt beschriebenen (Weiterleitungs-)protokolls dar und wird folgend dargestellt.

```
receiveMessage(Message msg)
{
  if (msg.destinationID == localHandle)
  {  // ist schon am Ziel angekommen.
    pastryNode.receiveMessage(msg);
  }
  else if (localPastryNode.isRouted())
  {
    long l = sharedPrefix(localhandle,msg.destinationID);
    if (l < eventchannel_bitlength)
    {  // sende in anderem Kanal
       // bestimmen, über welchen Kanal/Knoten
       nextHop = getNextHop(msg);
       // senden
      pastryNode.send(msg);
    }
  }
  else { // ingoriere Nachricht }
}
```

Schlüsselgenerierung auf Namen oder ID des Nachrichtenkanals Während im Fall eines Pastry-Netzwerks innerhalb eines einzigen Nachrichtenkanals ein zufällig erzeugter Schlüssel ausreichend ist, muss in einem, mehrere Nachrichtenkanäle überspannenden Gleiche-zu-Gleiche-System eine Generierung, wie in Abschnitt 4.3.4 beschrieben, benutzt werden. Dabei wird Wahlweise die Identifikationsnummer des Nachrichtenkanals, oder das Ergebnis der Hash-Funktion, die auf dessen Namen ausgeführt wird, als Schlüsselpräfix verwendet werden.

5 Umsetzung

Dieser Teil beschäftigt sich mit der konkreten Realisierung der im vorigen Abschnitt besprochenen Strukturen und Algorithmen. Vom Groben ausgehend werden die verwendeten Software-Strukturen dargestellt, an welche sich eine genauere Erläuterung der Fasadenschicht und der Schicht, die die Schnittstelle zum EventChannelNetwork darstellt, anschließt.

5.1 Entwurf

Die Zweischichtige Konzeption, welche der Pastry-Socket-Anbindung zugrunde liegt, soll durch eine dreischichtige ersetzt werden. Diese erfolgt aus dem Grund nahe, dass das EventChannelNetwork selbst in unterschiedlichen Varianten vorliegt, die sich nicht in ihrer groben Struktur, wohl aber in Aspekten der Dynamik und der Behandlung von Zeitüberschreitungen unterscheidet. Diese Entkoppelung besitzt zudem den Vorteil, dass ein Austausch des EventChannelNetworks bei Einführung einer neuer Version mit geringem Aufwand vollzogen werden kann.

Die einzelnen Teilprotokolle sollen wie zuvor auch, unabhängig von der Transportschicht registriert werden können. Da dies zur Ausführungszeit stattfindet, wäre auch ein Aktualisierungsmechanismus möglich. Ein Protokoll besteht aus mehreren speziellen Nachrichten, welche durch eine Nachrichtentyp-ID eine Zuordnung zwischen Anfrage und Antwort zulässt.

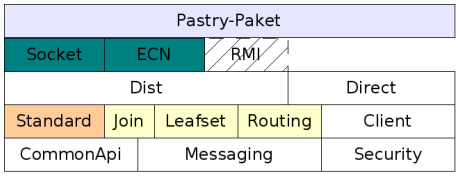

Abbildung 18: Die FreePastry-Module mit ECN-Anbindung

Als Anknüpfungspunkt für die EventChannelNetwork-Basis wird das `rice.pastry.dist` Paket gewählt. In Abbildung 18 ist die Einordnung des neuen Pakets (ECN) in die bestehende API zu erkennen.

5.1.1 Das Pastry-EventChannelNetwork Modul

In den folgenden zwei Abbildungen sind die Interaktionen des PastrySocket-Pakets bzw. der ECN-Variante mit den darüber, sowie der darunterliegenden Netzwerkanbindungsschicht dargestellt.

Dabei ist zu erkennen, wie die Nachrichtendienstschnittstelle vollständig im zentralen EventChannel-Manager (`ECManager`) untergebracht wurde. Dieser kümmert sich um das Anlegen und Verwalten von Nachrichtenkanälen. Zudem ist hier die Anbindung an einen konkreten Nachrichtendienst zu finden. Im Nachrichtenkanalverwalter findet die Transformation der Pastry-Nachrichten (`Message`) in die vom EventChannelNetwork verwendeten Nachrichten (`RemoteEvent`) statt. Beim Anlegen eines neuen Pastry-Netzwerks muss zunächst ein Nachrichtenkanal erzeugt werden und mit einem `PastryAdminChannelFilter` verknüpft werden. Dieser handhabt die rudimentären Dienstleister-Funktionalitäten eines Pastryknotens. Damit sind zunächst die Behandlung eines Knotenbeitritts (`NodeIDRequestMessage`), und Blattwerksanfragen enthalten. An den `PastryAdminChannelFilter` lassen sich nach dem Beobachter-Muster mit Hilfe der beiden Klassen `MessageReceiver` und

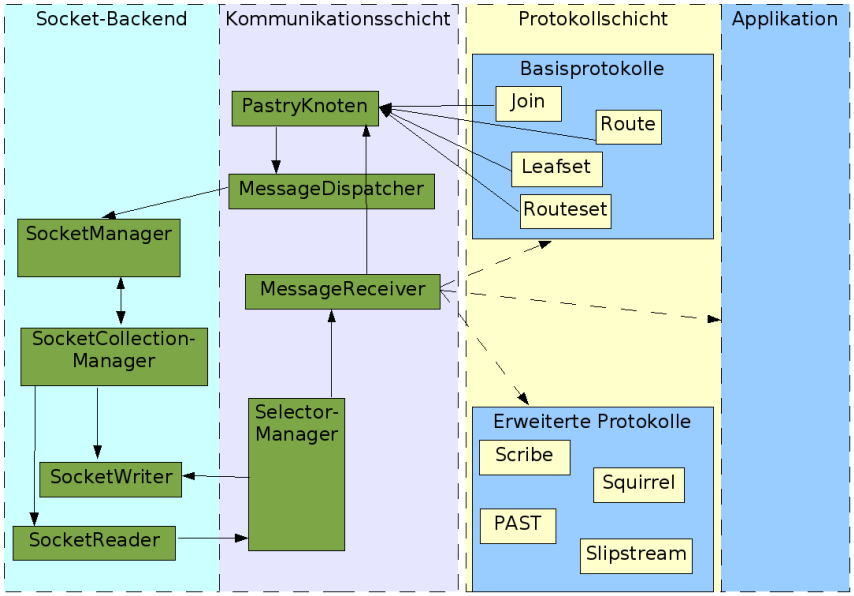

Abbildung 19: Interaktionen der Socket-API mit FreePastry

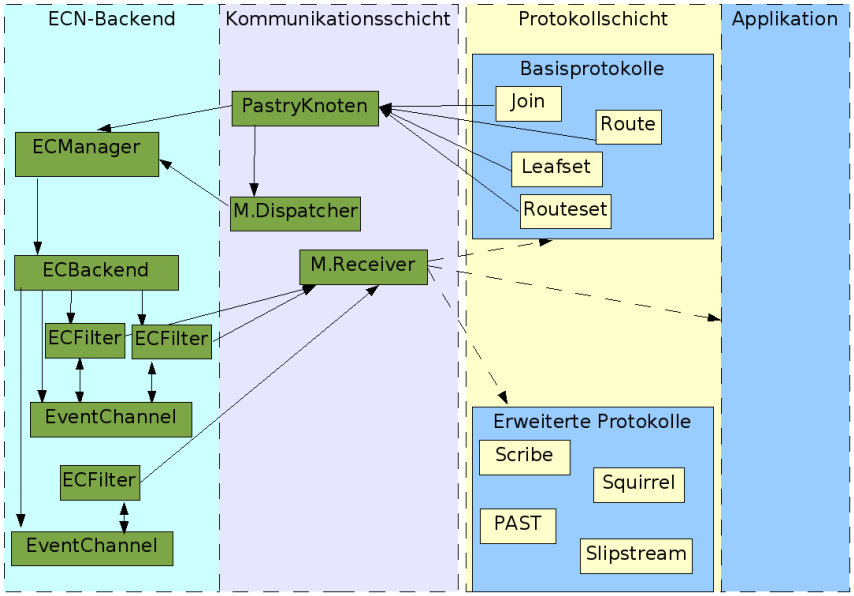

Abbildung 20: Interaktionen der ECN-API mit FreePastry

`MessageDispatch` einhängen. Damit können weitere Protokolle oder anwendungsspezifische Nachrichten, welche der Administrationbehandler nicht kennt, an die entsprechenden (lokalen) Empfänger weitergeleitet werden. Funktionalitäten, wie die Zwischenspeicherung von Nachrichten, oder das versenden von blockierenden und nicht-blockierenden Nachrichten findet über die Schnittstelle des `ECManager` statt.

5.1.2 Die Fassade

Die Fassade bildet das Frontend zur schon bestehenden FreePastry-API. Das EventChannelNetwork-Paket (`de.fzi.ecn.pastry`) soll die gleichen Funktionalitäten bereitstellen wie das in FreePastry standardmäßig benutzte Sockel-Paket (`rice.pastry.socket`). Das EventChannelNetwork-Modul enthält zwei Unterpakete, von denen eines, analog zur herkömmlichen Variante, Nachrichten-Klassen (`de.fzi.ecn.pastry.messages`) enthält. Ein weiteres Unterpaket (`de.fzi.ecn.pastry.backend`) stellt die Schnittstelle zum EventChannelNetwork dar. Im Folgenden wird ein Überblick über die Pakete und die enthaltenen Klassen, ausgehend von der obersten Schicht bis hinunter an die Anbindung des EventChannelNetwork gegeben.

ECNPastryNode Diese Klasse dient als Repräsentation des lokalen Knotens. Ein Pastryknoten erhält das zusätzliche Attribut `isRouterNode`, welche bestimmt, ob dieser Knoten über Weiterleitungstabellen, samt zugehörigem Protokoll besitzt. Diese Eigenschaft muss vor der Erzeugung des Knotens gegeben sein. Darüber hinaus ist daraus zu achten, dass ein Weiterleitungsknoten mit mindestens zwei Nachrichtenkanälen verknüpft wird.

ECNPastryNodeFactory Die Fabrikklasse erzeugt paketkonforme Knoten und ebenso eine nachrichtendienstspezifische Instanz der Klasse `ECBackend`, die innerhalb der `ECManager`-Klasse mit dem lokalen Knoten verknüpft wird.

Sende- und Empfangsfunktionalitäten, die die `rice.pastry.dist.DistPastryNodeFactory` normalerweise Bereitstellt wurden in `ECNPastryNodeFactory`nach Möglichkeit an die `ECManager`-Klasse weitergeleitet.

ECNPastryNodeHandle Dient der Abstraktion eines entfernten Knotens.

ECNPastryNodeHandlePool Ein einfacher Behälter für entfernte Knoten (`ECNPastryNodeHandle`)

ECNSourceRoute Stellt einen Pfad über mehrere aufeinanderfolgende EventChannels dar.

ECManager Diese Klasse kümmert sich um die Verwaltung des Nachrichtendienstes. Hierzu gehört die Erzeugung einer `EventChannelFactory` innerhalb einer `ECBackend`Klasse, welche folgend noch betrachtet werden soll. Damit stellt der `ECManager`eine Verbindung zur abstrakte Nachrichtendienstschnittstelle her. In dieser Klasse werden die zu sendenden Pastry-Nachrichten in die vom EventChannelNetwork verwendeten `RemoteEvent`s gepackt. Beim empfangenen von EventChannel-Nachrichten wird die enthaltene `Message` herausgelöst und an die darüberliegende Paketschicht weitergegeben. Eine weitere wichtige Funktionalität umfaßt das Aktualisieren der in der `ECNPastryNodeFactory` enthaltenen Ping-Tabelle, oder das Zwischenspeichern von Nachrichten. Die Assoziation einer zu sendenden Nachricht mit einem Nachrichtenkanal findet ebenfalls in dieser Klasse statt. Somit handelt es sich um den zentralen Zugriffspunkt auf die EventChannelNetwork-Schnittstelle.

5.1.3 Die abstrakte EventChannelNetwork Schnittstelle

Damit ein leichtes Austauschen des asynchronen Nachrichtendienstes möglich ist, wird mit einer abstrakten Zwischenklasse der `ECManager` von einer konkreten EventChannelNetwork implementierung entkoppelt. Im Folgenden wird diese Schnittstelle genauer betrachtet.

ECBackend Diese Klasse kümmert sich um die Organisation und Ansteurung des asynchronen Nachrichtendienstes. Um die Interaktion mit der `ECManager`-Klasse zu ermöglichen, sind eine Reihe weiterer Strukturen nötig, die von `ECBackend` ausgehend beschrieben werden.

Abbildung 21: Die ECBackend-Klasse

In Abbildung 21 ist das UML-Diagramm dieser Klasse zu sehen. Zu den zentralen Aufgaben gehört:

- Initialisieren der EventChannelNetwork-Fabrik Damit geht auch eine Anlegen, bzw. Abonnieren eines Basiskanals („PastryAdminChannel") einher.
- Knotenbeitritt zum Pastry-Netzwerk Mit Hilfe einer Bootstrap-Methode (`bootstrap`) kann der Pastry-Betrittsprozess initiiert werden. Durch eine Anfrage mit `isOpen` kann schnell festgestellt werden, ob ein entsprechender Initialisierungnachrichtenkanal überhaupt vorhanden ist.
- Verwalten der Nachrichtenkanäle Dazu gehört eine Suchen vorhandener, das Erzeugen und Schließen von Nachrichtenkanälen. Entsprechende Behandlungskomponenten werden mit erzeugt und registriert.
- Verwaltung der Kommunikation Dies betrifft das Senden und Empfangen von EventChannelNetwork-Nachrichten. Für das Senden stehen zwei Funktionen zur Verfügung. Die erste `sendEvent` bedient die Dienstgebenden Funktionen, die zweite Sendemethode `sendRequest` stellt, unter zu Hilfenahme eines EventChannel-Filters, eine Anfrage, mit darauf folgender Antwortnachricht dar. Die `EventChannelFilter`-Klasse wird im Folgenden noch genauer beschrieben. In diesem Fall wird sie benutzt um die, zu einem späteren Zeitpunkt eintreffende Nachricht herauszufiltern und mit Hilfe der `EventChannelFilter`-Methode `newEvent` über die eingetroffene Antwort zu benachrichtigen.

EventChannelFilter Stellt eine abstrakte Klasse zur Verfügung, welche aus einer Liste von `RemoteEvents` eine entsprechende Nachricht herausfiltert.

EventChannelResponseFilter Stellt eine abstrakte Klasse zur Verfügung, welche aus einer Liste von RemoteEvents eine entsprechende Nachricht herausfiltert. Sollte keine solche Nachricht enthalten sein, so wird nach einer definierten Zeit `waitTime` erneut die Liste durchsucht, sooft, wie eine bestimmte Anzahl von Versuchen `tries`definiert wurde. Dieser Antwort-Filter überprüft zunächst den Nachrichtentyp und lässt so eine schnelle Entscheidung über die relevanz der geprüften Nachrichten zu. Da diese Variante eine blockierende Anfrage darstellt implementiert diese Klasse zusätzlich die Schnittstelle `Continuation`. Damit wird eine Integration in die von FreePastry realisierte Prozessorabstraktion erreicht. Diese Klasse liegt wiederum in zwei Varianten

Folgend, in Abbildung 22, ist die Vererbungsstruktur der EventChannelFilter dargestellt.

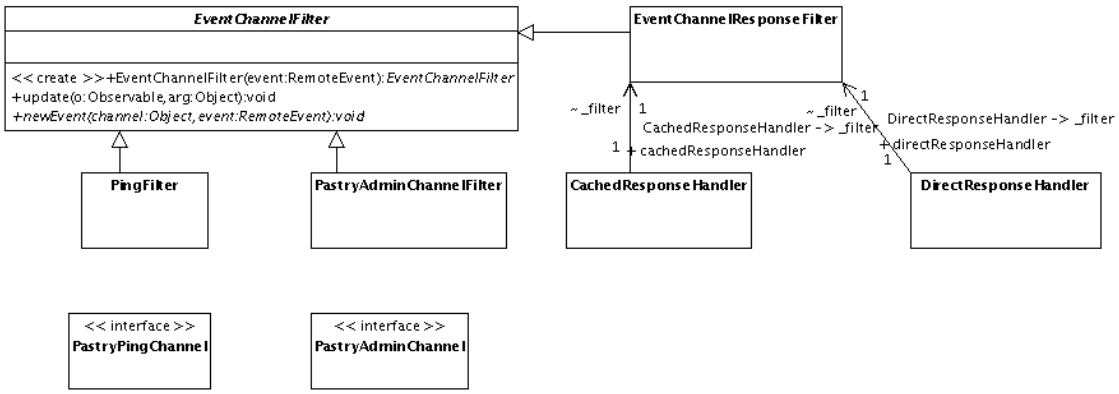

Abbildung 22: Unterschiedliche EventChannelFilter

PastryAdminChannelFilter In Abbildung 22 ist die Klasse `PastryAdminChannelHandler` zu erkennen. Dieser filtert zunächst Administrationnachrichten des EventChannelNetwork, oder eigens versendete Anfragen. Anschließend erfolgt eine Prüfung auf Pastry-Transportschicht-Anfragen, die an direkt von diesem Filter übernommen werden können. Sind die Nachrichtentypen unbekannt sein, so werden diese an den `MessageDispatcher` des Pastryknotens weitergereicht. Dieser überstellt die erhaltenen Nachrichten an die entsprechenden, bei der Erzeugung des Knotens registrierten, protokollspezifischen `MessageReceiver`-Objekte. (LeafSet, RouteSet, Anwendungsspezifische Nachrichten)

Folgender Quellcode (5.1.3) soll exemplarisch die Behandlung eines Knotenbeitritts illustrieren.

```
public void handleNodeIDRequest(RemoteEvent event)
{
  NodeId senderId = event.getMessage().getSenderId();
  if (_handler.hasCachedResponse(event)
  { // wurde schon durch anderen Knoten beantwortet
  }
  else {
    // versende Antwort.
    _handler.sendPastryAdminMessage(
        new NodeIdResponseMessage(
          _pastryNode.getNodeId(),
          _pastryNode.getECManager().getLocalAddress().getEpoch()
        )
    );
```

```
    }
    // Anstatt den Anfragenden Knoten auf weitere Boot-Infos warte
    // zu lassen kann man diese auch gleich mitsenden
    if (_pastryNode.getLeafSet().inRange(senderId))
    { // Blattwerk ist für den Knoten interessant
      _handler.sendPastryAdminMessage(
        new LeafSetResponseMessage(_pastryNode.getLeafSet())
      );
    }

    // genause wird mit dem RouteSet verfahren, allerdings kann
    // nicht geprüft werden, ob der anfordernde Knoten überhaupt
    // die Weiterleitungsknotentabellen benötigt.
    if (_pastryNode.isRouterNode())
    {
      int l = sharedPrefix(_pastryNode.getNodeId(),senderId);
      // sende nur die Reihen, die beide Knoten gemeinsam haben
      for (int row=0; row < l; row++)
      {
         _handler.sendPastryAdminMessage(
           new RouteRowResponseMessage(
             _pastryNode.getRoutingTable().getRow(row)
           )
         );
      }
    }
  }
```

Die Nachrichtenbehandlung im PastryAdminChannelFilter (Abbildung 5.1.3).

```
public void newEvent(Object receiver, RemoteEvent event) {
    Object oMessage = event.getMessage();
    if (!(oMessage instanceof ECNMessage))
    { // ist keine ECN Pastry Nachricht, also kann sie nicht behandelt werden.
      return;
    }
    int eventType = event.getType();
    if ( eventType == NODE_ID_REQUEST_MESSAGE_TYPE )
    { // Neuer Knoten möchte beitreten
      handleNodeIDRequest(event);
    }
    else if (eventType == LEAFSET_REQUEST_MESSAGE_TYPE)
    {
      handleLeafSet(event);
    }
    else if (eventType == ROUTE_ROW_REQUEST_MESSAGE_TYPE)
    {
      handleRouteSet(event);
    }
    ...
    ...
    else
    { // unbekanntes PastryAdministrations Event
      _pastryNode.receiveMessage(event.getMessage());
    }
```

```
}
```

5.2 Implementierung

Die Implementierung wird für die flexible Variante des EventChannelNetworks dargestellt. Hier muss nur noch um eine Ansteuerung der konkreten realisiert werden. Dies betrifft das Anlegen oder Anfragen schon vorhandener Nachrichtenkanäle. Eine Erzeugung eines `PushEventTransmitters` und eines `PushEventHandlers` , die die Sende-, bzw. Empfangseinheiten darstellen, ist nötig.

Initialisierung und Verwaltung Folgend wird die Erzeugung eines Nachrichtenkanals vorgestellt, wie sie innerhalb einer dynamischen Backend-Implementierung der Methode `ECBackend.createChannel` erfolgt.

```
  ...
  if (_ecnfactory == null)
  { // falls noch keine Fabrik vorhanden-> mit Rundruf-Sockets initialisieren
    initECNFactory(sockets);
  }
  else
  { // fragen , ob der Kanal schon vorhanden ist.
    channel = getChannel(symbol);
    if (channel != null)
    { // wenn ja, werden weitere physikalischen Netzwerkverbindungen
      // zum Kanal hinzugefügt.
      for (int i = 0; i < sockets.length; i++)
        if (!channel.containsSocket(sockets[i]))
          channel.addSocket(sockets[i]);
    }
    else
    { // Kanal ist nicht vorhanden -> anlegen
      channel = _ecnfactory.createChannel(channel_symbol,
                                          ...,
                                          sockets);
    }

    //Warteschlangen für die eintreffenden Nachrichten.
    PushEventReceiver.getPushEventReceiver()
       .addQueueNdActivities("queue for channel " + symbol, ...);
  }
  return channel;
}
```

Damit wird ein erster, zur Kommunikation bereiter Nachrichtenkanal erzeugt, bzw. abonniert und durch zusätzliche physikalischen Verbindungen erweitert. Folgend ist noch der Quelltextausschnitt, der die EventChannel-Fabrik initialisiert und anschliessen nach vorhandenen Nachrichtenkanälen sucht.

```
  ...
  ecnfactory = EventChannelFactory.getFactory(getAdminChannelSockets());
  // warten, bis aktive Kanäle gefunden wurden
  java.util.Enumeration channels = ecnfactory.getChannels(2000);
  ...
```

5.3 Hello World - Ein Beispiel

Abschließend wird noch ein minimal modifiziertes Beispiel aus der FreePastry-API herangezogen, um die Basisfunktionalitäten des nun aufgebauten Netzwerks zu demonstrieren. Lediglich die Auswahl des Pastry-Transport-Protokolls ist auszuwechseln, dass an Stelle von `PROTOCOL_SOCKET`das neu hinzugekommene `PROTOCOL_ECN` verwendet wird. Dieses Beispiel demonstiert die Basisfunktionalitäten der realisierten EventChannelNetwork Anbindung. Dazu werden Knoten erzeugt, welche durch Angabe von Verbindungsparametern (in diesem Fall einer UDP-Verbindung) zum ersten erzeugten Knoten den Bootvorgang durchlaufen. Im zweiten Schritt wird eine bestimmte Anzahl zufällig adressierter Nachrichten über das gebildete Gleiche-zu-Gleiche-Netzwerk versendet.

Die in diesem Abschnitt referenzierten Quelltextausschnitte sind zwecks Übersichtlichkeit im Anhang untergebracht. In Abbildung 24 ist eine minimale Pastry-Anwendung zu sehen. Mit einfachsten Mitteln lässt sich auch eine spezialisierte Nachricht (siehe Quelltextausschnitt 23) erstellen, die somit anwendungsspezifische Daten versenden und empfangen kann.

Nun muss noch die Applikation, mit einigen Parametern versehen gestartet werden. Eine Initialisierung der Werte findet nicht mittels Methodenparameter statt, sondern durch Nutzung der Environment-Funktionalitäten. Auf diese Weise werden die Rundrufadresse, Ports, oder etwa die minimalen, bzw. maximalen Ping-Intervalle spezifiziert.

6 Ausblick

Mit dieser Ausarbeitung wurde gezeigt, wie ein Gleiche-zu-Gleiche-System auf dem EventChannelNetwork prinzipiell realisiert werden kann. Auf diese Weise ist es möglich unterschiedlichste Plattformen über ein gemeinsames Protokoll zu verknüpfen. Jedoch ist eine konkrete Anpassung an bestimmte Aufgaben nötig. Davon abgesehen sind an dieser Stelle zunächst Tests und Performanz-Untersuchungen nötig, welche die Verfahren bewerten, oder eine modifizierte Form nahelegen. In diesem Zusammenhang kommt der am FZI in der Entwicklung befindliche „ModelChecker" zum Einsatz, welcher die gegebenen Protokolle auf Deadlock-Freiheit und weitere verteilte Abhängigkeiten prüfen soll. Dies sind essentielle Absicherungen, die nötig sind, um zu garantieren, dass keine unvorhergesehenen Zustände auftreten. Vor allem im Hinblick auf den Einsatz in sicherheits- oder geschäftskritischen Anwendungen sind diese Überprüfungen notwendig. Dennoch ist gezeigt worden, dass sich eine Gleiche-zu-Gleiche Schnittstelle basierend auf einer anonymen, asynchronen Nachrichtenkommunikation realisieren lässt.

Anpassung der die Pastry-Middleware erweiternden Protokolle

Bisher wurden lediglich die Pastry-Grundfunktionen auf den Nachrichtendienst abgebildet. Eine Anpassung beispielsweise des Scribe- oder der PAST-Protokolls (siehe Abschnitt 2.3.2) kann nach der, in dieser Studienarbeit vorgestellten Weise analog durchgeführt werden.

Anpassung an spezielle Umgebungen

Durch die in Abschnitt 4.5.2 empfohlene Hierarchisierung der Knoten kann Speicherplatz und Rechenleistung für die Anwendungsschicht eingespart werden. Durch eine weitere Reduzierung der Fähigkeiten und des damit verbundenen Ressourcenbedarfs ist es Möglich, das Gleiche-zu-Gleiche-System auch auf Kleinstrechner zu portieren. Bei mobilen Geräten, welche schnellen Veränderungen des zugrundeliegenden Netzwerks ausgesetzt sind, ist im Besonderen auf eine entsprechnde Adaptionsfrequenz zu achten. Ebenso wurde im Laufe diese Ausarbeitung auf die Wichtigkeit einer geeigneten Schlüssellange und eines konsistenten Schlüsselgenerators hingewiesen.

Caching

Um ein effizientes Zwischenspeicherung zu erreichen, ist es nötig durch Experimente die Verhaltensweise des hier vorgestellten Verfahrens zu untersuchen. Das betrifft nicht nur die Größe des

Zwischenspeichers, sondern auch die Ersetzungsstrategien. Eine Erweiterung der Pastry-Protokolle ist nötig um unterschiedliche Echtzeit- oder Konsistenz-Garantien zu unterstützen.

7 Anhang

```
public class HelloMsg extends Message {
  public Id target;  // Zielknotenschlüssel
  private int msgid; // Nachrichten-ID
  public boolean messageDirect = false;
  private NodeHandle src; // lokaler Schlüssel

  public HelloMsg(Address addr, NodeHandle src, Id tgt, int mid) {
    super(addr);
    target = tgt;
    msgid = mid;
    this.src = src;
  }

  public String getInfo() {
    // Gibt die enthaltenen Informationen zurück
  }

  public int getId() {
    // Die Nachrichten-ID
  }

  public String toString() {
    return "Hello #" + msgid + " from " + src.getId();
  }
}
```

Abbildung 23: Beispiel: Quelltext: Eine HelloWorld Message

```
public class HelloWorldApp extends PastryAppl {
  private int msgid = 0;
  private static Address addr = new HelloAddress();
  private static Credentials cred = new PermissiveCredentials();

  public HelloWorldApp(PastryNode pn) {
    super(pn);
  }

  // Sendet Nachrichten an zufällig ausgewählten Schlüssel
  public void sendRndMsg(RandomSource rng) {
    Id rndid = Id.makeRandomId(rng);

    Message msg = new HelloMsg(addr, thePastryNode.getLocalHandle(), rndid,
      ++msgid);
    routeMsg(rndid, msg, cred, new SendOptions());
  }

  public void messageForAppl(Message msg) {
    // Wird aufgerufen, wenn eine neue Nachricht angekommen ist.
  }

  public boolean enrouteMessage(Message msg, Id key, NodeId nextHop,
                                SendOptions opt) {
    // Wird aufgerufen, wenn über diesen Knoten eine Nachricht
    // weitergeleitet wurde.
    return true;
  }

  public void leafSetChange(NodeHandle nh, boolean wasAdded) {
     // Wird aufgerufen, wenn sich das Blattwerk geändert hat.
     * @param nh hinzugefügter/entfernter Schlüssel
     * @param wasAdded hinzugefügt (true) or entfernt (false)
  }

  public void routeSetChange(NodeHandle nh, boolean wasAdded) {
    /* Wird nach Änderung der Weiterleitungstabelle aufgerufen
     * @param nh hinzugefügter/entfernter Schlüssel
     * @param wasAdded hinzugefügt (true) or entfernt (false)
  }

  public void notifyReady() {
   Wird vom ECNPastryNode aufgerufen,
   sobald etwas ins Blattwerk eingefügt wurde.
  }
}
```

Abbildung 24: Beispiel: Quelltext: einer HelloWorld-Anwendung

```
  private static class HelloAddress implements Address {
    private int myCode = 0x1984abcd;

    public int hashCode() { return myCode; }

    public boolean equals(Object obj) {
      return (obj instanceof HelloAddress);
    }

    public String toString() { return "[HelloAddress]";  }
  }
```

Abbildung 25: Beispiel: Quelltext: Eine HelloWorld-Anwendung (Fortsetzung)

```
public static void main(String args[]) throws IOException {

    Environment env = new Environment("src/de/fzi/hija/es/pastry/testing/freepastry");
    doIinitstuff(args, env);
    DistHelloWorld driver = new DistHelloWorld(env);

    // create first node
    PastryNode pn = driver.makePastryNode(true);

    // Warten, bis der erste ECNPastryKnoten bereit ist, dann
    // können die anderen von diesem booten
    while (!pn.isReady()) {
        pn.wait();
    }

    for (int i = 1; i < numnodes; i++) {
      pn = driver.makePastryNode(false);
      while (!pn.isReady()) {
          pn.wait();
      }
    }

    for (int i = 0; i < nummsgs; i++) {
      for (int client = 0; client < driver.helloClients.size(); client++) {
        HelloWorldApp app = (HelloWorldApp) driver.helloClients.get(client);
        app.sendRndMsg(driver.environment.getRandomSource());
      }
    }
  }
```

Abbildung 26: Beispiel: Quelltext: Die Hauptroutine

Abbildungsverzeichnis

Literatur

[1] The gnutella protocol specification, 2000. http://dss.clip2.com/GnutellaProtocol04.pdf.

[2] Gurmeet singh manku, Juni 2004.

[3] Bittorrent economy, November 2005. http://www.bittorrent.com/bittorrentecon.pdf.

[4] The chord projekt, November 2005. http://pdos.csail.mit.edu/chord/.

[5] CoopNet Microsoft Research Project, November 2005. http://research.microsoft.com/ padmanab/projects/coopnet/.

[6] Coral content distribution network, Dezember 2005. http://www.coralcdn.org/.

[7] The Freenet Projekt, Oktober 2005. http://www.freenetproject.org/.

[8] Napster, Dezember 2005. http://www.napster.com/.

[9] Opennap, September 2005. http://opennap.sourceforge.net/.

[10] P-Grid - The Grid of Peers, November 2005. http://www.p-grid.org/.

[11] Pastry Internet Ressourcen, Dezember 2005. http://research.microsoft.com/~antr/Pastry.

[12] Real-Time Specification for Java, Dezember 2005. http://www.rtsj.org.

[13] Tapestry, Dezember 2005. http://p2p.cs.ucsb.edu/chimera/.

[14] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *IPTPS'03*, February 2003.

[15] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Oktober 2003.

[16] Ian Clarke. The freenet projekt, 1999. Original white paper by Ian Clarke, 1999.

[17] L. P. Cox and B. D. Noble. Pastiche: making backup cheap and easy. In *Fifth USENIX Symposium on Operating Systems Design and Implementation*, Dezember 2002.

[18] Peter Druschel and Antony Rowstron. PAST: A persistent and anonymous store. In *HotOS VIII*, May 2001.

[19] Peter Druschel John Kubiatowicz Frank Dabek, Ben Zhao and Ion Stoica. Towards a common api for structured p2p overlays. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Februar 2003.

[20] David K. Giord, John Jannotti, Kaashoek James, Kirk L. Johnson, and M. Frans. Overcast: Reliable multicasting with an overlay network, November 02 2000.

[21] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A decentralized peer-to-peer web cache. In *12th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, Juli 2002.

[22] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks, Juni 2004.

[23] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, pages 114–117, April 1965.

[24] Moni Naor and Udi Wieder. Novel architectures for P2P applications: the continuous-discrete approach, April 10 2003.

[25] Stefan Saroiu Marvin Theimer Nicholas J. A. Harvey, Michael B. Jones and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. März 2003.

[26] Antony Rowstron and Peter Druschel. Fig. 1. state of a hypothetical pastry node. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* [27], pages 329–350.

[27] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[28] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 188–201, Oktober 2001.

[29] Antony Rowstron and Peter Druschel. Table 1. pseudo code for pastry core routing algorithm. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* [27], pages 329–350.

[30] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In Jon Crowcroft and Markus Hofmann, editors, *Networked Group Communication, Third International COST264 Workshop (NGC'2001)*, volume 2233 of *Lecture Notes in Computer Science*, pages 30–43, November 2001.

[31] Marc Schanne. EventChannelNetwork. Dissertationsvorschlag Version 0.8, Frühjahr 2005. Nachrichtenkanal-Netzwerk als Basis zur Kommunikation in verteilten Anwendungen unter Echtzeitanforderungen.

[32] James Ellis Stephen Daniel and Tom Truscott (1980). Usenet - a general access unix® network., 1980.

[33] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[34] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. OverqoS: offering internet qoS using overlays. *Computer Communication Review*, 33(1):11–16, 2003.

[35] R. Rivest T. Cormen, C. Leiserson. Introduction to algorithms, 1990. ISBN 0262032937.