Bachelorarbeit

Nora Berg

HAMcast und Ariba/MCPO
Untersuchungen zur Komplexität von Servicekompositionen im Future Internet

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Nora Berg

HAMcast und Ariba/MCPO
Untersuchungen zur Komplexität von Servicekompositionen im Future Internet

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas C. Schmidt
Zweitgutachter: Prof. Dr. Hans H. Heitmann

Eingereicht am: 27. Mai 2013

Nora Berg

Thema der Arbeit
HAMcast und Ariba/MCPO
Untersuchungen zur Komplexität von Servicekompositionen im Future Internet

Stichworte
Future Internet, H∀Mcast, HAMcast, Komplexität, Ariba, MCPO, Netzwerk-Stack, Overlay, Komposition

Kurzzusammenfassung
Dieses Bachelorarbeit befasst sich mit der Komplexität bei Kompositionen von Future-Internet-Architekturen. Aus komponierten Future-Internet-Architekturen entstehen leicht komplexe Netzwerk-Stacks, die vielfältige Probleme mit sich bringen. Diese enstehen aus den unterschiedlichen Konzepten, denen sie folgen und die Abbildung dieser Konzepte aufeinander. Es wird untersucht, wodurch sich Komplexität in Netzwerkstacks auszeichnet und wie sie zustande kommt. Es werden konkrete Fragestellungen über die Komplexität der Future-Internet-Architekturen H∀Mcast und Ariba/MCPO sowie ihrer Komposition entwickelt. Diese werden durch Untersuchungen des Netzwerkstacks sowie eines Tests ausgewertet. Es wird dargestellt, dass Problemstellungen entstehen für die es keine optimale Lösung geben kann, und dass die Abbildung der einzelnen Konzepte aufeinander ein Schlüsselelement für die Komplexität sowie für die Leistung der kombinierten Stacks darstellt.

Title of the paper
HAMcast and Ariba/MCPO
A Study about the Complexity of Future Internet Service Compositions

Keywords
Future Internet, H∀Mcast, HAMcast, Complexity, Ariba, MCPO, Netzwerk-Stack, Overlay, Composition

Abstract
This thesis discusses the complexity of compositions of Future Internet schemes. Combined Future Internet schemes easily become complex network stacks, which entail diverse problems. They arise from different concepts and the mapping in between. This work analyses the complexity based on the Future Internet schemes H∀Mcast and Ariba/MCPO and the wrapper module to combine them. Characteristics of complex network stacks are evaluated, and lead to concrete questions about the complexity of the combined network stack. An evaluation

is performed and its results confirm that problems in the mapping between Future Internet concepts arise, which cannot be solved optimally and that this mapping is a key element for performance in combined network stacks.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In den letzten Jahren wurden einige Future-Internet-Konzepte entwickelt, die es einem Anwendungsprogrammierer erleichtern, verteilte Anwendungen zu entwickeln. Diese Konzepte stellen sich häufig auftretenden Problemen, wie dem Überwinden von protokollspezifischen Grenzen in Multi- und Unicast oder dem Überwinden von NATs und Firewalls. Eine einzelne Future-Internet-Architektur behandelt im Allgemeinen genau eine Problemstellung dieser Art und liegt in Form einer Zwischenebene zwischen Netzwerk-Stack und Anwendung vor. Da ein Anwendungsprogrammierer aber vielen Probleme gegenübersteht, wäre es für ihn von Vorteil, gleich mehrere dieser Architekturen benutzen zu können. Ein Ansatz ist deshalb, verschiedene dieser Future-Internet-Architekturen zu kombinieren, um so eine Architektur zu erschaffen, die mehrere Probleme behebt. Die Konzepte, nach denen die einzelnen Future-Internet-Architekturen entwickelt werden, haben unterschiedliche Ziele, bieten unterschiedliche Eigenschaften und benötigen unterschiedliche Voraussetzungen, um ihre Dienste korrekt auszuführen.
Aus diesem Grund kann es bei der Komposition verschiedener Future-Internet-Architekturen zu vielerlei Komplikationen und Problemen kommen. Weiterhin entstehen durch die Komposition schnell komplexe Software-Stacks, die sich durch viele interne Abhängigkeiten auszeichnen.

In dieser Arbeit wurde die Komplexität komponierter Future-Internet-Architekturen anhand der Komposition von H∀Mcast und Ariba/MCPO untersucht. H∀Mcast ist eine Future-Internet-Architektur, die sich mit der Vernetzung verschiedener Multicast-Technologien beschäftigt. Es existieren eine Reihe von Multicast-Protokollen die nicht überall im Internet funktionieren, da sie oft die Unterstützung von Routern und Zwischen-Hops benötigen. H∀Mcast vernetzt diese verschiedenen Protokolle, damit diese untereinander Daten austauschen können. Weiterhin bietet H∀Mcast dem Anwendungsprogrammierer eine allgemeine Multicast-Schnittstelle an, unter der verschiedene Multicast-Technologien angesprochen werden. Dies erlaubt dem Anwendungsprogrammierer technologieunabhängige Multicast-Anwendungen zu entwickeln.
Ariba/MCPO besteht aus zwei Bestandteilen. Ariba stellt dem Benutzer eine Unicast-Schnittstelle zur Verfügung mit der sich transparent Protkollheterogenität sowie Middleboxes wie NATs und Firewalls verbergen lassen. Darauf aufbauend stellt MCPO ein Multicast-Overlay bereit,

welches die Verbindungen von Ariba nutzt.
In der Komposition dieser beiden Future-Internet-Architekturen wurde Ariba/MCPO als Multicast-Technologie in H∀Mcast integriert.

Diese Arbeit evaluiert die Komplexität beim Zusammenspiel dieser Architekturen. Dazu werden im zweiten Kapitel die grundlegenden Future-Internet-Architekturen H∀Mcast und Ariba/MCPO sowie das Adaptermodul, welches die dazugehörigen Schnittstellen verbindet, vorgestellt. Danach wird ein Überblick über das Stichwort „Komplexität“ gegeben (Kapitel 3). Dazu gehört wie sich Komplexität im kombinierten Netzwerk-Stack bemerkbar macht und welche Aspekte die Komplexität erhöhen. Daraus ergeben sich die Problemstellungen in Kapitel 4. Hier wird erläutert, wie sich die Komplexitätsaspekte in H∀Mcast und Ariba/MCPO auswirken, wo konkrete Probleme auftreten, welche Anforderung das Adaptermodul erfüllen muss, damit die Komposition gut funktioniert. Es werden konkrete Fragestellungen entworfen um die Komplexität der Komposition zu überprüfen. Im Evaluationskapitel werden diese Problemstellungen getestet und ausgewertet.

2 Grundlagen

2.1 H∀Mcast

Im gegenwärtigen Internet wird Multicast nicht einheitlich unterstützt. Es gibt Protokolle und Anwendungen für alle Ebenen des Netzwerk-Stacks und diese sind untereinander im Allgemeinen nicht kompatibel. Möchte ein Anwendungsprogrammierer Multicast zur Datenverteilung benutzen, ist er gezwungen sich zur Entwicklungszeit für ein oder mehrere Multicast-Technologien zu entscheiden und diesen Kommunikationskanal direkt mit zu implementieren und im laufenden Betrieb zu warten. Da nicht überall im Internet die gleichen Multicast-Protokolle unterstützt werden, ist nicht sichergestellt, dass alle Endknoten erreicht werden können. H∀Mcast ist ein hybrider Multicast-Dienst (Meiling et al., 2012), der das Ziel verfolgt, Multicast an möglichst vielen Orten durch eine allgemeine Multicast-Schnittstelle (Wählisch et al., 2013) verfügbar zu machen. Das Grundkonzept ist, eine allgemeine Schnittstelle als Bibliothek zur Verfügung zu stellen, welche die Aufrufe für die einzelnen Multicast-Protokolle übersetzt.

Damit Multicast-Kommunikation zwischen verschiedenen Technologien möglich ist, muss es eine einheitliche Adressierung der Gruppen geben. Zwei Knoten aus verschiedenen Technologien, die verschiedene Adressierungen benutzen und dieselbe Gruppe abonniert haben, sollen auch die gleichen Daten erhalten. Damit dies möglich ist, genügt es nicht, nur die allgemeinen Funktionen wie „join“ und „leave“ auf die Funktionen der einzelnen Multicast-Technologien abzubilden. Weiterhin muss ein allgemeines Adressierungsschema unterstützt werden, welches auf die speziellen Adressierungen der unterliegenden Multicast-Technologien abgebildet wird. Die angebotene Bibliothek benutzt dieses Adressierungsschema für Anwendungen und macht sie damit unabhängig von der unterliegenden Technologie. Um letztendlich Daten von einer Multicast-Domäne in eine andere zu transferieren, gibt es ausgewählte Knoten (Interdomain Multicast Gateways, kurz: IMGs), die verschiedene Multicast-Technologien unterstützen und die Daten zwischen diesen vermitteln (siehe Abbildung 2.1).

Die angebotene Bibliothek benutzt dieses Adressierungsschema für Anwendungen und macht sie damit unabhängig von der unterliegenden Technologie.

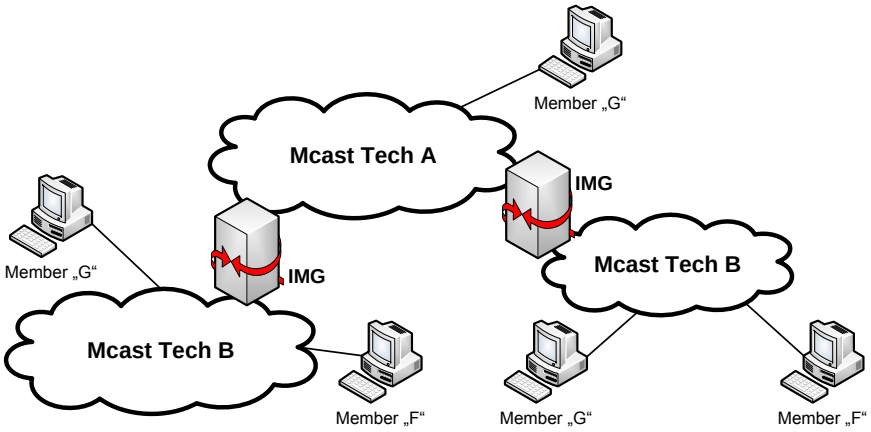

Abbildung 2.1: H∀Mcast Multicast-Netzwerk

Auf diese Weise ist es möglich unabhängig von der Multicast-Technologie Anwendungen zu entwickeln, die Multicast benutzen. Die Multicastechnologien werden in H∀Mcast als eigene Module implementiert und können geändert, entfernt oder hinzugefügt werden, ohne dass die Anwendung davon betroffen ist.

Implementierung

H∀Mcast setzt sich aus drei Hauptkomponenten (vgl. Abbildung 2.2) zusammen: Die API-Schnittelle *libHAMcast*, die *Middleware*, und die Technologiemodule (*Technology Interfaces*), welche das Konzept des hybriden, adaptiven Multicast umsetzen.

Die H∀Mcast-API-Library ist die Schnittstelle, die der Anwendungsprogrammierer nutzt, um die Multicastfunktionen auszuführen. Die Schnittstelle bietet einen Multicastsocket, der die Multicastfunktionen der allgemeinen Multicast-Schnittslle, wie z.B. „join“ und „send“, sowie ein asynchrones „receive“, umsetzt. Diese Aufrufe werden von der libHAMcast per IPC an die H∀Mcast-Middleware übergegeben.

Die Middleware läuft pro System genau einmal und kann dabei für mehrere Anwendungen zuständig sein. Für jede Anwendung nimmt sie die Anfragen über einen H∀Mcast-Socket der API entgegen, und gibt die Anfragen an die entsprechenden Technologiemodule weiter.

Die Technologiemodule bilden die Schnittstelle zwischen den Multicast-Technologien und der H∀Mcast-API. In ihnen werden allgemeinen Funktionen der API sowie die Adressierungen auf die Funktionen und Adressierungen der Multicast-Technologien abgebildet. Die Technologiemodule liegen als C/C++-Bibliotheken vor und werden von der Middleware geladen.

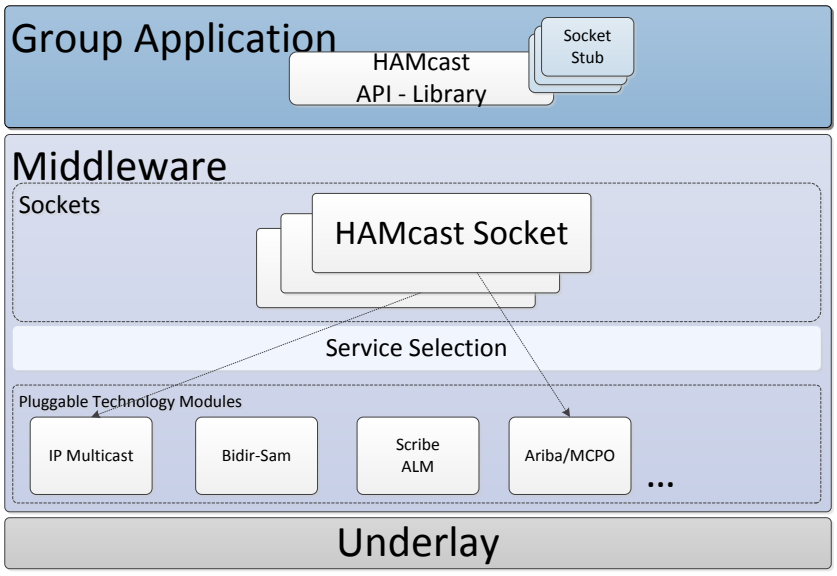

Abbildung 2.2: H∀Mcast-Netzwerkstack

2.2 Ariba/MCPO

Ariba/MCPO ist eine Komposition von zwei Future-Internet-Konzepten. Ariba stellt ein zuverlässiges Unicast-Overlay bereit und ist eine eigenständige Future-Internet-Architektur. MCPO ist eine Erweiterung Aribas um einen Multicast-Dienst. Ariba und MCPO sind C++-Bibliotheken und stellen ihre Dienste mittels einer Programmierschnittstelle bereit.

2.2.1 Ariba

Ariba (Hübsch et al., 2010) stellt zuverlässige Ende-zu-Ende-Verbindungen auf Basis eines Overlay-Netzes (SpoVNet) bereit. Die Verbindungen des SpoVNets haben die Eigenschaft hinter Middleboxes und über Protokollgrenzen hinweg kommunizieren zu können, sowie die Ende-zu-Ende-Verbindungen zu Knoten herzustellen, die sich in anderen physischen Netzarten befinden (z.B. über Bluetooth verbunden sind). Um die Verbindungen bereitzustellen, baut Ariba ein Overlay-Netz (SpoVNet (Bless et al., 2008)) auf. Ein SpoVNet ist anwendungsabhängig und identifiziert sich über einen SpoVNet-Namen. Auch wenn ein Knoten mehrere Adressen haben kann (für jedes genutzte Netzwerkinterface eine), identifiziert er sich innerhalb eines SpoVNets über genau einen selbstgewählten Namen. Auf diese Weise erfolgt ein ID/Locator-Split. Die Adressierung der Daten erfolgt ausschließlich über die Knoten-ID, sodass Nachrichten

unabhängig von der konkreten Adresse des Netzwerkinterfaces zugestellt werden können. Auf diese Weise wird ebenfalls Mobilität der Endknoten ermöglicht.

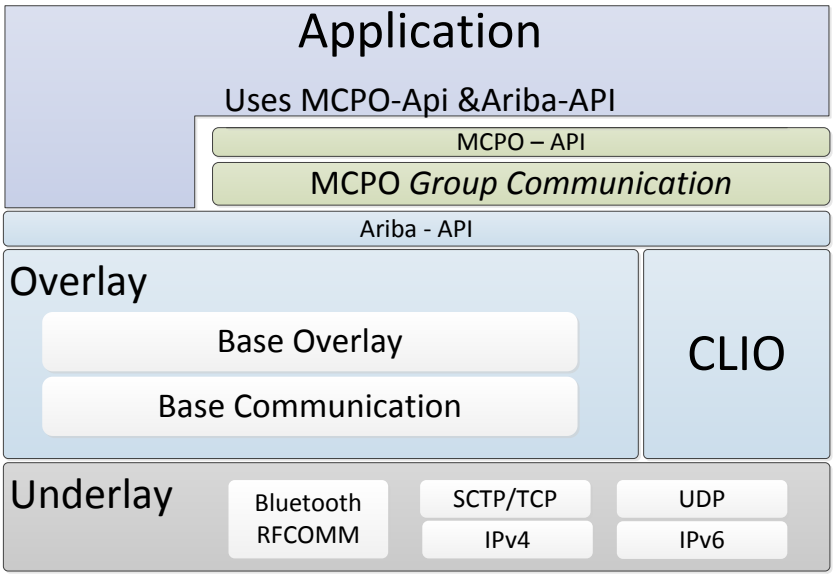

Abbildung 2.3: Ariba/MCPO-Netzwerk-Stack

Um die Eigenschaften des ID/Locator-Splits und der zuverlässigen Ende-zu-Ende-Verbindung zu ermöglichen benutzt Ariba zwei Hauptkomponenten.

Die Base Communication ist für das Erstellen und Aufrechterhalten der Transportverbindungen zuständig, die über das Underlay laufen. Jeder Knoten hat mindestens eine Netzwerkverbindung. Für jede Netzwerkverbindung hat er eine Adresse, die ihn in diesem Netzwerk identifiziert. Die Base Communication stellt den sogenannten *Endpoint Descriptor* bereit, der im wesentlichen eine Menge der Adressen ist, unter der ein bestimmter Endpunkt erreicht werden kann. Um die Heterogenität der unterliegenden Netzwerke zu überwinden, stehen ausgewählte Knoten als *NAT-Relays* (zur Überwindung der Middleboxes) oder *Protokoll-Relays* (zur Überwindung von Protokolldifferenzen) zur Verfügung. Dadurch entstehen zwei Arten Verbindungen. *Direkte Verbindungen*, sind Verbindungen, über die zwei Knoten direkt kommunizieren können. *Indirekte Verbindungen* sind die Verbindungen, die über die verschiedenen Relays kommunizieren. Damit das Ende-zu-Ende-Prinzip erhalten bleibt, werden indirekte Verbindung über Relays als Tunnel realisiert. Die Relays, die benötigt werden um einen Knoten hinter einer Middlebox, bzw. über ein anderes Protokoll, zu erreichen, sind im Endpunktdeskriptor dieses Knotens enthalten. Die NAT-Relays und deren Discovery funktionieren dabei

ähnliche wie TURN (Mahy et al., 2010). Protokollrelays werden mittels Gossiping-Approach verbreitet (siehe (Hübsch et al., 2010)).
Die Hauptaufgabe der Base Communication besteht darin, für überliegende Schichten von den den netzwerkspezifischen Adressierungen zu abstrahieren und Verbindungen über die Technologien im Underlay bereitzustellen.

Das Base Overlay hat die Hauptaufgabe die Verbindungen zwischen den einzelnen Knoten herzustellen und zu erhalten. Dafür ordnet es KnotenIDs die dazugehörigen Endpunktdeskriptoren zu. Dies geschieht mittels einer Distributed Hash Table (DHT), welche die KnotenID als Key-Wert nutzt. Dafür sendet das Base Overlay eine „descriptor query message“ an die KnotenID. Sobald es eine Antwort mit dem Endpunkt-Deskriptor des gewünschten Knotens erhält, wird mittels der Base Communication eine Transportverbindung zu einer passenden Adresse erstellt (direkt oder indirekt mittels Relay) und die Daten werden übertragen.

2.2.2 MCPO

MCPO (Institut für Telematik, 2013) stellt auf Grundlage des Ariba-Overlays Multicast-Funktionalität zu Verfügung. Dafür benutzt es die Verbindungen von Ariba um einen Multicastverteilbaum ähnlich zu „NICE“ (Banerjee et al., 2002) aufzubauen. Die Nachrichten werden im Verteilbaum geflutet und an den Endpunkten durch ein Portkonzept gefiltert.

Der Verteilbaum von MCPO ist hierarchisch auf verschiedenen Ebenen aufgebaut (siehe Abbildung 2.4). Alle teilnehmenden Knoten werden in Cluster auf Ebene 0 unterteilt. Jedes Cluster wählt einen Leader, die sich dann ebenfalls in der Ebene 1 befindet. Ebene 1 wird wiederum in Cluster unterteilt, die jeweils einen Leader wählen der sich in der Ebene darüber befindet und so weiter.

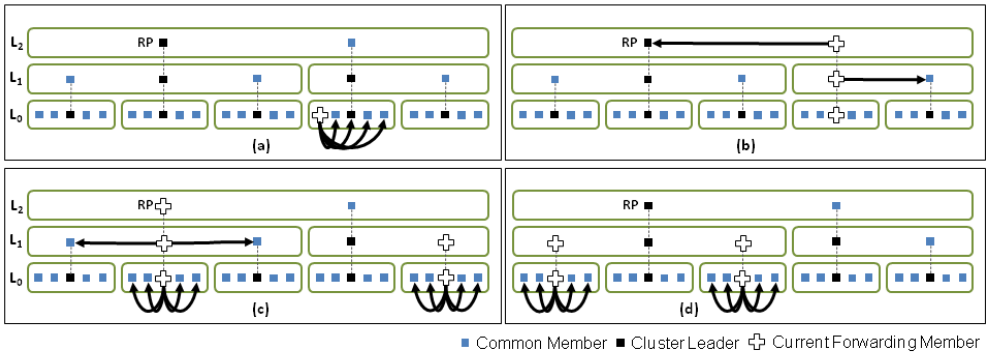

Abbildung 2.4: MCPO Verteilbaum[1]

Um die Daten zu verteilen, sendet ein Knoten nur an sein eigenes Cluster auf Ebene 0 (2.4a). Der Clusterleader sendet diese Daten dann weiter in sein Cluster in Ebene 1 (2.4b). Dessen Leader sendet die Daten weiter in die nächsthöhere Ebene (2.4b) während die restlichen Knoten im Cluster diese Daten an die Knoten auf der Ebene darunter weitergeben(2.4c,2.4d) (siehe Abbildung 2.4). Auf diese Weise werden die Daten in MCPO verteilt.

2.3 Adaptermodul

Das Adaptermodul bindet das Ariba/MCPO-Multicast-Netz in H∀Mcast ein. Auf diese Weise kann H∀Mcast in Ariba/MCPO-Multicast-Domänen kommunizieren und die Nachrichten einer solchen Domäne in anderen Multicast-Netzen verfügbar machen. Um das zu erreichen, wird Ariba/MCPO als Modul in H∀Mcast integriert.

Dafür implementiert das Adaptermodul auf der einen Seite das Modul-Interface von H∀Mcast und auf der anderen Seite die Ariba/MCPO-Schnittstelle. Der Adapter hat die Aufgabe die Funktionen der Technologie-Interfaces von H∀Mcast auf die Funktionen der Ariba/MCPO Schnittstelle abzubilden. Dabei besteht die Schwierigkeit die einzelnen Konzepte von H∀Mcast und Ariba/MCPO beizubehalten, und so zu nutzen, dass die Multicast-Funktion möglichst effizient ausgeführt wird.

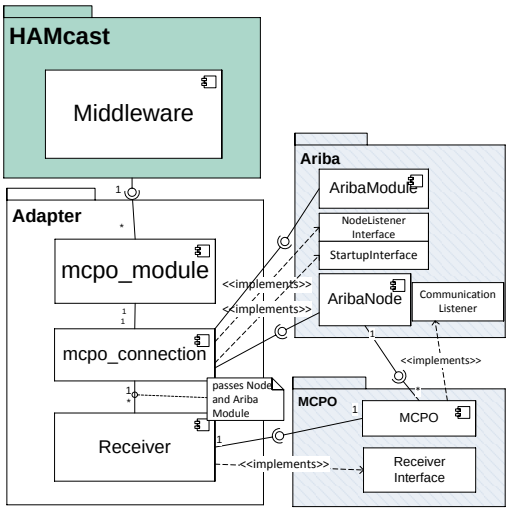

Abbildung 2.5: Die Einbettung des Adapter-Moduls zwischen H∀Mcast und Ariba/MCPO

[1]http://www.ariba-underlay.org/wiki/Documentation/MCPO

Abbildung 2.5 beschreibt die Einbettung des Adapter-Moduls zwischen die Schnittstellen von H∀Mcast und Ariba/MCPO. Die Middleware bietet eine Schnittstelle an, die von der *mcpo_module*-Klasse implementiert wird. *Mcpo_connection*, sowie *Receiver*, implementieren die Schnittstelle zu Ariba/MCPO. Das Adaptermodul liegt als C++-Bibliothek vor, und wird von der H∀Mcast-Middleware während der Middleware-Initialisierungsphase eingebunden.

Durch die Kombination dieser Future-Internet-Architekturen entsteht ein Stack aus mehreren Overlay-Netzen. Ariba bildet dabei das unterste Overlay-Netz, MCPO baut einen Multicast-Verteilbaum darüber. H∀Mcast ist in der Lage verschiedene Multicast-Technologien zu verbinden, sodass diese untereinander Daten austauschen können. Aus diesem Grund bildet H∀Mcast das oberste Overlay-Netz als Zusammenschluss von Multicast-Domänen.

Das Adaptermodul ist nicht nur dafür zuständig, die Funktionen der H∀Mcast-API an Ariba und MCPO weiterzuleiten, sondern auch, die Adressierungsschemata der Konzepte aufeinander abzubilden.

3 Komplexität

Komplexität, wie sie hier untersucht werden soll, bezieht sich auf die Komplextiät von Netzwerk-Stack-Architekturen, -konzepten und ihrer Komposition. Sie ist abzugrenzen von der allgemeinen Laufzeitkomplexität, die sich mit dem Verhalten einzelner Algorithmen beschäftigt. Nicht betrachtet werden implementierungsspezifische Optimierungen, die sich mit Codeänderungen ausführen lassen.

Die Komplexität einer Netzwerkarchitektur teilt sich in die Komplexität des Kommunikationsablaufs und die Komplexität der Steuerung. Erstere beinhaltet die Fragestellung, durch wie viele Netzwerk-Stack-Ebenen die Nachrichten tatsächlich fließen und wie oft sie dabei in ein neues Datenformat umgeschrieben werden. Die Komplexität der Steuerung beinhaltet die Durchlässigkeit der Ebenen hinsichtlich der Konfigurationsinformationen, die Vielfalt der Funktionalitäten und damit die erhöhte Wahrscheinlichkeit der nicht klar getrennten Aufgabenbereiche. Die Komplexität der Steuerung beinhaltet ebenfalls den allgemeinen Aufbau der Future-Internet-Architekturen und die Fragestellung, auf welche Weise die Architekturen verknüpft sind.

Als weiterer Komplexitätsaspekt wird die Verdrängung von Komplexität auf die Anwendungsebene durch das Ende-zu-Ende-Prinzip diskutiert.

Wie komplex eine Netzwerkarchitektur letztendlich ist, lässt sich anhand einiger Indizien ausmachen. Bestimmte Zusammenhänge erhöhen die Komplexität für den Benutzer einer Schnittstelle sowie die Komplexität innerhalb einer Future-Internet-Bibliothek.
Nachfolgend werden Aspekte vorgestellt, auf die Ariba/MCPO, H∀Mcast und die Komposition dieser beiden Konzepte mittels Adapter untersucht werden.

3.1 Aufhebung der strikten Trennung von Netzwerk-Stack-Ebenen steigert die Komplexität

Future Internet-Anwendungen stellen neue Funktionalitäten für verteilte Anwendungen bereit. Um diese Funktionen anzubieten, werden oft Informationen von verschiedenen Ebenen des

Netzwerk-Stacks benötigt. Das führt zu Komponenten, die sich logisch zwischen Ebenen des ISO-OSI-Netzwerkmodells befinden, oder sich vertikal zu den Ebenen ausrichten. Auf diese Weise wird mit den Abstraktionsebenen, die das Schichtenmodell vorsieht, gebrochen.

Um eine logische Trennung von Zuständigkeiten innerhalb der Future-Internet-Anwendungen zu erhalten, sind diese modular aufgebaut, sodass jede Ebene möglichst genau eine Komponente umfasst (auch solche, die mit verschiedenen Ebenen kommunizieren). Liegen mehrere Komponenten auf einer Ebene und bietet die Ebene darunter in ihrer Schnittstelle globale Steuerungselemente an, dann sind diese Komponenten in der Lage sich gegenseitig zu beeinflussen. Dies geschieht, sobald eine der Komponenten das Verhalten der unteren Ebenen ändert.

Dadurch muss ein Anwendungsprogrammierer, der die angebotenen Schnittstellen nutzt, genau wissen, wie sich seine Aktionen innerhalb der genutzten Future-Internet-Anwendung auf die anderen Ebenen/Komponenten auswirken. Die Komplexität von Netzwerkarchitekturen nimmt durch die klare Trennung und Abschirmung der Ebenen ab, steigert jedoch die Gefahr erhöhten Mehraufwands auf oberen Ebenen durch fehlende Informationen.

3.2 Komplexität anhand der Funktionalität der Komponenten und Schnittstellen

Ein Hinweis auf ein komplexes System ist die Breite des Zuständigkeitsbereichs (vgl. Bush and Meyer (2002)) und inwiefern sich die Funktionalitäten darin gegenseitig beeinflussen. Je weiter und detaillierter sich der Aufgabenbereich auffächert, der mit einer Future-Internet-Anwendung bearbeitet werden kann, desto höher ist die Wahrscheinlichkeit, dass intern Abhängigkeiten bestehen, die berücksichtigt werden müssen. Da diese Funktionsbreite verwaltet werden muss, wird die Anzahl der Konfigurationsparameter für das System als Indiz über die Komplexität angesehen. Je mehr Parameter z. B. vor Systemstart eingestellt werden müssen, desto mehr Abhängigkeiten gibt es von diesen Konfigurationsdaten und desto komplexer wird dann auch die Schnittstelle. Bei komplexen Netzwerk-Stacks ist es notwendig, mit so wenig Konfigurationsparametern wie möglich auszukommen (vgl. Carpenter (1996)). Ausgangspunkt für die Anzahl der Konfigurationen ist die Anzahl der Parameter, die von den Future-Internet-Systemen benötigt werden, um ein laufendes System aufzubauen. Das ist nicht zwangsläufig die Anzahl der Konfigurationsparameter, die im Vorfeld bekannt sein müssen. Sie können auch z. B. zur Initialisierungszeit berechnet werden. Wenn das System von diesen errechneten Konfigurationen abhängt, wird es aber allein durch die Berechnung nicht weniger

komplex. Wie diese Konfigurationsparameter zur Verfügung gestellt werden, ist letztendlich implementationsabhängig.

Ariba/MCPO benötigt für Ariba die Informationen, welchem SpoVNet beigetreten werden soll, welcher Endpunkt benutzt werden soll und wie ein Rendezvous-Punkt herausgefunden werden soll. MCPO zusätzlich eine ServiceID zur Identifikation in Ariba.
H∀Mcast benötigt zum laufenden System die Information, welches Technologiemodul geladen werden soll und wo sich dieses befindet. Weitere Konfigurationsdaten sind modulabhängig und werden dem Modul zu Initialisierungszeit übergeben.

In der Kompositionen von Future-Internet-Architekturen besteht eine Herausforderung darin, möglichst viele Konfigurationen aufeinander abzubilden, um zusätzliche Komplexität zu verhindern. Die Abbildung der Konfigurationsdaten sollte weitestgehend automatisch passieren und konsistent den Funktionsbereich abdecken. Hierbei besteht die Schwierigkeit, die Daten so abzubilden, dass die Funktionalität so erhalten bleibt, wie sie von dem jeweiligen Future-Internet-Konzept-Konzept vorgesehen ist.
Dies ist ein Kernpunkt der Komposition von Future-Internet-Konzepten. Diese Abbildung der Funktionen entscheidet nicht nur über die Beibehaltung der Konzepte, sondern ebenfalls über den Mehraufwand der bei jedem Aufruf einer bestimmten Funktion entsteht.

Ein weiterer Aspekt der Funktionalität betrifft das Hinzufügen von Funktionen zu einem System (Carpenter, 1996). Je aufwendiger es ist eine neue Funktion zu den Future-Internet-Architekturen hinzuzufügen, desto komplexer ist ein System. Dies ist ebenfalls abhängig vom Aufbau der Komponenten. Eine höhere Abhängigkeit zwischen den einzelnen Komponenten (Kohäsion), impliziert, dass mehr Komponenten in den Änderungsprozess inbegriffen sind und dementsprechend angepasst werden müssen.

3.3 Das Ende-zu-Ende-Prinzip verschiebt Komplexität auf die Anwendungsebene

Das Ende-zu-Ende-Design-Prinzip besteht darin, Funktionen, die von einer bestimmten Anwendung gebraucht werden, auch auf der Ebene dieser Anwendung zu programmieren. Weiterhin muss auf dieser Ebene sichergestellt werden, dass gewünschte Funktionen korrekt ausgeführt werden bzw. eventuell entstandene Fehler toleriert werden. Um Ende-zu-Ende-Verbindungen zu realisieren, wird die Schnittstelle zu der Ebene darunter genutzt, die Funktionen, z. B. für den Kommunikationskanal, anbietet. Das Wissen darüber, welche Funktionen die Anwendung direkt benötigt und auf welche Weise diese umgesetzt werden müssen, ist nur in der Anwendung selber enthalten. Deswegen sind Funktionen der unteren Schichten abstrakt und anwendungs-

unabhängig gehalten, da ohne das Wissen über die Anwendung bestimmte Funktionen nicht passgenau zur Verfügung gestellt werden können (Saltzer et al., 1984).

Die Ebenen im kombinierten Netzwerk-Stack garantieren für ihre Funktionen bestimmte Eigenschaften. Dennoch müssen gewünschte Eigenschaften von den Anwendungen überprüft werden, da nicht alle Fehler ausgeschlossen sind. Mit Funktionen wie „retry“, „buffering“, „timeout“ und mehreren Kopien von Datensätzen (Saltzer et al., 1984), können Übertragungsfehler auf ein akzeptables Maß reduziert werden. Manche Fehler, wie z.B. der Absturz eines Computers mit dem gerade kommuniziert wird, können jedoch nicht von den unterliegenden Ebenen vor der Anwendung transparent verborgen werden. Es ergibt sich daraus die Fragestellung, an welcher Stelle im Netzwerk-Stack die Funktionalität am effektivsten implementiert werden kann, sodass sie einerseits einfach zu nutzen ist, und andererseits den Anwendungen, die diese Eigenschaft nicht benötigen, nicht im Wege steht.

Beispielsweise, wenn eine zuverlässige Verbindung benötigt wird, reicht es nicht eine verteilte Anwendung über TCP kommunizieren zu lassen und zu erwarten, dass die Anwendung zuverlässig funktioniert, und alle Daten immer überall ankommen. Zusätzlich muss auf allen überliegenden Ebenen Aufwand betrieben werden, damit diese Eigenschaft beibehalten wird.

Bei Kompositionen von Netzwerk-Stacks besteht die Gefahr, das bestimmter Aufwand an der falschen Stelle betrieben wird. Betrachtet man folgenden Aufbau: Ein unterliegendes System (nachfolgend: S1) minimiert einen Fehler und stellt damit eine bestimmte Eigenschaft bereit (z.B. Zuverlässigkeit). Ein darüberliegendes System (S2) ignoriert diese Eigenschaft, sie geht verloren und ist für die überliegende Anwendung (S3) unsichtbar. S3 benötigt diese Eigenschaft wieder und muss fast alles neu implementieren, was bereits sichergestellt worden ist. In diesem Fall wäre es sinnvoller, entweder auf die mittlere Ebene (S2) zu verzichten, sodass die Anwendung auf die Fehlerquellen der ersten Ebene (S1) reagieren kann, oder die gewünschte Eigenschaft direkt auf einer höheren Ebene zu implementieren.
Bei der Kombination von Future-Internet-Architekturen kann man sich nicht aussuchen, auf welchen Ebenen die Funktionen implementiert sind und auf welchen Ebenen welche Eigenschaften gebraucht werden. Die Vernachlässigung, genau wie die für die Anwendung unnötigen aber sichergestellten Eigenschaften, kosten im Zweifelsfall Leistung und erhöhen die Komplexität des Netzwerk-Stacks (letzteres z. B. wenn die Funktion einer Schnittstelle implementiert werden muss, obwohl sie nicht benötigt wird).

Im Bezug auf das Ende-zu-Ende-Prinzip wird bei der Entwicklung einer Future-Internet-Architektur zusätzliche Komplexität auf die Anwendungsebene verschoben (z. B. muss die Anwendungsschnittstelle eines Overlays verstanden, und der Zutritt zu diesem Overlay ver-

waltet werden). Wenn man Future-Internet-Architekturen kombiniert, dann müssen alle diese zusätzlichen Verwaltungsaufgaben bedient werden.

Die Komplexität kann also durch das Ende-zu-Ende - Prinzip erhöht werden, da sämtliche Komplexität auf die oberste Anwendungsebene ausgelagert wird. Das bedeutet, dass die darunterliegenden Ebenen nur Fehler kaschieren dürfen, wenn dabei keine Eigenschaften der unteren Ebenen verloren gehen. Ist dies nicht der Fall, und eine obere Ebene benötigt genau diese Eigenschaft, muss diese Eigenschaft von der Anwendungen grundsätzlich neu implementiert werden. Im Fall von z.B. Zuverlässigkeit führt das zu einer doppelten Implementation von Zuverlässigkeitsmechanismen (z. B. Protokollen), wo sonst eine Fehlerüberwachung und -behandlung ausgereicht hätte.

4 Problemstellungen

Bei der Servicekomposition von Future-Internet-Architekturen kommt es zu verschiedenen Herausforderungen. Es sind Probleme, die aus den Komplexitätsaspekten (vgl. Kapitel 3) entstehen. Sie beruhen auf widersprüchlichen Zielen und Eigenschaften der Future-Internet-Konzepte und Aufgaben, die deswegen unnötig oft oder gar nicht ausgeführt werden müssen. Weitere Herausforderungen entstehen aus komplexen Netzwerk-Stack-Architekturen, die Ebenen nicht komplett abschirmen und zu Seiteneffekten führen können und aus speziellen Anforderungen einer Future-Internet-Architektur, die von überliegenden Ebenen berücksichtigt werden müssen.

4.1 Adressierung

Die Komplexitätsbeschreibung in Kapitel 3 zeigt, dass Anzahl der Konfigurationsparameter als Maß für Komplexität. Die Adressierungen in H∀Mcast und Ariba/MCPO stellen eben solche Konfigurationsparameter dar. Die Abbildung von Konfigurationen wird zur Reduzierung der Komplexität genutzt. Da sich die Gruppenadressierung in H∀Mcast und Ariba/MCPO unterscheidet, müssen diese Konfigurationstypen aufeinander abgebildet werden.

In H∀Mcast werden die Gruppen durch H∀Mcast-URI adressiert. Diese folgt der Form:

```
scheme :// group @ instantiation : port / credentials
```

Hierbei beschreibt der *scheme* einen bestimmten Namensraum, die *group* einen bestimmten Gruppennamen, die *instantiation* eine optionale Quelle (z.B. bei Source Specific Multicast). Der *port* beschreibt das allgemeine Port-Konzept und die optionalen *credentials* stehen für gruppen- und anwendungsspezifische Informationen (z.B. für Authentifizierung) zur Verfügung.

In Ariba/MCPO gibt es mehrere Ebenen von Adressierungen, sodass jede Gruppe von mehreren Identifikatoren beschrieben wird.

Ariba benutzt für eine unterste logische Ebene einen SpoVNet-Namen, der das Peer-to-Peer-Overlay beschreibt. Der Knotenname (Nodename) ist frei wählbar und identifiziert einen bestimmten Knoten innerhalb des SpoVNets. Mithilfe dieser Identifikatoren sind alle Knoten innerhalb eines Netzwerkes erreichbar und Ariba bietet nun nach oben hin der Anwendung die

Möglichkeit dieses Netzwerk zu benutzen. In Ariba werden Anwendungen per ServiceID (unsigned Integer) identifiziert, sodass Ariba Nachrichten einer bestimmten Anwendung zuordnen kann (Portkonzept).

MCPO ist eine solche Anwendung und benötigt deswegen eine ServiceID für Ariba. Weiterhin stellt MCPO Gruppenkommunikation zur Verfügung, und bietet deswegen eine Adressierung von Multicast-Gruppen an. Diese wird innerhalb MCPOs ebenfalls als ServiceID gehandhabt.

Die Problemstellung besteht nun darin, die H∀Mcast-URI effektiv auf die Identifikatoren im Ariba/MCPO-Stack abzubilden, sodass die Konzepte beider Futur-Internet-Architekturen korrekt erhalten bleiben. Dabei soll möglichst wenig der Funktionalitäten eingeschränkt und ein hoher Datendurchsatz erzielt werden. Die Vor- und Nachteile möglicher und vorhandener Abbildungen werden evaluiert und abgewogen.

4.2 Widersprüchliche Ziele

Widersprüchliche Ziele von Future-Internet-Konzepten führen bei Kombination ihrer Implementierung zu vermehrter Arbeit oder zu Fehlern. Nachfolgend wird erläutert welche Komplikationen aufgrund widersprüchlicher Ziele in Ariba/MCPO als Komposition mit H∀Mcast aufgetreten sind.

4.2.1 Problemstellungen durch Verbindungsorientierung und Verbindungslosigkeit

Multicast ist nicht verbindungsorientiert, da eine einzelne Gruppenadresse eine Menge von Listenern beschreibt. Diese Adresse ist virtuell und zu ihr kann keine Verbindung aufgebaut werden. Die Umsetzung von Multicast kann durchaus verbindungsorientiert sein, falls es sich um N Eins-zu-Eins-Verbindungen handelt (z.B. wenn Daten in einem Overlay mittels TCP verschickt werden).
Eine verbindungsorientierte Umsetzung hat gewisse Nachteile. Ein Sender der vielen Empfängern verbindungsorientiert Daten schickt, bekommt von jedem Empfänger Acks als Antwort zurück. Diese Acknowledgements müssen vom Sender verarbeitet werden. Weiterhin wird der Gruppensender (oder Forwarder) mit dem Auf- und Abbau der Verbindungen belastet. Bei erhöhter Teilnehmerzahl lassen sich deswegen verbindungsorientierte Multicast-Architekturen schwerer skalieren als verbindungslose.
Multicast-Anwendungen verwenden aus diesem Grund meistens verbindungslose Datenübertragung. Zudem wird Multicast in den häufigsten Fällen in einem Zusammenhang benutzt, der keine unbedingte Zuverlässigkeit erfordert und deswegen auch keine Verbindungsorientierung

benötigt. Dies ist z. B. der Fall, wenn Video-Streams über das Netz geschickt werden. Wenn eine Nachricht verloren geht, merkt der Empfänger es, je nach Codierung der Bilder, nicht. Falls dieses Bild nach einer längeren Zeit doch noch ankommt, wird dieses nicht mehr benötigt (z.B. bei Livestreams), deswegen macht ein erneutes Senden eines Bildes keinen Sinn. Der Sender würde jedoch beeinträchtigt, wenn er für jedes Bild, von jedem Knoten an den gesandt wird, ein Acknowledgement bearbeitet.

H∀Mcast und Ariba/MCPO haben unterschiedliche Verbindungszustände. Das Konzept von Ariba stellt verbindungsorientierte, zuverlässige Verbindungen zur Verfügung, die über Netzwerktechnologie-Grenzen hinweg und hinter Middleboxes kommunizieren können. MCPO überträgt die Daten mittels der Verbindungen, die von Ariba zur Verfügung gestellt werden. Dabei werden Rückgabewerte, ob Verbindungen noch erhalten sind oder eine Nachricht angekommen ist, nicht berücksichtigt. MCPO ist damit ebenfalls verbindungslos. H∀Mcast ist verbindungslos und versendet die Daten an eine bestimmte Adresse (H∀Mcast-URI). Wie die unterliegenden Module die Nachrichten übertragen, fragt die H∀Mcast-Middleware nicht ab. MCPO benutzt über Ariba verbindungsorientierte Kommunikation. An den Ariba-Knoten wird jeweils eine TCP- bzw. SCTP-Verbindung eingerichtet. Daraus ergibt sich die Fragestellung inwiefern sich der Auf- und Abbau von Verbindungen auf die verschiedenen Multicast-Funktionen auswirkt.

4.2.2 Zuverlässiger und unzuverlässiger Multicast

Ariba/MCPO mit H∀Mcast implementieren einen unzuverlässigen Multicast-Dienst über dem zuverlässigen Overlay Ariba. Traditionell wird Multicast unzuverlässig umgesetzt, da die meisten Anwendungen wie z. B. die Verbreitung von Musik- und Videostreams keine Zuverlässigkeit benötigen und Zuverlässigkeit die Geschwindigkeit dieser Anwendungen beeinträchtigen würde.

Eine zuverlässige Verbindung eines Subsystems zu betreiben und darüber ein System zu implementieren, welches den Zuverlässigkeitsaspekt vernachlässigt führt zu vervielfachtem Aufwand. Einerseits trägt ein überliegendes System immer die Kosten des unterliegenden Systems für die Zuverlässigkeit. Andererseits ist die Implementierung für zuverlässigen Multicast aufwendig, und es ist fraglich, ob dieser Aufwand gerechtfertigt ist, wenn die meisten Multicast-Anwendungen keinen zuverlässigen Multicast benötigen.

Dies führt zu der Fragestellung, wie sinnvoll es ist, MCPO zuverlässig zu gestalten, um die Ziele von Ariba beizubehalten, und der Anwendung zuverlässigen Multicast zu bieten. Dies ist auch abhängig von dem Aufwand, der für zuverlässigen Multicast betrieben werden muss. Weiterhin stellt sich die Frage ob und auf welcher Ebene die Sicherheitsmechanismen für

Zuverlässigkeit einbezogen werden sollten und welche Vor- und Nachteile das bietet. Weiterhin ist auszuwerten wie sich H∀Mcast im Bezug auf Zuverlässigkeit verhält.

4.3 Ergänzen einer Funktionalität als Indiz für Komplexität

Anhand der Stellen, die man in einer System-Architektur ändern muss, um eine Funktionalität hinzuzufügen, lässt sich die Komplexität dieses Systems ablesen (Carpenter, 1996). Dieses Verfahren legt offen welche Abhängigkeiten es gibt, da diese geändert und implementiert werden müssen.

Unter der Annahme, dass eine Ebene als Komponente repräsentiert wird (ein gängiges Designprinzip, welches auch H∀Mcast und Ariba/MCPO nutzen), sieht man direkt an der Beteiligung einer Komponente, dass diese Ebene in den Änderungsprozess inbegriffen ist. Sind es mehrere Ebenen-Komponenten, so erstreckt sich die Funktionalität ebenfalls über mehrere Ebenen.

Für das Zusammenspiel von H∀Mcast und Ariba/MCPO lässt sich diese Art der Komplexität am besten auswerten, indem man vergleicht, was geändert werden muss um jeweils eine spezifische Netzwerktechnologie (in Ariba) bzw. ein Technologie-Modul (H∀Mcast) hinzuzufügen. Es ergeben sich die Fragen:

- Wieviele Schnittstellen müssen eingebunden werden (Abhängigkeiten)?
- Welchen Voraussetzungen müssen die Technolgiekomponenten erfüllen?
- Welche und wie viele Funktionalitäten müssen die Technologiemodule bereitstellen?

4.4 Auswirkungen gestapelter Overlay-Netze auf den Gruppenbeitritt

Ariba und MCPO erzeugen zwei Overlaynetze übereinander. Damit verbunden sind in beiden Komponenten jeweils Initialisierungsphasen. Bei MCPO wird der Knoten anfangs in den Verteilbaum eingegliedert und durchläuft dabei mehrere Phasen bis die ersten Daten gesendet/empfangen werden können. Nach der erfolgreichen *Bootstrap*-Phase muss der Knoten permanent *Heartbeat-Messages* senden, um nicht aus dem Baum ausgegliedert zu werden.
Ähnliches gilt im BaseOverlay von Ariba. Um nicht aus dem Overlay ausgegliedert zu werden, werden regelmäßig die Verbindungen mit Keepalive-Nachrichten offengehalten.
Das Adaptermodul für H∀Mcast nutzt die MCPO-Objekte auf eine Weise, dass bei jedem Gruppenbeitritt dem MCPO-Overlay neu beigetreten werden muss. Dadurch wird der Aufwand

der Intitialisierungsphase von MCPO bei jedem Gruppenbeitritt durchgeführt. Aus diesem Grund ist es interessant auszuwerten, wie lange der Gruppenbeitritt bei der Kombination von H∀Mcast und Ariba/MCPO letztendlich dauert und mit was für Zeiten ein Benutzer dieses Stacks rechnen müsste.

4.5 Problemstellungen durch unterschiedliche Erwartungen an das Zeitverhalten

Weitere Probleme entstehen in kombinierten Future-Internet-Architekturen durch die Erwartungen an das Zeitverhalten. Unterschiedliche Ebenen erwarten Rückmeldungen zu unterschiedlichen Zeitpunkten bevor sie z.B. ein „retry“ versuchen. Verzögert eine Ebene eine Funktion, kann es passieren, dass die Ebene darüber einen Neuversuch startet, obwohl die Ebene darunter vollkommen funktionstüchtig ist. Der Stand der Dinge ist, Netzwerk-APIs möglichst asynchron zu implementieren um Geschwindigkeiten zu erhöhen (durch Multithreading oder mehreren Prozesse) und Fehlerquellen (z.B. durch Deadlocks) zu minimieren. Wenn eine untere Ebene nichts verwirft und alle Anfragen asynchron annimmt (z.B. weil sie vom Konzept her zuverlässig ist) müssen alle diese Versuche ausgeführt werden. Eine höhere Ebene, die in ein „Timeout“ läuft, wird einen zweiten Sendeversuch unternehmen. Wenn die untere Ebene längere Zeitfenster benutzt, führt das zu einer Überlastung der unteren Ebene, da sie immer mehr Daten versenden muss. Der zusätzliche Rechenaufwand der dabei entsteht, verlangsamt die untere Ebene weiter. Damit wird das Problem verschlimmert, dass die untere Ebene nicht die Zeiterwartung der oberen Ebene erfüllt.
Dies kann verhindert werden, indem man ein beispielhaftes „Join Group“, blockierend implementiert. Blockierende Funktionen in Netzwerk-Stacks beeinflussen das Zeitverhalten im schlimmsten Fall bis zur obersten Anwendung. Ein blockierendes „Join“ führt bei einer TV-Anwendung, in der ein „Sender einschalten“ als „Join“ interpretiert wird, dazu, dass der Nutzer gezwungen ist zu warten, bis diese Aktion komplett abgeschlossen ist.
Daraus ergeben sich folgende Fragestellungen:

- Wie verhalten sich die Future-Internet-Architekturen unter großer Sende-Last?
- Wie oft kommt es zu Sende-Neuversuchen in Ariba?
- Inwiefern belasten die Neuversuche oder Maintenance-Methoden die verschiedenen Komponenten H∀Mcast, Ariba und MCPO?

- Welche Schnittstellen sind synchron, bzw. asynchron sind und welche Folgen hat diese Auswahl?
- Wie geht MCPO als Schicht unter H∀Mcast konzeptionell mit einer gesteigerten Anzahl von aufeinanderfolgenden Joins/Leaves um? (Anwendungsfall: durch verschiedene Fernsehsender durchschalten und bevor der Join-Prozess bei einem Sender abgeschlossen ist, wird auf den nächsten umgeschaltet)

4.6 Netzwerk-Stack-Ebenen als Software-Architektur

Die Komplexität einer Netzwerk-Stack-Architektur steigt, sobald die Ebenen sich nicht komplett abschirmen und es Komponenten gibt, die mit mehreren Ebenen arbeiten. Ariba bietet die Möglichkeit auf untere Ebenen zuzugreifen, um anwendungsspezifische Optimierungen vorzunehmen. MCPO ist keine abschirmende Ebene, da ein MCPO-Objekt in der Initialisierung einen Ariba-Node verlangt, der eigentlich als Teil von Ariba, eine Ebene tiefer liegt.

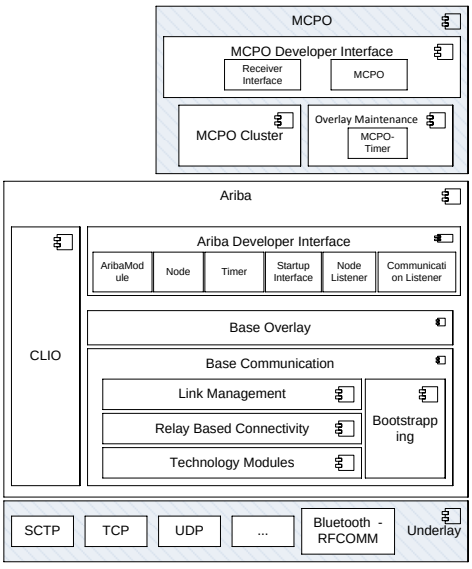

Abbildung 4.1: Kombinierter Netzwerk-Stack von H∀Mcast und Ariba/MCPO

Die Ariba/MCPO-Architektur bietet eine komplexe Schnittstelle für Anwendungen (vgl. Bild 4.1). H∀Mcast stellt eine Schnittstelle für die Module bereit, über die sämtliche Informationen

ausgetauscht werden. Da MCPO die Schnittstelle von Ariba erweitert und nicht komplett ersetzt, besteht hier die Schwierigkeit, die Funktionen so abzubilden, dass die Konzepte korrekt erhalten bleiben.

Es wird untersucht, inwiefern die Bibliotheken H∀Mcast und Ariba/MCPO, sowie der Adapter, die Trennung der Ebenen berücksichtigen, welchen Zweck sie mit Brüchen der Ebenenstruktur verfolgen und zu welchen Komplikationen das führt. Weiterhin wird ausgewertet inwieweit H∀Mcast, Ariba und MCPO dem Anwendungsprogrammierer eine einfach benutzbare Schnittstelle zur Verfügung stellen.

5 Evaluation

In diesem Kapitel werden Problemstellungen aus Kapitel 4 ausgewertet. Es wird gezeigt, inwiefern bei der Abbildung der Funktionen und Adressierungsschemata die Konzepte erhalten bleiben. Anhand der einzelnen Schnittstellen der Komponenten wird die Durchlässigkeit der Ebenen und dessen Folgen diskutiert. Die Komplexität der Komponenten wird anhand einer neuen Funktionalität evaluiert. Zur Erhaltung der Konzepte wird der Aufwand für ein zuverlässiges MCPO-Modul evaluiert, und inwiefern diese Zuverlässigkeit den kompletten kombinierten Stack beeinflussen würde.

Getestet wird die Verzögerung, die ein H∀Mcast-Benutzer mit dem Ariba/MCPO-Modul in Kauf nimmt, und dessen Ergebnisse ausgewertet.

5.1 Adressierungen

Die Hauptaufgabe der Abbildung der H∀Mcast-URI (vgl. Kapitel 4.1) auf die Ariba/MCPO-Gruppenadressierung besteht darin, die Netzwerkarchitekturen aufeinander abzubilden, sodass die einzelnen Future-Internet-Konzepte möglichst wenig eingeschränkt werden. Das Overlay-Konzept von Ariba/MCPO soll erhalten bleiben und in gesamter Breite genutzt werden. Aribas SpoVNet-Overlay kann dabei als eine Art Netzwerk betrachtet werden und dessen Name als Netzwerkinterface (ähnlich zu *eth0*). Die Gruppen im H∀Mcast-Ariba/MCPO-Adapter werden über eine ServiceID (diese Klasse entspricht einem unsigned Integer) innerhalb Aribas identifiziert.

Im Ariba/MCPO-Modul wird eine Gruppe also durch die zwei Identifikatoren beschrieben, SpoVNet-Name und ServiceID zur Identifikation der Anwendung in Ariba. Innerhalb eines MCPO-Objektes gibt es noch ein weiteres „Gruppenkonzept“, welches ankommende Nachrichten an den Endknoten nach beigetretenen „Gruppen“ (ebenfalls durch ServiceIDs dargestellt) filtert. Da MCPO in dieser Form nur einen Verteilbaum pro MCPO-Objekt (also pro Service-Identifikation in Ariba) erzeugt, entspricht das genau dem allgemeinen Portkonzept. Eine Abbildung der H∀Mcast-Gruppen auf diese MCPO-„Gruppen“ würde zu einer verminderten Skalierung führen, da alle Daten durch den kompletten Verteilbaum geflutet werden. Aus

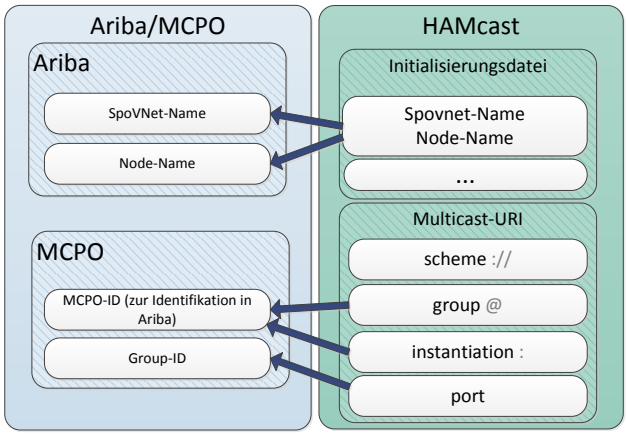

Abbildung 5.1: Abbildung der H∀Mcast-URI auf die Ariba/MCPO-Identifikatoren

diesem Grund wird im Adapter zwischen H∀Mcast und Ariba-MCPO der H∀Mcast-URI-*port* auf die MCPO-Gruppen abgebildet.

Die Abbildung der *group @ instantiation* der H∀Mcast-URI erfolgt mittels Hash auf die ServiceID, die ein Receiver-MCPO-Objekt in Ariba beschreibt. Auf diese Weise wird berücksichtigt, dass die H∀Mcast-URI auch für Funktionalitäten ausgelegt ist, die Ariba/MCPO nicht unterstützt.

In Ariba/MCPO werden Multicast-Nachrichten mit einem Empfänger (die Gruppe) adressiert und an die Gruppenmitglieder weitergeleitet. Die H∀Mcast-URI selbst unterstützt auch Souce-Specific-Multicast (nachfolgend SSM). Diese Funktionalität ist in Ariba/MCPO nicht möglich, da der Sender bei diesem Konzept mitberücksichtigt wird. Im SSM-Konzept leiten die Forwarder bestimmte Nachrichten nur weiter, wenn der Empfänger auch die Gruppendaten von genau diesem Sender abonniert hat. Es wird also bei SSM für jede Nachricht ein Tupel (Sender,Empfänger) benötigt, der in Overlays wie dem MCPO/Ariba-SpoVNet nicht zur Adressierung gehört.
Um diese Logik bei der Servicekomposition dennoch zu erhalten, wird im Adapter für jedes Tupel (S,E) eine Gruppe angelegt (durch das Hashing der *group @ instantiation*). Ein Empfänger muss für jeden Sender einer Gruppe beitreten. Wenn er alle Nachrichten empfangen will, benutzt er nur den *group*-Teil der URI. Wenn man dieses Konzept erhalten will, muss der Sender also einmal ohne SSM in die Gruppe senden, und einmal in einer Gruppe die seine Sende-*Instantiation* berücksichtigt. Wenn SSM in dieser Form konsequent verwendet werden soll, müssen alle Nachrichten doppelt verschickt werden, einmal mit *instantiation* und einmal

ohne.
Dies ist eine Stelle an der die Funktionalität von H∀Mcast nicht optimal auf Ariba/MCPO abgebildet werden kann.

Der Nachteil dieser Abbildung der Adressierungen ist, dass sie mit dem MCPO-Konzept bricht, welches nativ nur ein MCPO-Objekt je Endknoten vorsieht und allein die Filterung von Nachrichten an Endpunkten vornimmt. Im nativen MCPO ist es nicht möglich die ServiceID zu bestimmen, mit der sich MCPO bei dem Ariba-Objekt registriert. Dies wurde nachimplementiert, damit ein Verteilbaum pro Gruppe ermöglicht wird. Dieses Konzept ist dann nur noch unter einer bestimmten ServiceID mit dem ursprünglichen MCPO-Konzept kompatibel. Dafür müsste die H∀Mcast-URI auf die ursprüngliche ServiceID MCPOs abgebildet werden, und die H∀Mcast-Anwendung dürfte danach Gruppen nur noch auf Port-Ebene unterscheidet.

Der Vorteil dieser Abbildung ist, dass die Ariba/MCPO-Gruppen gut genug skalierbar sind und die Daten nicht an alle Multicast-Teilnehmer geflutet werden. Weiterhin bleibt der ID/Locator-Split in Ariba korrekt erhalten und es wird keine Funktionalität von Ariba eingeschränkt.

Weitere Problemstellungen entstehen durch die Abbildung der Adressierungen auf die ServiceIDs. Diese Abbildung steht im Widerspruch zum Ariba-Konzept, in dem eine ServiceID anwendungsspezifisch ist. Im Modul ist die ServiceID gruppenspezifisch, damit das NICE-Multicast-Konzept aufgeht. Es bedeutet aber auch, dass andere Ariba-Knoten (im gleichen SpoVNet), die die Adressierung echt anwendungsspezifisch nutzen, eventuell mit Gruppendaten von Ariba/MCPO belastet werden (z.B. durch Routing dieser Daten), obwohl sie weder an der Gruppe noch an der Anwendung „Multicast“ interessiert sind. Auf diese Weise wird durch die Kombination von H∀Mcast und Ariba/MCPO ein komplettes SpoVNet beeinflusst. Dies ist nur ein Problem, wenn der Multicast-Dienst in einem bestehenden SpoVNet laufen soll, in dem noch andere Anwendungen laufen. In diesem Fall kann es zu Störungen der Anwendungen kommen, wenn viele Multicast-Daten geroutet werden. Weiterhin ist es dann nicht vorhersehbar inwiefern sich ServiceIDs der Anwendungen (die ja auch frei gewählt werden können) mit den ServiceIDs für die Gruppen überschneiden. Durch die Benutzung eines eigenen SpoVNets nur für eine Multicast-Anwendung, können diese Probleme vermieden werden.

5.2 Zuverlässigkeit in MCPO

Ariba stellt zuverlässige Verbindungen für überliegende Anwendungen zur Verfügung. MCPO garantiert keine zuverlässigen Datenverteilung, da Zuverlässigkeit bei Multicast nicht üblich ist. Allerdings stellt MCPO keine einfache Anwendung dar, die Ariba benutzt, sondern eine Anwendungsschnittstelle, die Multicast zur Verfügung stellt. Auf MCPO-Ebene ist nicht klar, ob eine Anwendung zuverlässige Datenübertragung benötigt oder nicht. Wenn MCPO die Zuverlässigkeit Aribas beibehalten will, muss enormer Aufwand betrieben werden, um zuverlässigen Multicast sicherzustellen. Im Fall von zuverlässigen Multicast reicht es nicht die Verbindung zwischen zwei Zwischen-Hops im Verteilbaum zuverlässig zu gestalten. Zusätzlich muss eine Sicherung vom Anfangs- zu Endknoten erfolgen (vgl. Kapitel 3.3), bzw. muss sichergestellt werden, dass der Knoten die Nachrichten im Falle von Nachrichtenverlust aus einer anderen Quelle beziehen kann. Wenn Knoten A die Multicast-Nachrichten an Knoten B sendet, Knoten B diese weiter an Knoten C sendet, kann bei Knoten B eine Nachricht die korrekt von Knoten A empfangen wurde, verschwinden (z.B. durch Überfüllung eines Puffers). Knoten B schickt dann alle Nachrichten außer diese eine weiter, und Knoten C wird niemals feststellen, dass diese eine zusätzliche Nachricht existiert hat.

Diese Sicherungen sind ein Aufwand, den die Anwendung in Kauf nehmen muss, zusätzlich zum Aufwand der zuverlässigen Verbindungen in Ariba. Benötigt eine Anwendung keinen zuverlässigen Multicast, ist dieser Aufwand überflüssig und höher als wenn der Multicast-Dienst nicht zuverlässig implementiert wäre. Wenn eine Anwendung jedoch zuverlässigen Multicast benötigt und MCPO diesen vernachlässigt, muss die Anwendung diesen neu über MCPO implementieren. Die Anwendung hat dann keinen Vorteil von Aribas zuverlässigen Verbindungen, obwohl sie dessen Kosten mitträgt.

Die Diskussion, ob Zuverlässigkeit in MCPO sinnvoll wäre, wägt zwischen zwei Hauptzielen ab.

- Die Beibehaltung unterliegender Konzepte um mehrfachen Aufwand für gleiche Ziele zu minimieren.
- Das Minimieren der Kosten anhand der erwarteten Multicast-Anwendung, die in den seltensten Fällen zuverlässigen Multicast benötigt.

Welchem der beiden Ziele man bei kombinierten Netzwerk-Stacks folgen, sollte ist letztendlich von der Anwendung abhängig, die darauf laufen soll. Da H∀Mcast Zuverlässigkeit nicht maßgeblich unterstützt, ist es in diesem Fall positiv zu bewerten, dass MCPO nicht den zusätzlichen Aufwand für Zuverlässigkeit auf sich nimmt.

5.3 Eine neue Funktionalität in H∀Mcast und Ariba

Das Hinzufügen einer neuen Funktionalität weist auf die Komplexität eines Netzwerk-Stacks hin. In Ariba betrifft es das Hinzufügen einer Netzwerktechnologie und in H∀Mcast das Hinzufügen eines neuen Multicast-Moduls. Die Komponenten einer Future-Internet-Architektur, die in das Integrieren einer neuen Funktionalität mit einbezogen werden, weisen auf die Anzahl der internen Abhängigkeiten hin. Reicht eine Implementierung an einer Stelle im Netzwerk-Stack, so sind die Aufgaben der Komponenten logisch voneinander abgetrennt. In diesem Fall sinkt die Gefahr, dass die Funktionen, die in dieser Schnittstelle implementiert werden, sich gegenseitig beeinflussen. Die Komplexität ist damit geringer. Sind Implementierung von Schnittstellen aus verschiedenen Ebenen nötig, steigt die Gefahr, dass die implementierten Funktionen sich gegenseitig beeinflussen, weil sie voneinander abhängen können.

H∀Mcast und Ariba/MCPO sind beide darauf ausgelegt von mehreren Technologien zu abstrahieren, und diese in einem Gesamtnetz zusammenzufassen. Deswegen bieten beide die Möglichkeit Technologiemodule zu implementieren. Diese Module werden zu der jeweiligen „Modulschicht“ hinzugefügt. Hierbei müssen die Module unter Ariba eine zuverlässige Verbindung bereitstellen und bei H∀Mcast die entsprechenden Multicast-Technologien.

Nachfolgend wird der Aufwand und die Abhängigkeiten verglichen, die beachtet werden müssen, wenn ein neues Netzwerkmodul in Ariba und H∀Mcast hinzugefügt werden soll. Der Einfluss auf die verschiedenen Ebenen wird betrachtet, sowie der Einfluss auf ebenenübergreifende Komponenten.

Eine neues Technologiemodul für H∀Mcast

Wenn ein neues Technologiemodul für H∀Mcast entwickelt werden soll, ist dieses von genau einer Schnittstelle abhängig. Die Komplexität entsteht über das passgenau Abbilden der Funktionen auf die Schnittstelle, die von der neuen Technologie angeboten wird. Die Modulschnittstelle bildet, bis auf eine Ausnahme, genau die H∀Mcast-Anwendungsschnittstelle ab. Dadurch wird die Komplexität der Schnittstelle gering gehalten, da keine weiteren Funktionen möglich sind. Auf diese Weise wird viel Komplexität in das Modul selber verlagert.
Gruppenspezifische Zusatzinformationen können mittels H∀Mcast-URI übergeben werden (z.B. Passwörter für Gruppen mittels *Credentials*). Neben den Funktionen aus der H∀Mcast-Anwendungsschnittstelle stellt die Modulschnittstelle Informationen zur Modul-Initialisierung zur Verfügung, die beliebige Key-Value-Paare beeinhaltet, und die Daten aus einer Initialisierungsdatei bezieht. Benötigt eine Anwendung darüber hinaus Informationen vom Anwender, die nicht die Gruppe betreffen, sondern die Technologie an sich, so müssen diese über einen

anderen Kommunikationsweg zur Verfügung gestellt werden (z.B. Popup zum Eingeben des CAPTCHAs, ähnlich wie der Linux Network-Manager die Passwörter für WLAN-Netze fordert).

Die Modulschnittstelle von H∀Mcast abstrahiert beabsichtigt von diesen zusätzlichen Funktionen. Dies hat genau zwei Gründe:

- Das Konzept von H∀Mcast beinhaltet, dass der Anwender genau von diesen Technologiespezifischen Eigenschaften unabhängig ist, und soll deswegen nur mit den Funktionen arbeiten, die für alle Multicast-Technologien gelten.
- Die Middleware wird früher gestartet als die Anwendung und kann komplett ohne Anwendung initialisiert werden und laufen. Würde die Anwendungsschnittstelle Funktionen für technologiespezifische Eigenschaften beinhalten, dann wäre das Modul, welches diese Informationen benötigt, von einer Anwendung abhängig, die diese Funktionen implementiert. Letzteres widerspricht dem Konzept, dass H∀Mcast die Anwendungen und die Multicast-Technologie entkoppelt.

Damit hat ein Modul in H∀Mcast genau zwei Abhängigkeiten um die Multicast-Funktionalität zu erfüllen. Das Modul-Interface, sowie Daten in der optionalen Initialisierungsdatei. Eine weitere Voraussetzung für das Modul ist, dass alle dynamischen Konfigurationen, die eine Eingabe zur Laufzeit benötigen, über externe Kommunikationsschnittstellen stattfinden.

Ein neues Underlay-Modul für Ariba

Das Gegenstück zu den Technologie-Modulen in H∀Mcast bilden die Transportmodule in Ariba. Diese stellen für die *BaseCommunication* (vgl. Abschnitt 2.3) die Kommunikation auf den Netzwerk-Schnittstellen zur Verfügung. Diese ähnelt der Modulschnittstelle von H∀Mcast. Gängige Unicast-Funktionen müssen dabei erfüllt sein (*send*, *start*, *stop*), und bei Datenempfang wird eine Callback-Funktion aufgerufen (*receiveData*). Durch diese Schnittstelle abstrahiert Ariba von der eigentlichen Implementation der Schnittstelle. Weitere Konfiguration für diese Netztechnologien müssen, wie bei den Zusatzfunktionen bei H∀Mcast, auf anderem Wege sichergestellt werden (z.B. die Konfiguration einer LAN-Schnittstelle für einen PC).

Ein neues Underlaymodul für Ariba ist damit konzeptionell von genau dieser Schnittstelle abhängig. Weiterhin stellt es für Ariba eine zuverlässige Netzwerkkommunikation zur Verfügung.

5.4 Strukturierung und Durchlässigkeit der Ebenen

Der kombinierte Netzwerk-Stack mit H∀Mcast und Ariba/MCPO durchbricht an mehreren Stellen die klassische Ebenenstruktur. Abbildung 5.2 zeigt den gesamten Stack mit seinen Schnittstellen. Eine klare Trennung findet zwischen Adapter-Anwendung und H∀Mcast-Middleware statt. Hier ist genau eine Schnittstelle vorhanden, die von allen unteren Ebenen abstrahiert. Durchbrochen wird die Ebenenstruktur das erste Mal zwischen dem Adapter und Ariba/MCPO. Hier werden Elemente aus beiden Komponenten, Ariba und MCPO verwendet. Eine weitere Unterbrechung in der Ebenenstruktur bildet das CLIO-Modul (Cross Layer Information Overlay) in Ariba. CLIO benutzt Informationen aus allen Ebenen, um seine Funktionen zur Verfügung zu stellen.

5.4.1 Ariba/MCPO-Anwendungsschnittstelle

MCPO erweitert die Schnittstelle von Ariba um Multicast-Funkionen. Das Aribakonzept sieht vor, dass mit verschiedenen Knotenobjekten mehreren SpoVNets beigetreten werden kann. Darüber hinaus können auf einem Knoten, der in einem SpoVNet-Kontext läuft, mehrere Anwendungen laufen, die mittels ServiceID unterschieden werden.

In der Initialisierungsphase ist die Anwendung für die Initialisierung des Ariba-Objektes (*ariba_module*), des Knotens (*Node*) aus der Ariba-Schnittstelle und des MCPO-Objektes aus der MCPO-Schnittstelle, zuständig. Da MCPO laufende Ariba- und Node-Objekte fordert, muss die Anwendung eine bestimmte Reihenfolge der Initialisierungsschritte einhalten.

1. Erstellung des Ariba-Objektes
2. Initialisierung des Ariba-Objektes
3. Starten des Ariba-Objektes
4. Erstellung eines Ariba-Node-Objektes
5. Binden der Anwendung an das Node-Objekt
6. Beitritt des Node-Objektes im SpoVNet
7. Erstellen des MCPO-Objektes
8. MCPO bindet sich an das Node-Objekt

Im klassischen Ebenendesign wäre zu erwarten, dass MCPO Ariba transparent vor der Anwendung verbirgt. Dies ist hier nicht der Fall, da der Anwendungsprogrammierer das Ariba-Objekt, sowie das Node-Objekt initialisieren muss, die beide zur Ariba-Schnittstelle gehören.

Dieses Vorgehen ist jedoch notwendig, da das Ariba-Konzept vorsieht, dass mehrere Anwendungen pro Ariba laufen können. Dies wäre nicht möglich, wenn MCPO Ariba vor der Anwendung verbirgt.

Das Ariba-Konzept sieht weiterhin vor, dass ein Knoten mehreren SpoVNets beitreten kann. Ein Node-Objekt repräsentiert ein SpoVNet in der Ariba-Anwendungsschnittstelle. Dieses Konzept könnte ebenfalls nicht erhalten bleiben, wenn MCPO das Node-Objekt vor dem Anwendungsprogrammierer verbirgt. Aus diesem Grund, muss der Anwendungsprogrammierer die Initialisierung des Ariba-Objektes und des Node-Objektes selbst verwalten. Durch diese beiden Initialisierungen auf Anwendungsebene, kann ein SpoVNet anwendungsabhängig aufgebaut werden.

Um Ariba um die Multicast-Funktionalität zu erweitern, implementiert MCPO das „CommunicationListener“ Interface (siehe Abbildung 5.2) von Ariba, und bietet dem Anwendungsprogrammierer stattdessen sein eigenes ReceiverInterface an. Auf diese Weise übernimmt MCPO die Verantwortung für die Verbindungsfunktionen (onLinkUp, onLinkChanged, onLinkRequest, onMessage...) des Aribasockets zu einzelnen Knoten. Das ReceiverInterface bietet Multicast-spezifische Funktionen, wie z. B. „joinGroup“ und „onMessage(GroupID)". Dies hat den positiven Effekt, dass die einzelnen Verbindungen vor dem Anwendungsprogrammierer verborgen bleiben und, wie im Multicast-Konzept vorgesehen, eine Gruppe adressiert wird. Die Verteilung der Daten erfolgt so transparent.

Der Anwendungsprogrammierer, der nur mit Ariba/MCPO und ohne H∀Mcast programmiert, sieht eine Schnittstelle mit der er zwei Overlaynetze verwalten muss, Aribas SpoVNet und MCPO (ReceiverInterface). Er muss also wissen, wie Ariba und MCPO zusammenhängen, um die Gruppenkommunikation zu realisieren. Beispielhaft gesprochen, sieht er zweimal „Join“, einmal auf dem Node-Objekt (für das SpoVNet) und einmal für die Multicast-Gruppe, die er beitreten will, und muss wissen was diese bewirken und die Reihenfolge in der diese ausgeführt werden müssen. In der Kombination von H∀Mcast und Ariba/MCPO implementiert das Adaptermodul diese Abhängigkeiten.

Das Ende-zu-Ende-Prinzip verlagert Komplexität auf höhere Schichten. Das Aribakonzept sieht vor, dass ein SpoVNet anwendungsabhängig aufgebaut wird. Deswegen muss der Anwendungsprogrammierer das Node-Objekt sowie den NodeListener verwalten, die den Zugang zu einem SpoVNet repräsentieren. Da ein SpoVNet auch in der Kombination von Ariba/MCPO anwendungsabhängig ist, erwartet MCPO die Information, welchem SpoVNet beigetreten

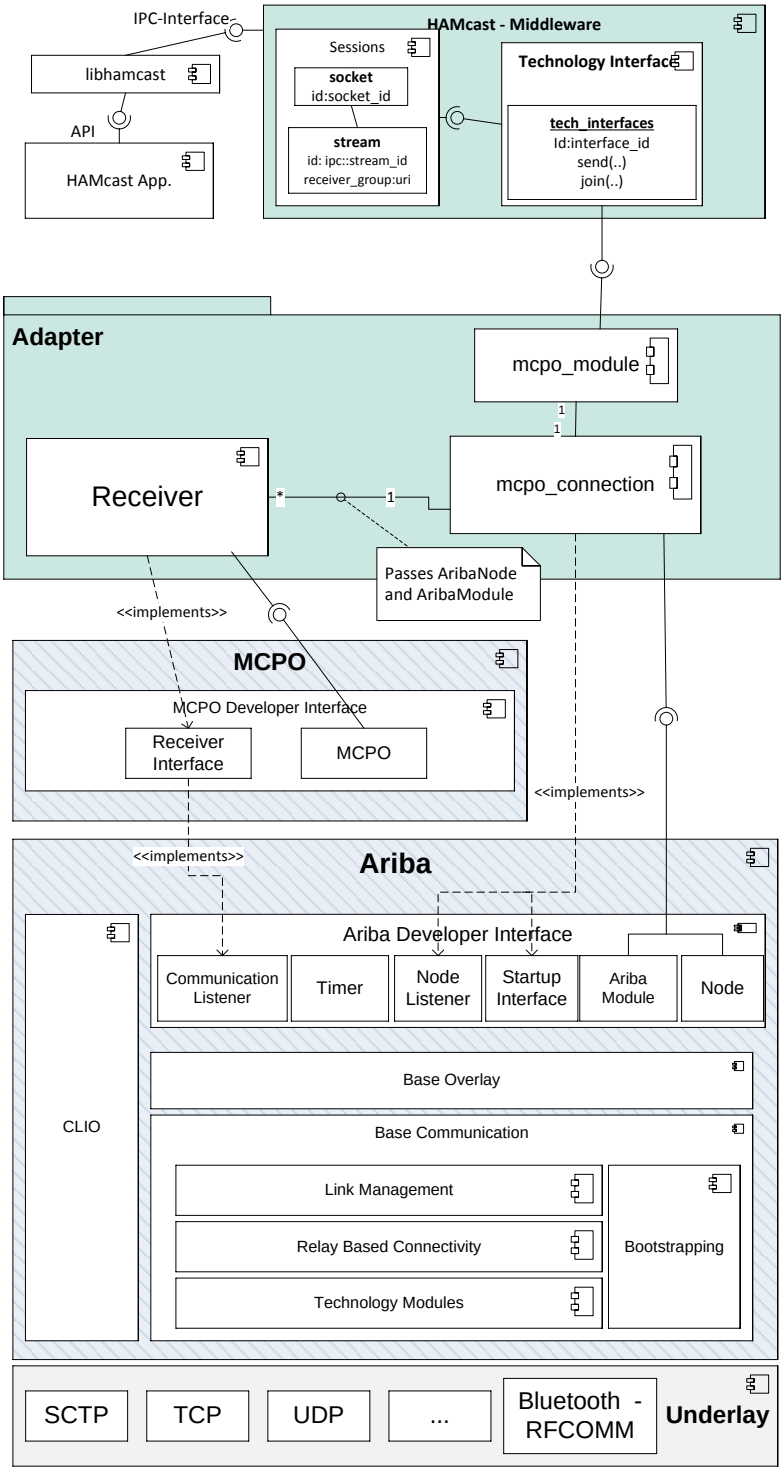

Abbildung 5.2: Kombinierter Netzwerk-Stack von H∀Mcast und Ariba/MCPO

werden soll, ebenfalls von der Anwendung. Die Information über das unterliegende SpoVNet von der überliegenden Anwendung zu erwarten verletzt jedoch das Transparenzprinzip. Dies zwingt den Anwendungsprogrammierer dazu, sich mit dem Konzept von Ariba zu beschäftigen, auch wenn dieses, logisch gesehen, eine Ebene unter MCPO liegt.

Wenn das Transparenzprinzip in MCPO umgesetzt werden soll, müsste MCPO die Information über das von der Anwendung gewünschte SpoVNet in Erfahrung bringen, ohne die Anwendung mit einzubeziehen. Da diese Auswahl des SpoVNets jedoch anwendungsabhängig ist, kann MCPO diese Aufgabe niemals zufriedenstellend erfüllen, was nach dem Ende-zu-Ende-Prinzip dazu führen würde, diese Funktion auf eine höhere Ebene auszulagern.

Bei der Komposition von Overlaynetzen kann es also zu einer Abwägung zwischen dem Ende-zu-Ende-Prinzip und dem Prinzip der Transparenz der Ebenen kommen. Beide Ziele widersprechen sich an dieser Stelle.

5.4.2 Umgang des Adaptermoduls mit der Ariba/MCPO-Schnittstelle

Im Bezug auf das Adaptermodul bedeutet die erweiterte Schnittstelle Aribas, dass der SpoVNet-Namen und der Knotenname vorhanden sein muss, wenn das Node-Objekt und das AribaModule-Objekt initialisiert werden. Das Adaptermodul initialisiert das Node- und AribaModule-Objekt, sobald die Middleware gestartet wird, da das Ariba-SpoVNet die grundlegende Kommunikationsschnittstelle für spätere MCPO-Objekte darstellt. Aus diesem Grund werden der SpoVNet-Namen, sowie die Bootstrapping-Informationen in die H∀Mcast -Middleware-Initialisierungsdatei abgelegt und dem Modul bei der Modulinitialisierung übergeben. An dieser Stelle wird wieder Wissen, welches im Adapter-Modul nicht bereitgestellt werden kann, auf eine höhere Ebene ausgelagert. Dafür sieht der H∀Mcast-Anwendungsprogrammierer einen SpoVNet-Namen, den er angeben muss, da ansonsten das Adaptermodul nicht funktioniert. Dieses Wissen wird für eine viel tiefer liegende Ebene genutzt, und bricht dadurch mit dem Transparenzprinzip zugunsten des Ende-zu-Ende-Prinzips.

Grundsätzlich stellt sich die Frage, mehr Daten aus der H∀Mcast-URI abzuleiten, sodass der Benutzer keine weiteren Konfigurationen für H∀Mcast angeben muss. Der SpoVNet-Name ließe sich aus der H∀Mcast-URI ableiten (z.B. aus der *scheme*). Dagegen sprechen jedoch zwei Dinge:
Das Modul müsste bei jedem Beitritt in eine neue Gruppe einem neuen Overlay-Netz beitreten (zusätzlich zu dem Initialisierungsprozess MCPOs). Dieser SpoVNet-Beitritt kostet Zeit und würde den Gruppenbeitrittsprozesses verlängern.
Der zweite Grund ist, dass dann immer noch die Informationen über mögliche Bootstrapknoten fehlen. Diese müssten entweder weiterhin in der Initialisierungdatei angegeben werden, das

würde für den Anwendungsprogrammierer jedoch nichts vereinfachen. Oder sie müssten sich rein durch Ariba-Funktionen zum Finden von Bootstrap-Knoten ersetzen lassen. Diese Funktionen (Broadcast und Multicast-DNS) sind allerdings nicht besonders erfolgsversprechend, da sie nur lokal funktionieren. Die erfolgsversprechenste Funktion (Bootstrapping), soll deswegen auch genutzt werden.
Auf diese Weise ist die bestmögliche Option, die SpoVNet-Konfiguration (also die Parameter für AribaModule und Node) auf die Anwendungsebene zu verschieben, auch wenn der Anwendungsprogrammierer dann gezwungen wird, sich mit dem SpoVNet-Konzept auseinander zu setzen.

5.4.3 Schnittstellen in H∀Mcast

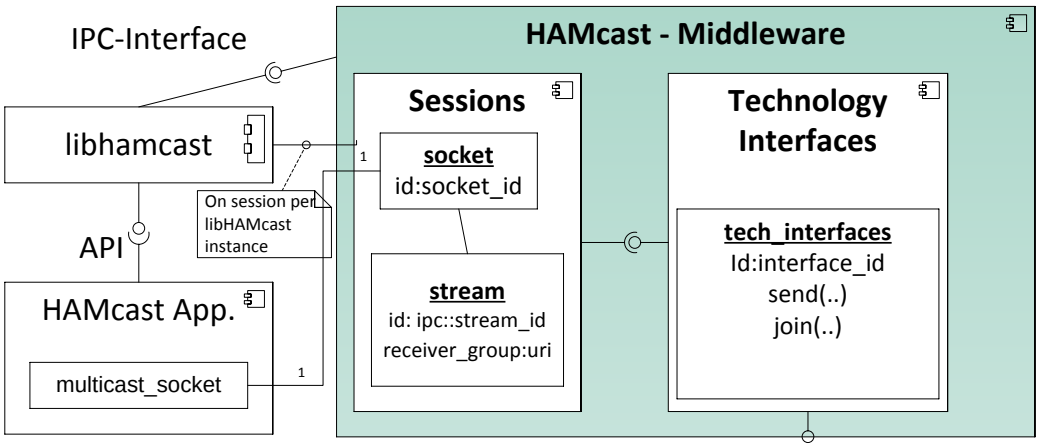

Abbildung 5.3: Komponentendiagramm H∀Mcast

H∀Mcast arbeitet mit zwei Hauptschnittstellen. Die Anwendungsschnittstelle („API“,siehe Abbildung 5.3) verbirgt dabei jegliche Komplexität der verschiedenen Multicast-Technologien hinter einer allgemeinen Schnittstelle für Multicast (Wählisch et al., 2013). Das hat den Vorteil, dass Anwendungsprogrammierer unabhängig von der unterliegenden Technolgie Multicast-Anwendungen entwickeln können. Man hat den Nachteil, dass man von den H∀Mcast-Eigenschaften abhängig ist. Module mit bestimmten Eigenschaften (z.B. Zuverlässigkeit) können nur lokal ausgewählt werden. Durch das automatische Übersetzen der Nachrichten von IMGs kann es passieren, dass Nachrichten über Multicast-Domänen geroutet werden, die nicht zuverlässig sind, und deswegen die Nachrichten nicht einmal bei den Knoten zuverlässig ankommen, die

die gleiche Technologie unterstützen wie der Sender. Die API-Schnittstelle von H∀Mcast ist also auf Kosten der Eigenschaften der unterliegenden Module unkompliziert. Konzeptionell ist H∀Mcast unabhängig von den Modulen darunter und kann deswegen keine Aussagen über die Module, bzw. Optimierungsmöglichkeiten bieten. Interessanterweise ist dies genau das Argument, das oft benutzt wird um im Ende-zu-Ende-Prinzip die Komplexität auf höhere Ebenen zu verschieben (Saltzer et al., 1984). Hierbei liegt jetzt jedoch die Komplexität in den Modulen, die nur indirekt über die Abbildungen der Funktionen und Adressierungen benutzt werden können.

Die H∀Mcast-API kaschiert also Eigenschaften von speziellen Netzwerktechnologien um mit einer unkomplizierten Schnittstelle möglichst viele Empfänger in verschiedenen Multicast-Domänen zu erreichen. Wie im Ende-zu-Ende-Prinzip wird Komplexität für weitere Ziele auf obere Anwendungsebenen verschoben. Konzeptionell integriert H∀Mcast viele verschiedene Module und soll deren Spezialeigenschaften nicht abbilden, damit die unterliegenden Techniken austauschbar bleiben. Das Konzept ist sinnvoll, wenn die unterliegenden Technologien relativ kleine Module sind, deren Gestaltungsspielraum nicht besonders groß ist. Im Fall von größeren unterliegenden Frameworks wie Ariba/MCPO führt das aber im Zweifelsfall zu doppeltem Aufwand der betrieben werden muss (z.B. Zuverlässigkeit in Ariba und der Anwendung über H∀Mcast).

5.4.4 CLIO - Cross Layer Information Overlay

CLIO ist eine Komponente von Ariba, die es dem Anwendungsprogrammierer ermöglicht, das Overlay-Netzwerk, welches er benutzt, zu optimieren (Haage et al., 2009). Mittels CLIO kann die Anwendung verschiedenste Informationen aus dem Underlay in Erfahrung bringen, die sonst transparent für Anwendung sind. Diese Informationen bestehen aus Messungen die CLIO durchführt (z.B. Round Trip Time). CLIO stellt dafür eine möglichst einfache Schnittstelle zur Verfügung, die ca. 50 verschiedene Messungen durchführen kann und die Ergebnisse an die darüber liegende Anwendung zurückgibt. Um den Overhead so gering wie möglich zu halten wird das Overlay nicht ständig überwacht, sondern die Messungen auf Anfrage der Anwendung vorgenommen. Um dennoch von der Netzwerktechnologie zu abstrahieren akzeptiert CLIO das Adressierungsschema der Anwendung und wandelt es intern in die Adressierungen der Underlays um (z.B. IP-Adressen).

Die Messungen CLIOs sind daraufhin ausgelegt möglichst wenig Nebeneffekte auf Ariba zu haben. Aus diesem Grund nimmt CLIO ausschließlich Messungen und keine Konfigurationen vor.

CLIO erstreckt sich über alle Ebenen des Ariba-Netzwerk-Stacks erstreckt, deswegen muss ein Anwendungsprogrammierer, der CLIO benutzen will, Ariba im Detail kennen, damit er die Ergebnisse der CLIO-Messungen richtig interpretieren kann. Dazu kommt, dass er die Möglichkeit benötigt, Ariba zu beeinflussen. Eine Anwendung, die direkt auf Ariba basiert, wie z.B. MCPO hat dazu die Möglichkeit. MCPO bietet die Möglichkeit Entfernungsmessungen im Join-Prozess eines neuen Knotens per CLIO auszuführen und diesen Knoten in ein entsprechendes NICE-Cluster, aufgrund dieser Messungen einzuordnen. Einem Anwendungsprogrammierer, der MCPO benutzt, dem helfen aber die CLIO Daten nur sehr indirekt, da dieser den Verteilbaum von MCPO gesteuert wird, und vom Anwender MCPOs nicht beeinflusst werden kann.

Zusammenfassend erfordert die CLIO-Komponente Detailwissen eines Anwendungsprogrammierers über Ariba, da CLIO sehr viele Ebenen miteinbezieht. Sobald zusätzlich MCPO verwendet wird, kann MCPO durch CLIO seine Datenverteilung optimieren. Der Anwendungsprogrammierer hat in diesem Fall jedoch wenig weiteren Nutzen, da er über MCPO wenig Einfluss üben kann. Allerdings muss er in diesem Fall CLIO auch nicht konfigurieren.

5.5 Test: Auswirkung komplexer Netzwerk-Stacks auf Zeitverhalten bei Gruppenbeitritt

In einem komplexen Netzwerk-Stack wie Ariba/MCPO mit H∀Mcast muss eine Nachricht, bevor sie letztendlich abgeschickt wird, durch eine Menge von Netzwerkabstraktionsschichten gereicht werden. Diese Ebenen fügen den Nachrichten neue Header hinzu, serialisieren sie, und müssen ihre eigenes Netzwerk verwalten. Für jeden Multicast-API-Aufruf existiert damit ein konstanter Aufwand, der immer für diesen Aufruf ausgeführt wird. Dieser Aufwand steigt mit der Komplexität eines Netzwerk-Stacks. Je mehr gegenseitige Abhängigkeiten im Stack vorhanden sind, desto mehr Einstellungen müssen überprüft und gegebenenfalls hergestellt werden. Das bedeutet im Fall von H∀Mcast mit Ariba/MCPO, dass die Verknüpfung des H∀Mcast-Modul mit Ariba/MCPO zu großen Teilen die Geschwindigkeit von bestimmten Aktionen bestimmt. Dabei ist die Geschwindigkeit, die das Ariba/MCPO-H∀Mcast-Modul erreichen kann nach oben hin begrenzt, durch die Geschwindigkeiten mit der Ariba/MCPO die verknüpfte Aktion ausführen kann.

Wenn eine Anwendung, die H∀Mcast benutzt, ein *JoinGroup* aufruft, wird die Nachricht über die H∀Mcast-Middleware an das Modul („*MCPO_Module*“) übergeben (vgl. Abbildung 5.4). Dieses Modul hat bereits ein Ariba-Modul und ein Ariba-Knoten-Objekt gestartet (dem SpoVNet wird bei Initialisierung der Middleware beigetreten), an die später das MCPO-Objekt

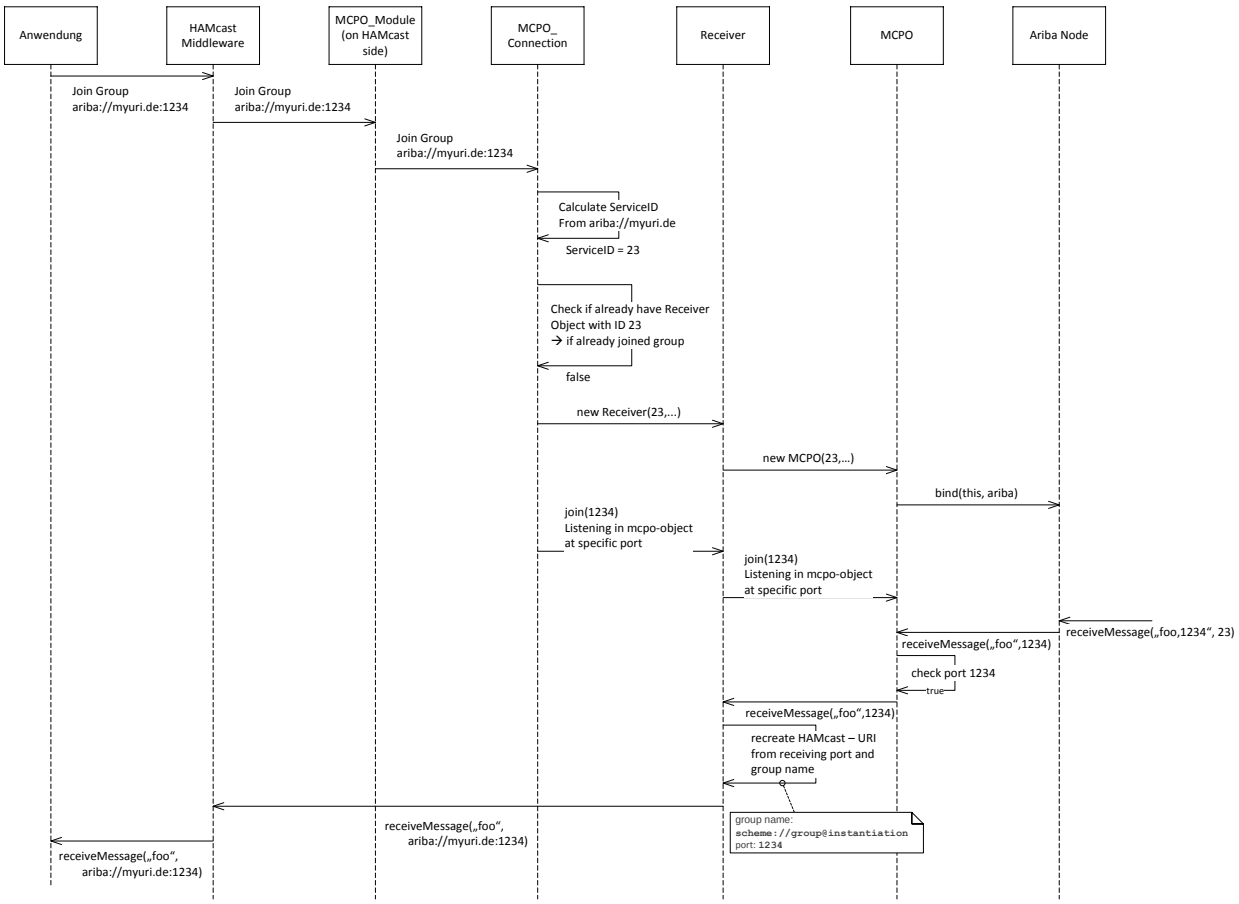

Abbildung 5.4: *JoinGroup* im H∀Mcast- Ariba/MCPO Netzwerk-Stack bis zum Empfang der ersten Nachricht aus der Gruppe.

gebunden wird. Die *MCPO_Connection*-Klasse bildet die H∀Mcast-URI auf eine ServiceID ab, indem es einen Hash-Wert darüber bildet. Mit dieser ServiceID wird die Gruppe im SpoVNet identifiziert. An dieser Stelle bringt die Abbildung der Adressierungen mit sich, dass bei jedem neuen Gruppenbeitritt überprüft werden muss, ob die Gruppe schon vorhanden ist oder es sich „nur“ um einen neuen Port handelt. Im letzteren Fall ist die „Join“ Operation sehr schnell, weil es sich um eine lokale Operation handelt. Der Gruppe wurde schon beigetreten und die Pakete an diesen Port sollen nicht weiter gefiltert werden. Wurde der Gruppe noch nicht beigetreten, wird ein neues Receiver und MCPO-Objekt erstellt, welches die Gruppe im Netzwerk-Stack repräsentiert.

Der konstante Aufwand bei Gruppenbeitritt besteht demnach aus der Initialisierung des Receiver-Objekts und des MCPO-Objektes, sowie der Anbindung dieser an das Ariba-Knoten-Objekt. Sobald das MCPO-Objekt an das Knotenobjekt von Ariba angebunden ist, muss es über das SpoVNet dem Multicast-Verteilbaum beitreten. Das beeinhaltet eine Suche nach Rendezvous Point sowie das Einsortieren des Knotens in das richtige Cluster. Die drei letzteren Aktionen sind mit echter Kommunikation nach außen verbunden und stellen damit den anteilig größten Aufwand bei Gruppenbeitritt.

Das Beitreten des SpoVNets gehört jedoch explizit nicht zu dem Gruppenbeitrittsaufwand, da dem SpoVNet direkt beim Start der Middleware beigetreten wird.

Die Nachrichten, die über das Netzwerk ankommen, ordnet der Ariba-Knoten über die ServiceID dem richtigen MCPO-Objekt zu. Dieses prüft den Port und gibt es weiter an das dazugehörige Receiver-Objekt, das bei Weiterreichung an die H∀Mcast-Receive-Funktion die zugehörige Gruppe übergibt. Dies ist ebenfalls konstanter Aufwand, der bei jedem Empfang einer Nachricht notwendig ist. Dieser Aufwand ist im Vergleich zum Aufwand bei Gruppenbeitritt vernachlässigbar, da es sich um simple, sehr effizient-programmierbare Zuordnungen handelt.

Nachfolgend wird die Verzögerung getestet, die auf Anwendungsebene vom Beitritt einer Gruppe bis zum ersten Datenerhalt vergeht. Um eine realistische Einschätzung zu erhalten ist es notwendig einen Testaufbau zu wählen, der kein Trivialbeispiel ist und dabei handhabbar bleibt. Weiterhin sollen die Knoten, an denen die Beitrittsverzögerung gemessen wird einer Gruppe beitreten, die schon stabil existiert. Auf diese Weise wird verhindert, dass die Messungen durch den initialen Gruppenaufbau verfälscht werden. Der initiale Gruppenaufbau wirkt sich nur auf die ersten Knoten (und damit auf einen sehr geringen Prozentsatz der Knoten) aus und wird deswegen hier nicht betrachtet.

5.5.1 Testaufbau: Join-Latency

Der Testaufbau (in Anlehnung an (Stopp and Hickman, 2004)) besteht aus einer festen Gruppe aus 10 Knoten, die alle derselben Gruppe beigetreten sind (vgl. Abbildung 5.5). Einer dieser dieser Knoten sendet regelmäßig Daten (1000 Nachrichten /s). 11 weitere Testknoten treten jeweils der Gruppe bei und messen dabei die Verzögerung, bis die erste Nachricht für diese Gruppe ankommt.

Im Testaufbau wurden 21 Knoten verwendet, die in verschiedenen Orten Europas (Planetlab) stehen. Auf den Testknoten läuft Fedora 12 und es wurde Ariba v0.8.2 verwendet, da mit der Vorgängerversion keine brauchbaren Testdaten erzeugt werden konnten. Als MCPO-Version wurde MCPO v0.5.1, mit leicht veränderte Schnittstelle zu Ariba verwendet. Die Schnittstelle

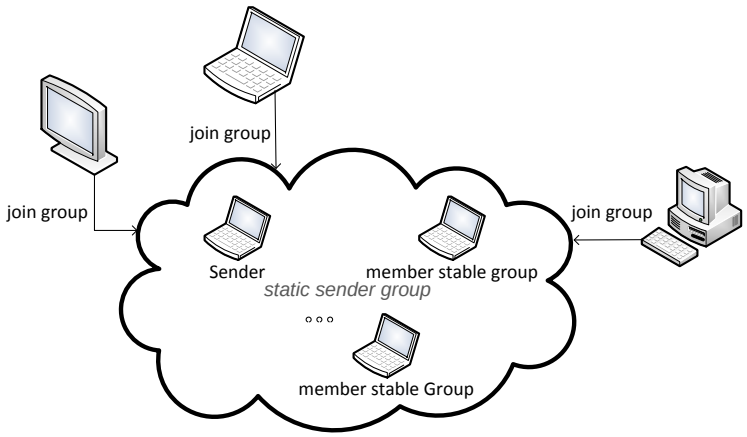

Abbildung 5.5: Testaufbau für ein „Join-Latency-Test“. Statische, fertig initialisierte Gruppen, sowie diverse Knoten die diesen beitreten.

wurde angepasst, da Ariba v0.8.2 im Vergleich zur 0.7.x Version eine leicht veränderte Schnittstelle anbietet, die inkompatibel mit MCPO v0.5.1 ist. Weiterhin wurde angepasst, dass das Adapter-Modul verschiedene MCPO-Objekte auf einem Knoten starten kann (variable ServiceID in Konstruktor). Letzteres war notwendig um die Adressierung von H∀Mcast effizient auf Ariba/MCPO abzubilden (vgl. Abschnitt 5.1)
Die Rollen der Knoten als Sender, Mitglieder der stabilen Anfangsgruppe und Empfänger wurden in jedem Testdurchlauf gewechselt, um Verzögerungen auszugleichen, die sich rein auf den Standort des Knotens zurückführen zu lassen.

Die erzeugten Testdaten, zeigen Problematiken, die im Ariba/MCPO-Netz entstehen können. Wenn der Sender der Gruppe ungünstig liegt, schafft er es nicht, überhaupt Daten an die MCPO-Gruppe zu schicken. Ein solcher Testlauf, enthält dann keine Messdaten über die Beitrittsverzögerung. Es wurde ebenfalls beobachtet, dass nur ein bis zwei Knoten Daten empfangen konnten, die sich zufällig in der Nähe des sendenden Knotens befanden, aber ansonsten zu den restlichen Knoten keinen Kontakt aufbauen konnten. Da auf diese Weise kein stabiles Ariba/MCPO-Netz zustande kam, die eine realistische Einschätzung der Beitrittsverzögerung zulässt, wurden die Testdurchläufe, in denen weniger als acht Empfängerknoten Daten empfangen haben, aus den Daten für die untenstehenden Graphen entfernt. So wurden nur Daten verwendet, die auf funktionierenden Ariba-Netzen mit vergleichbaren MCPO-Verteilbäumen basieren. Dies sind sechs Testdurchläufe deren Messdaten für die folgende Auswertung sowie die Graphen verwendet wurden.

5.5.2 Testergenisse: Join-Latency

Der Graph in Abbildung 5.6, zeigt die Gruppenbeitrittsverzögerung in Sekunden als Cumulative-Distribution-Function. So beträgt die Verzögerung bei 44 % der Knoten unter einer Sekunde, bis die ersten Nachrichten eintreffen. Dies ist ein Intervall, welches den meisten Anwendungen gerecht wird. Auffällig ist die Menge an Knoten die länger als 10 Sekunden benötigen, um in den MCPO-Verteilbaum eingegliedert zu werden. Diese 45% der Knoten haben einen Zeitraum benötigt der für die wenigsten Anwendungen akzeptabel ist. Die Menge der Knoten zeigt, dass es sich um kein Einzelphänomen handelt.

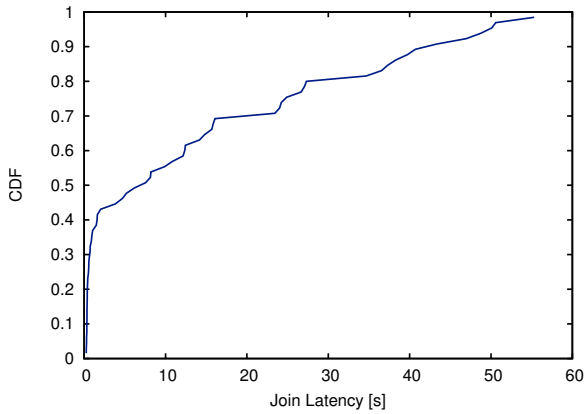

Abbildung 5.6: Die Verzögerung von Gruppenbeitritt auf Anwendungsebene ist abhängig von der langsamsten Zwischenebene.

Gründe für lange Join Latency bei spezifischen Knoten

Die Gründe für die wiederholte hohe Join Latency lassen sich teilweise aus den Logdaten der Tests ablesen. Dafür werden die Knoten die unter einer Sekunde liegen, mit den Knoten verglichen, die deutlich länger gebraucht haben, um klare Unterschiede herauszustellen. Hierbei werden alle logischen Ebenen Ariba/MCPO-H∀Mcast-Stack betrachtet.

Timeout auf verschiedenen Ebenen Aribas zuverlässige Verbindung erfordert das Überprüfen der Verbindungen zu den einzelnen Knoten. Ariba sendet dafür regelmäßig *Heartbeats* über die Verbindungen zu seinen Routing-Nachbarn, um das SpoVNet-Overlay zu verwalten. Die Zeit, bis ein Knoten aus dem SpoVNet ausgegliedert wird, ist dabei statisch festgelegt.

MCPO ist ebenso auf aktive Mitglieder im Verteilbaum angewiesen, sodass sich die Knoten bei ihren Cluster-Mitgliedern melden müssen, damit sie nicht aus dem Cluster entfernt werden. Dieses Verhalten ist in Overlay-Netzen wichtig, um Knoten, die nicht mehr erreichbar sind, aus einer Struktur entfernen zu können.
In der Kombination zweier Overlays ist es dabei notwendig, dass die Zeitfenster der oberen Ebene, die Zeitfenster der unteren Ebene berücksichtigen, da es sonst zu unerwünschten Effekten kommen kann. Sind die Zeitfenster ungünstig gewählt, kann es dazu führen, dass unnötig oft Nachrichten doppelt verschickt werden. Dies ist vor allem der Fall, wenn das Zeitfenster für die obere Schicht nicht länger ist, als das für die unterliegende Schicht. Sind die Zeitfenster gleichlang, wird im Grenzfall ein Timeout ausgelöst, obwohl sich die Nachricht, auf die gewartet wird, eventuell schon im lokalen Stack befindet. Die Knoten, die bereits im Cluster von MCPO eingegliedert sind sowie die Knoten im Ariba-SpoVNet, haben genau das gleiche Zeitfenster (10 Sekunden), bis sie endgültig aus dem Verteilbaum bzw. SpoVNet entfernt werden. Während der Join-Phase in MCPO hat der beitretende Knoten etwas mehr Zeit (15 Sekunden).
Die statischen Timeouts können dazu führen, dass Routing-Knoten an entstanden Engpässen von der Menge der Daten überfordert werden (vgl. dazu auch Jacobson (1988)). Wenn Ariba in einen Timeout läuft, wird versucht diese Daten noch einmal zu verschicken. Wenn ein Zwischenhop die Daten nicht weitergeschickt hat, weil er ausgelastet ist, belastet die erneute Nachricht diesen Knoten noch mehr. Ein statischer Timeout unterscheidet ebenfalls nicht, wie viele Anwendungen auf einem Ariba Knoten laufen. Da sich die Anwendungen den Zugang zum SpoVNet und dessen Bandbreite teilen, und innerhalb des Knotens die Anzahl der Anwendungen bekannt ist, könnte man mit einer Anpassung der Timeouts darauf eingehen und den Timeout der Belastung entsprechend höher setzen.

Auf einen Zusammenhang zwischen der Zeit der einzelnen Ariba-Verbindungen und einer langen Join-Latency deutet ein wiederholt beobachteter Zusammenhang hin. MCPO berechnet ständig den Abstand zu seinen Nachbarknoten indem es die Zeit misst, bis es von ihnen eine Antwortnachricht erhält. Die Distanz wird in Millisekunden berechnet und ist hier als logarithmische Skala dargestellt. Abbildung 5.7 zeigt die berechnete Distanz zweier Beispielknoten zu ihren Routing-Nachbarn. Knoten A hat dabei bis zum Erhalt der ersten Nachricht 55,3 Sekunden warten müssen. Knoten B erhielt die erste Nachricht bereits nach 0,3 Sekunden. Auffällig ist, dass Knoten A eine weit höhere Distanz zu seinen Routing-Nachbarn vorweist als Knoten B. Hier sieht man das ein zentralerer Knoten (B) auch sehr viel schneller Daten erhält. Die Mittlere Distanz zu seinen Routing-Nachbarn beträgt bei Knoten A 163 ms bei Knoten B nur 19 ms.

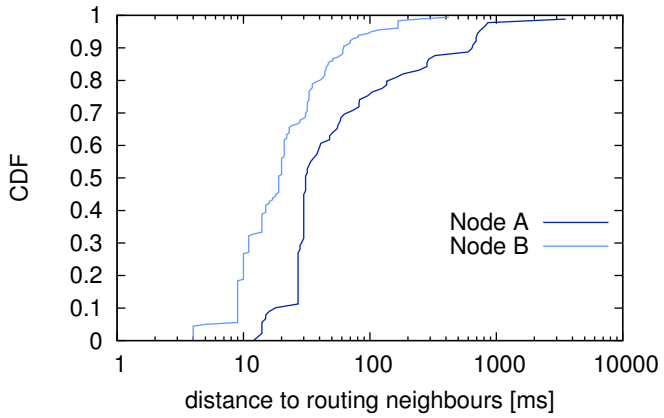

Abbildung 5.7: Beispiel zweier Knoten mit unterschiedlicher Join-Latency, in Abhängigkeit von ihrer Distanz zu den Nachbarknoten.

Aus den bisherigen Testdaten kann dafür jedoch keine Regel abgeleitet werden, da es viele Konfigurationen eines Ariba/MCPO-Netzes gibt, bei denen dieser Zusammenhang nicht zwangsläufig gilt. Dies ist z.B. der Fall, wenn sich zwei bis drei Knoten zwar sehr weit entfernt von allen anderen Knoten befinden, aber untereinander sehr nah beieinander sind, und keiner von ihnen den Sender in dem Test darstellt. In diesem Fall wären die Auswertung der Distanzen zu den Nachbarn geringer, aber die Join-Latency nicht unbedingt. Um einen solchen Zusammenhang zu zeigen wären weitere Tests notwendig, welche die Topologie des unterliegenden Netzes, sowie die Topologie des MCPO-Verteilbaums berücksichtigen.

6 Zusammenfassung

In dieser Arbeit wurde die Komplexität des Zusammenspiels der Future-Internet-Architekturen H∀Mcast und Ariba/MCPO evaluiert. Dabei wurden die Komplexität auf einige Kernaspekte zusammengefasst und anhand dieser Aspekte Problemstellungen im kombinierten H∀Mcast-Ariba/MCPO-Netzwerk-Stack untersucht.

Die Evaluierung zeigt, dass die Abbildung der Funktionen und Konfigurationsparameter (z.B. Adressierungen der Gruppen) einen Kernpunkt für die Effizienz der einzelnen Funktionen darstellt. Funktionsbereiche bringen erhöhten Aufwand für den Anwendungsprogrammierer mit sich, wenn sie sich nicht passgenau abbilden lassen. Durchbrochene Netzwerk-Stack-Ebenen erhöhen die Komplexität der Schnittstelle, da sie die unteren Ebenen beeinflussen, die vor einem Anwendungsprogrammierer eigentlich verborgen bleiben sollten. Das erhöht den Aufwand für den Anwendungsprogrammierer, da er sich mit Konzepten der unteren Ebenen auseinander setzen muss.

Der Test auf Verzögerung bei Gruppenbeitritt zeigt, dass sich widersprüchliche Konzepte nicht optimal aufeinander abbilden ließen, ohne dass dabei hoher Aufwand bei den abgebildeten Funktionen entstand. Neue Problemstellungen ergaben sich durch diese Tests, die die Abstimmung der Timeouts auf verschiedenen Ebenen betreffen.

Das Ende-zu-Ende-Prinzip verschiebt Komplexität. Im Fall des Ariba/MCPO-Stacks findet die meiste Komplexität in den oberen Ebenen statt, während die unteren Ariba-Ebenen, eher die einfacheren Kommunikationsaufgaben übernehmen. Daraus entsteht eine relativ komplexe Anwendungsschnittstelle. In H∀Mcast ist genau das Gegenteil der Fall. Das Ziel ist es, dem Anwendungsprogrammierer Komplexität zu ersparen, indem es eine einfache Multicast-Schnittstelle zu bietet. Daraus entsteht eine relativ einfache Middleware und die Komplexität wird in die Module, also auf eine untere Ebene, verlagert. In der Kombination der beiden Future-Internet-Architekturen findet sich die meiste Komplexität mit den meisten Abhängigkeiten, im Adaptermodul, sowie der Ariba/MCPO-Schnittstelle.

Zusammenfassend ist die Komposition von H∀Mcast und Ariba/MCPO, deren oberen Schnittstellen beide für reine Anwendungen ausgelegt sind, und deren Konzepte sich nur oberflächlich ähneln, mit hoher Komplexität verbunden. Eigenschaften, die von Ebenen zur Verfügung

gestellt werden, sind an der falschen Stelle im Stack implementiert und können vom Anwendungsprogrammierer nicht genutzt werden. Jede Anwendung ist dennoch von der Geschwindigkeit der einzelnen Komponenten abhängig und der Anwendungsprogrammierer von der langsamsten Zwischenebene. Der Anwendungsprogrammierer muss sich weiterhin mit Konzepten der unteren Ebenen beschäftigen, die eigentlich transparent sein sollten. So mögen die einzelnen Systeme ihre Probleme zwar lösen und ihre Aufgabenbereiche gut abdecken, aber die Kombination verschiedener Future-Internet-Technologien zu benutzen, kann auf Seiten des Anwendungsprogrammierers einen hohen Preis haben.

7 Ausblick

Der Test über die Gruppenbeitrittsverzögerung führt zu weiteren Fragestellungen zur Analyse der Overlay-Netze. Dazu gehört ein Gegentest mit anderen H∀Mcast-Modulen um Vergleichswerte zu erhalten. Weiterhin wäre es notwendig den Test mit längeren Timeouts in Ariba und MCPO zu wiederholen, und zu überprüfen, ob eine Verbesserung der Problematik auftritt. Als weiterer Grund für die lange Beitrittsverzögerung sollte die Topologie der Netze, und deren Verwaltungsmechanismen mit einbezogen werden.

Wenn diese Gruppenbeitrittsverzögerung verbessert werden kann, wäre auszuwerten wie die verschiedenen Module in H∀Mcast mit einer gesteigerten Anzahl von hintereinander folgenden „Join"- und „Leave"-Nachrichten umgehen, und wie sich diese auf die lokalen Knoten, sowie die einzelnen Routing-Knoten, in den Overlay-Netzen auswirken.

Im Kapitel über die Problemstellungen wurden weitere Themen angeschnitten, die bisher nicht weiter untersucht wurden. Dazu gehört das Verhalten der Future-Internet-Architekturen unter großer Anwendungsdaten-Sendelast in einem größeren Verteilbaum, und die Auswertung, an welchen Stellen im Stack sich die Unterschiede zwischen synchronen und asynchronen Funktionsaufrufen bemerkbar machen.

Eine Analyse von Komplexität in Textform hat den Nachteil, dass man die Komplexität daraus schlecht vergleichen kann. Ferner wäre also zu überlegen, wie man Komplexität in Netzwerk-Stacks und Future-Internet-Konzepten vergleichbar machen kann, und wie eine Metrik für so einen Stack aussehen könnte. Weiterhin steht die Frage im Raum, wie Komponenten und ihre Schnittstellen gestaltet werden müssen, damit sie untereinander kompatibel sind und man sie beliebig kombinieren kann, um seine gewünschten Eigenschaften für den Stack selbst mit einem Baukastensystem komponieren zu können.

Literaturverzeichnis

Banerjee, S., Bhattacharjee, B., and Kommareddy, C. (2002). Scalable Application Layer Multicast. In *Proc. of SIGCOMM '02*, pages 205–217, New York, NY, USA. ACM Press.

Bless, R., Hübsch, C., Mies, S., and Waldhorst, O. (2008). The underlay abstraction in the spontaneous virtual networks (spovnet) architecture. In *Proc. 4th EuroNGI Conf. on Next Generation Internet Networks (NGI 2008)*, pages 115–122, Krakow, Poland.

Bush, R. and Meyer, D. (2002). Some Internet Architectural Guidelines and Philosophy. RFC 3439, IETF.

Carpenter, B. E. (1996). Architectural Principles of the Internet. RFC 1958, IETF.

Haage, D., Holz, R., Niedermayer, H., and Laskov, P. (2009). A Cross-Layer Information Service for Overlay Network Optimization. In *Kommunikation in Verteilten Systemen 2009 (KiVS 2009)*.

Hübsch, C., Mayer, C. P., Mies, S., Bless, R., Waldhorst, O. P., and Zitterbart, M. (2010). Reconnecting the internet with ariba: self-organizing provisioning of end-to-end connectivity in heterogeneous networks. *SIGCOMM Comput. Commun. Rev.*, 40(1):131–132.

Institut für Telematik, K. I. f. T. (2013). Documentation/mcpo - ariba - overlay-based virtual network substrate. `http://ariba-underlay.org/wiki/Documentation/MCPO`.

Jacobson, V. (1988). Congestion Avoidance and Control. *SIGCOMM Comput. Commun. Rev.*, 18(4):314–329.

Mahy, R., Matthews, P., and Rosenberg, J. (2010). Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766, IETF.

Meiling, S., Charousset, D., Schmidt, T. C., and Wählisch, M. (2012). HAMcast – Evaluierung einer systemzentrierten Middleware-Komponente für einen universellen Multicast-Dienst im Future Internet. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 35(2):83–89.

Saltzer, J. H., Reed, D. P., and Clark, D. D. (1984). End-to-End Arguments in System Design. *ACM Trans. Comput. Syst.*, 2(4):277–288.

Stopp, D. and Hickman, B. (2004). Methodology for IP Multicast Benchmarking. RFC 3918, IETF.

Wählisch, M., Schmidt, T. C., and Venaas, S. (2013). A Common API for Transparent Hybrid Multicast. IRTF Internet Draft – work in progress 08, IRTF.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 27. Mai 2013 Nora Berg