Untersuchungen zur Komplexität komponierter Netzwerkarchitekturen im Future Internet

Nora Berg, Sebastian Meiling, Thomas C. Schmidt, Matthias Wählisch
nora.berg@haw-hamburg.de, {smeiling, t.schmidt, waehlisch}@ieee.org
INET, Dept. Informatik, HAW Hamburg
Berliner Tor 7, 20099 Hamburg

***Zusammenfassung*—Um Probleme im derzeitigen Internet zu vereinfachen, wurden diverse Future-Internet-Architekturen entwickelt. Das Ziel dieser Systeme ist es, die Komplexität, die eine Problemlösung benötigt, unter einer möglichst einfachen Anwendungsschnittstelle zu verbergen. Dafür führen Systeme neue Abstraktionsebenen ein, verarbeiten Informationen mittels Cross-Layer-Komponenten, nutzen neue Namensschemen und werden selbst schnell komplex. Die verschiedenen Future-Internet-Konzepte widmen sich verschiedenen Aufgabenstellungen. Um verschiedene Problembereiche auf einmal abzudecken, besteht ein Ansatz darin, die Architekturen miteinander zu verbinden. Die Netzwerk-Stacks, die dabei entstehen, erhöhen die Komplexität weiterhin. H∀Mcast und Ariba/MCPO sind zwei Future-Internet-Ansätze mit unterschiedlichen Konzepten, die mittels Adapter verbunden wurden, um die Vorteile beider Architekturen zu nutzen. Es wird untersucht, wie sich Komplexität auswirkt, wie sie entsteht und wie man Komplexität in Future-Internet-Anwendungen verringert ohne die Funktionalität unnötig einzuschränken. Die Auswirkungen der Komplexität eines Netzwerk-Stacks auf die Data-Forwarding-Ebene sowie auf die Control-Operationen, werden anhand zweier Testfällen gezeigt.**

I. Einleitung

Im derzeitigen Internet steht ein Anwendungsprogrammierer einer Reihe von Herausforderungen gegenüber, sobald er Software entwickelt, die nicht auf dem typischen Client-Server-Prinzip basiert. Im Laufe der letzten Jahre sind einige Future-Internet-Architekturen entstanden, die mit Peer-to-Peer-Konzepten arbeiten und dem Anwendungsprogrammierer eine einfache Schnittstelle bieten. Um die Komplexität der Problemlösung vor dem Anwender zu verbergen, werden neue Abstraktionen und Konzepte eingeführt, die zu komplexen Systemen unterhalb der Schnittstelle führen. Die Probleme, die diese Architekturen behandeln, liegen z.B. bei der Mobilität der Endgeräte, der Überwindung von Middleboxes und der Bereitstellung heterogener Netzwerkdienste.

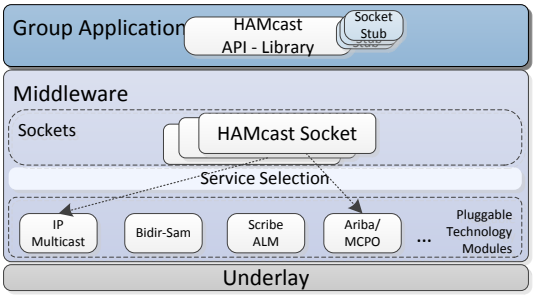
Da einzelne Future-Internet-Architekturen immer nur einen Teilbereich dessen abdecken, was ein Anwendungsprogrammierer gerne benutzen würde, werden hier Möglichkeiten und Grenzen der Komposition verschiedener Ansätze untersucht. Exemplarisch wurden zwei Future-Internet-Frameworks ausgewählt, und als Netzwerk-Stack kombiniert. Hierbei handelt es sich zum einen um den hybriden, adaptiven Multicast-Dienst H∀Mcast [1], der Multicast-Protokolle aus unterschiedlichen Netzwerkschichten, unter einer gemeinsamen API zusammenfasst. Als zweites Framework wurde Ariba/MCPO ([2],[3]) gewählt. MCPO stellt eine Multicast-Schnittstelle (MCPO) zur Verfügung, welche auf einem Overlay für zuverlässige Ende-zu-Ende-Verbindungen (Ariba) aufsetzt. Da Ariba hinter Middleboxes kommunizieren kann, wird durch Ariba/MCPO ein Dienst zur Verfügung gestellt, der Gruppenkommunikation hinter Middleboxes ermöglicht. Mittels der Komposition dieser Future-Internet-Architekturen, wird die Komplexität des komponierten Netzwerk-Stacks evaluiert. Es wird untersucht, wo im Netzwerk-Stack, durch Komplexität Probleme verursacht werden, an welchen Stellen sich Konzepte widersprechen und wie man die gegenseitige Beeinflussung der Systeme überschaubar hält. Weiterhin wird untersucht, welche Faktoren unmittelbar Einfluss auf die Komplexität haben, und damit als Faktoren für die Bewertung der Komplexität eines Systems genutzt werden können. Um die Auswirkungen von Komplexität auf der Control-Plane zu untersuchen, wird getestet, wie lange der Gruppenbeitritt im komponierten Netzwerk-Stack benötigt. Weiterhin wird mit der Messung des Datendurchsatzes die Auswirkung von Komplexität auf der Data-Plane untersucht.

In dieser Ausarbeitung werden in Abschnitt II die Future-Internet-Technologien H∀Mcast und Ariba/MCPO vorgestellt. Anschließend wird in Abschnitt III diskutiert wodurch Komplexität entsteht und welche Aspekte Komplexität beinhaltet. Daraus ergeben sich konkrete Problemstellungen für die Komposition, um die Komplexität möglichst gering zu halten (Abschnitt IV). Die Umsetzung des Adapter-Moduls wird in Abschnitt V beschrieben und anhand der Komplexitätskriterien bewertet. Die Komposition wird in Abschnitt VI getestet und dessen Ergebnisse evaluiert.

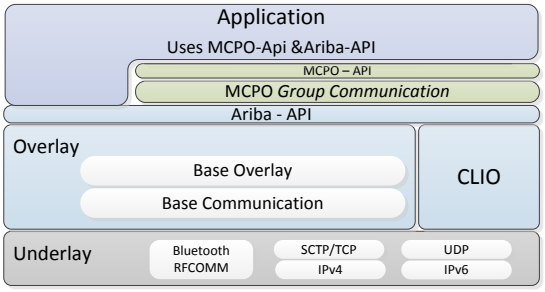
II. Future Internet - Technologien

A. H∀Mcast

Im gegenwärtigen Internet wird Multicast nicht einheitlich unterstützt. Es gibt Protokolle und Anwendungen auf allen 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 die Anbindung an den Kommunikationskanal direkt mit zu implementieren und im laufenden Betrieb zu warten. Aufgrund der inhomogenen Multicast-Protokoll-Verteilung im Internet, ist nicht dennoch nicht sichergestellt, dass alle Endknoten erreicht werden können. H∀Mcast ist ein hybrider Multicast-Dienst [1], der das



(a) H∀Mcast- Netzwerk-Stack



(b) Ariba/MCPO - Architektur (nach ariba-underlay.org)

Abbildung 1. Aufbau der Future-Internet-Frameworks H∀Mcast und Ariba/MCPO

Ziel verfolgt, den Zugriff auf Multicast, durch eine allgemeine Schnittstelle [4] an möglichst vielen Endknoten verfügbar zu machen. Das Grundkonzept ist, eine allgemeine Schnittstelle zur Verfügung zu stellen, welche die Aufrufe für die einzelnen Multicast-Protokolle übersetzt. Weiterhin verfolgt H∀Mcast das Late-Binding-Konzept, sodass die Multicast-Technologie erst zur Laufzeit gewählt wird und nicht schon während der Implementierung.

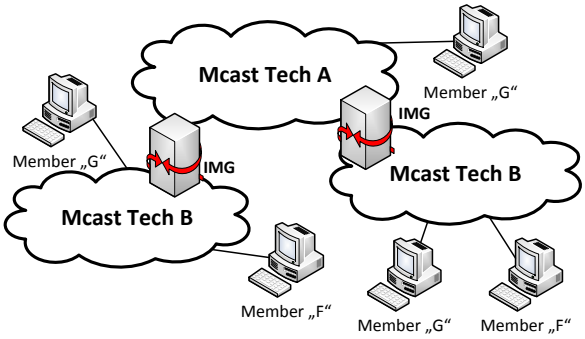


Abbildung 2. H∀Mcast- Netzarchitektur

Damit Multicast-Kommunikation in verschiedene Technologien möglich ist, muss es eine einheitliche Adressierung der Gruppen geben. Zwei Knoten aus verschiedenen Technologien, die verschiedene Adressierungen benutzen und dieselbe Gruppe abonnieren, sollen auch die gleichen Daten erhalten. Es genügt dabei nicht, nur die allgemeinen Funktionen wie „join“ und „leave“ auf die Funktionen der einzelnen Multicast-Technologien abzubilden. Weiterhin muss ein allgemeines Namensschema unterstützt werden, welches auf die speziellen Adressierungen der unterliegenden Multicast-Technologien abgebildet wird.

H∀Mcast benutzt ein solches Namensschema für Anwendungen und realisiert damit einen ID/Locator-Split. So werden Anwendungen unabhängig von der unterliegenden Technologie. Um letztendlich Daten von einer Multicast-Domäne in eine andere zu transferieren, gibt es ausgewählte H∀Mcast-Knoten (Inter Domain Multicast Gateways, IMGs), die verschiedene Multicast-Technologien unterstützen und die Daten zwischen diesen vermitteln (siehe Abbildung 2). Die Daten können in allen Domänen verbreitet werden, welche die gleichen Protokolle unterstützen wie H∀Mcast.

Die Implementierung von H∀Mcast besteht aus drei Hauptkomponenten. Der Anwendungsprogrammierer benutzt die *API-Library*, welche mit der H∀Mcast-*Middleware* kommuniziert. Die Middleware läuft auf einem Knoten genau einmal, sodass alle Programme dieselbe Middleware benutzen. Die Middleware lädt *Module*, die in Form von C/C++-Bibliotheken vorliegen. Jedes dieser Module repräsentiert eine Multicast-Technologie (vgl. Abbildung 1(a)), und diese können geändert, entfernt oder hinzugefügt werden, ohne dass die Anwendung davon betroffen ist.

B. Ariba/MCPO

Ariba [2] ist ein Future-Internet-Ansatz, der Anwendungen eine zuverlässige Ende-zu-Ende-Verbindung zwischen Netzteilnehmern zur Verfügung stellt. Die von Ariba bereitgestellte Kommunikation funktioniert auch in heterogenen Netzen, hinter NATs und Firewalls, sowie über Protokoll-Grenzen hinweg. Dabei verbirgt Ariba transparent die Heterogenität der Netze vor der Anwendung und bildet so eine Fassade im Netzwerk-Stack oberhalb der Transportschicht.

Um die Verbindungen bereitzustellen, baut Ariba ein Overlay-Netz (SpoVNet [5]) 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. Optional zu Ariba kann CLIO (Cross Layer Information Overlay) verwendet werden. CLIO stellt diverse Informationen aus verschiedenen Ebenen des Netzwerk-Stacks bereit (z.B. Performance-Messungen).

Der Multicast/Multipeer Overlay Service (MCPO) [3] stellt auf Grundlage des Ariba-Overlays Multicast-Funktionalität zu Verfügung. Dafür benutzt es die Verbindungen von Ariba um einen Multicast-Verteilbaum ähnlich zu „NICE“ [6] aufzubauen. Die Nachrichten werden im Verteilbaum geflutet und an den Endpunkten durch ein Portkonzept gefiltert.

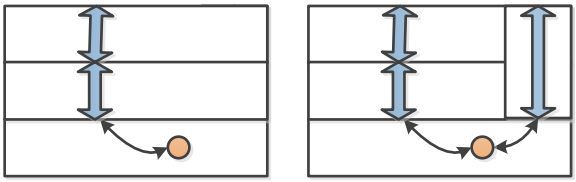
III. Komplexität

Komplexität, wie sie hier untersucht werden soll, bezieht sich auf Netzwerk-Stack-Architekturen und -Konzepte. Sie ist

abzugrenzen von der allgemeinen Komplexitätsanalyse, die sich mit dem Verhalten einzelner Algorithmen beschäftigt. Nicht betrachtet werden implementierungsspezifische Optimierungen, die sich durch Codeänderungen erzielen lassen. Die Komplexität von Future-Internet-Architekturen beinhaltet den allgemeinen Aufbau der spezifischen Frameworks sowie die Komplexität der Vermittlungsmechanismen, die sie verwenden. Bei kombinierten Future-Internet-Architekturen beinhaltet sie zusätzlich die Fragestellung, auf welche Weise die Architekturen verknüpft sind.

A. Das Umgehen der strikten Trennung von Netzwerk-Schichten 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 Cross-Layer-Komponenten, welche die Abstraktionsebenen des Schichtenmodells nicht einhalten.



(a) (b)

Abbildung 3. Informationsfluss im klassischen Netzwerk-Stack (a) und im Netzwerk-Stack mit Cross-Layer-Elementen (b)

In Abbildung 3 sieht man den Informationsfluss zwischen den Ebenen. Abbildung 3(a) zeigt den klassischen Netzwerk-Stack, dessen Ebenen nur mit den direkt angrenzenden Nachbarn kommunizieren. Nur direkt überliegende Schichten haben die Möglichkeit eine Information auf unterer Ebene abzurufen und zu schreiben. Abbildung 3(b) zeigt den Zugriff einer Cross-Layer-Komponente auf eine untere Ebene. Diese Ebene hat nun ebenfalls den Zugriff auf eine bestimmte Information. Die Ebene links davon, ist ebenfalls von dieser Information abhängig. Wird beispielsweise der Wert dieser Information der untersten Ebene von der Cross-Layer-Komponente geändert, müssen eventuell auch Informationen auf der linken Ebene angepasst werden. Es ist aber nicht garantiert, dass die linke Ebene von er Änderung des Wertes Notiz nimmt. So entstehen zusätzliche Abhängigkeiten die berücksichtigt werden müssen, und bei Nicht-Berücksichtigung zu Fehlern führen.

Ein zusätzlicher Aspekt betrifft Cross-Layer-Informationen auf Anwendungsebene. Diese verlangen von einem Anwendungsprogrammierer enormes Wissen, wie sich seine Funktionsaufrufe innerhalb der genutzten Future-Internet-Anwendung auf die anderen Ebenen/Komponenten auswirken (z.B. die linken Ebenen in Abbildung 3(b)). Ansonsten läuft er Gefahr, ungewollt diese anderen Ebenen/Komponenten des Future-Internet-Frameworks zu beeinflussen. Wenn die Cross-Layer-Funktionen die Informationen der unteren Ebene nur abrufen, aber nicht schreiben, steigt zwar die Komplexität der oberen Komponente und der Schnittstelle, die parallel liegenden Ebenen werden jedoch nicht direkt beeinflusst.

Die Komplexität von Netzwerkarchitekturen ist durch die klare Trennung und Abschirmung (Information Hiding) der Ebenen begrenzt. Um in einer Future-Internet-Architektur die gleiche Funktionalität ohne Cross-Layer-Informationsfluss zur Verfügung zu stellen, ist im Zweifelsfall nicht möglich oder führt zu Mehraufwand.

B. Die Komplexität wächst mit der Anzahl der verschiedenen Funktionalitäten

Ein Hinweis auf ein komplexes System ist die Breite seines Zuständigkeitsbereiches (vgl. [7]). Je weiter und detaillierter sich der Aufgabenbereich auffächert, der mit einer Future-Internet-Anwendung bearbeitet werden kann, desto mehr Möglichkeiten gibt es, das Verhalten der Ebene zu beeinflussen. Die Anzahl der verschiedenen Funktionalitäten, bezieht sich nicht auf die Anzahl der einzelnen Funktionen, sondern auf die Bereiche, die sie abdecken. Beispielsweise ist eine Komponente, die einzig Multicast unterstützt („join“, „leave“, „send“, etc.) einfacher, als eine Schnittstelle, die zusätzlich Funktionen über die Beschaffenheit des Verteilbaums bereitstellt (z.B. „getAllOverlayNodes"). Die steigende Komplexität betrifft hierbei nicht nur die Anwendungsschnittstelle, die diese Funktionen benutzen kann, sondern vor allem, wie diese Informationen beschafft werden und welche Randbedingungen sie haben. In diesem Beispiel wird z. B. die Frage aufgeworfen, wie lange der Status gilt, dass die Menge der zurückgegebenen Knoten wirklich alle Knoten enthält.

Die Anzahl der Konfigurationsparameter für die Funktionalitäten kann als Indiz über die Komplexität eines Systems angesehen werden. Je mehr Daten z.B. bei Systemstart vorliegen müssen, desto mehr Abhängigkeiten gibt es von diesen Konfigurationsdaten und desto komplexer wird auch die Schnittstelle. Bei komplexen Netzwerk-Stacks ist es notwendig, mit so wenig Konfigurationsdaten wie möglich auszukommen (vgl. [8]). Ausgangspunkt für die Anzahl der Konfigurationsdaten 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 Konfigurationsdaten, die im Vorfeld bekannt sein müssen. Wie diese Parameter zur Verfügung gestellt werden, ist letztendlich implementationsabhängig und einige können auch während der Laufzeit berechnet werden. Weitere Komplexität entsteht mit der Anzahl der Konfigurationsparameter, die in der Abbildung zwischen mehreren Future-Internet-Architekturen, weder belegt noch berechnet werden können. In diesem Fall geschieht die Belegung entweder mittels Cross-Layer-Informationen, oder durch Daten, die außerhalb des lokalen Stacks zur Verfügung stehen (z.B. Multicast-DNS Namensauflösung). Beide Varianten steigern die Komplexität.

Ariba/MCPO benötigt für Ariba die Informationen, welchem SpoVNet beigetreten werden soll, welcher Endpunkt benutzt werden soll und einen die Adresse eines Rendezvous-Punkts. MCPO benutzt zusätzlich eine Service-ID zur Identifikation in Ariba. H∀Mcast benötigt zum laufenden System die Information, welche Technologiemodule geladen werden sollen und wo sich diese befinden. Weitere Konfigurationsdaten

sind modulabhängig und werden dem Modul zur Initialisierungszeit übergeben.
In der Kompositionen von Future-Internet-Architekturen besteht eine Herausforderung darin, möglichst viele Parameter aufeinander abzubilden, um Funktionalität zu verbinden und 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 korrekt erhalten bleibt. Im Fall des Adaptermoduls zwischen H∀Mcast und Ariba-MCPO bedeutet das, möglichst viele benötigte Parameter für Ariba/MCPO aus den Daten, die H∀Mcast zur Verfügung stellt, automatisch zu generieren.
Ein weiterer funktionaler Komplexitätsaspekt betrifft das Hinzufügen von Funktionen zu einem System [8]. 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 von dem Änderungsprozess betroffen sind und dementsprechend angepasst werden müssen.

C. Das Ende-zu-Ende-Prinzip verlagert 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 sichergestellt werden. 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 anwendungsunabhängig gehalten, da ohne das Wissen über die Anwendung bestimmte Funktionen nicht passgenau zur Verfügung gestellt werden können [9].

Die Ebenen in einem 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 [9], 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, ohne überliegende Ebenen ungewollt durch zusätzlichen Aufwand zu beeinträchtigen. Diese Gefahr des zusätzlichen Aufwands, zeigt sich bei Kompositionen von Netzwerk-Stacks. 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 aber 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 verlagert. Dies geschieht indem eine Future-Internet-Architektur ein bestimmtes Ziel verfolgt (z.B. vereinfachte Ende-zu-Ende-Verbindungen hinter Middleboxes), und für diesen Zweck eine neue Abstraktionsebene einführt (z.B. ein zusätzliches Overlay-Netz). Diese neue Anwendungsschnittstelle muss verwaltet werden (z.B. der Zutritt zu dem Overlay) und erzeugt damit neue Komplexität auf Anwendungsebene. Werden Future-Internet-Architekturen kombiniert, müssen alle diese zusätzlichen Verwaltungsaufgaben erfüllt werden.

IV. Problemstellungen

Die Herausforderung bei der Implementation eines Adapters zwischen zwei Future-Internet-Architekturen besteht darin, die Anforderungen der einzelnen Architekturen so zu erfüllen, dass die Konzepte erhalten bleiben und das Gesamtobjekt möglichst performant funktioniert. Die Anforderungen der einzelnen Architekuren sind dabei jedoch weit gefächert, beeinflussen sich gegenseitig oder widerspechen sich teilweise. Diese Zusammenhänge werden exemplarisch am Adapter zwischen H∀Mcast und Ariba/MCPO illustriert.

A. Gruppenbezeichnung

Die Future-Internet-Architekturen H∀Mcast und Ariba/MCPO benutzen verschiedene Arten der Gruppenbezeichnung. Diese Bezeichnungen müssen aufeinander abgebildet werden, da das Konzept von H∀Mcast ein einheitliches Namensschema vorsieht. Alle Module, die H∀Mcast integriert, bilden dieses Schema auf ihre spezifische Gruppenbezeichner ab. Das ermöglicht, dass Nachrichten aus einer Multicast-Technologie auch in Netze mit anderen Gruppenbezeichnungen übertragen werden können.

Die Bezeichnung der Gruppen in H∀Mcast bestimmt sich durch die H∀Mcast-URI in 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/Anwendungsspezifische Informationen (z.B. für Authentifizierung) zur Verfügung.

In Ariba/MCPO existieren mehrere Ebenen von Bezeichnern, sodass jede Gruppe von mehreren Identifikatoren beschrieben wird.

- Der `SpoVnet-Name` beschreibt das Peer-to-Peer-Overlay als unterste logische Ebene. Auf diesem Netzwerk soll MCPO laufen, und der SpoVNet-Name entspricht dann einem Netzwerkinterface ähnlich zu z.B. *eth0*.
- Der `Knotenname` (Nodename) ist frei wählbar und identifiziert einen bestimmten Knoten innerhalb des SpoVNets. Mithilfe dieser Identifikatoren sind alle Knoten innerhalb eines Netzwerkes erreichbar.
- Eine `Service-ID` (unsigned Integer) beschreibt eine Anwendung auf einem Ariba-Knoten (Portkonzept). MCPO ist eine solche Anwendung und benötigt deswegen eine Service-ID für Ariba
- Innerhalb MCPOs werden Multicast-Gruppen ebenfalls durch eine `Service-ID` beschrieben

Bei der Abbildung der Bezeichner ist wichtig, dass die Konzepte beider Future-Internet-Architekturen erhalten bleiben und ein hoher Datendurchsatz erzielt werden kann. Die Vor- und Nachteile möglicher und vorhandener Abbildungen werden evaluiert und abgewogen.

B. Widersprüchliche Ziele

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

1) Verbindungsorientierung versus Verbindungslosigkeit: Multicast ist nicht verbindungsorientiert, da ein einzelner Gruppenbezeichner eine Menge von Empfängern beschreibt. Bei der Datenübertragung wird eine Nachricht an diesen virtuellen Bezeichner geschickt, zu der keine Ende-zu-Ende-Verbindung aufgebaut werden kann. Ein Sender hat meistens keine Information darüber, welche Knoten sich in der Gruppe befindet, in die er sendet. Das Netzwerk, auf dem Multicast umgesetzt ist, kann dabei durchaus verbindungsorientiert Daten verschicken (z.B. Daten in einem Overlay, die mittels TCP verschickt werden). Es handelt sich dann aber um die unterliegende logische Ebene, deren Daten jeweils zwischen zwei Hops verbindungsorientiert verschickt werden. Dies ist für die darüberliegende Multicast-Ebene keine verbindungsorientierte Ende-zu-Ende-Verbindung. Eine verbindungsorientierte untere Ebene hat gewisse Nachteile. Die Gruppenmitglieder werden bei jeder Datenübertragung mit dem Auf- und Abbau von Verbindungen belastet. Dies beeinträchtigt vor allem Sender und Forwarder, die mehrere Verbindungen zu den jeweiligen Empfängern herstellen und erhalten müssen. 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 jeden Empfänger Verbindungsstati verwalten muss.

H∀Mcast und Ariba/MCPO haben unterschiedliche Verbindungszustände. Das Konzept von Ariba stellt verbindungsorientierte, zuverlässige Verbindungen zur Verfügung, die über heterogene Netzwerke 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 einen bestimmten Bezeichner (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, wie teuer der zusätzliche Overhead durch die Verbindungen Aribas unter H∀Mcast und MCPO ist, und wie hoch der maximal erreichbare Datendurchsatz werden kann.

2) Zuverlässiger versus 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.
Eine zuverlässige Verbindung auf einer unteren Ebene zu betreiben und darüber eine Ebene zu implementieren, welche den Zuverlässigkeitsaspekt vernachlässigt, führt zu vervielfachtem Aufwand. Einerseits trägt ein überliegendes System (in diesem Fall MCPO) 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, solange die meisten Multicast-Anwendungen keinen zuverlässigen Multicast benötigen.
Implementiert man zuverlässigen Multicast, um den Zuverlässigkeitsaspekt des Ariba-Konzeptes zu erhalten, wird der Aufwand durch Überprüfung von Sendungen und Neusendung der Daten vervielfacht. Es reicht in diesem Fall nicht, alle Verbindungen zwischen den einzelnen Knoten entlang eines Pfades zu sichern. Zusätzlich muss eine Sicherung vom Anfangs- zu Endknoten erfolgen (vgl. Absatz III-C), bzw. muss sichergestellt werden, dass der Knoten die Nachrichten

im Falle von Nachrichtenverlust aus einer anderen Quelle beziehen kann.
Diese Sicherungen wären ein Aufwand, den die Anwendung in Kauf nehmen müsste, 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.

Dies führt zu der Fragestellung wie sinnvoll es ist, MCPO zuverlässig zu gestalten, um die Ziele von Ariba beizubehalten. Weiterhin stellt sich die Frage ob und auf welcher Ebene diese Sicherheitsmechanismen einbezogen werden sollten und welche Vor- und Nachteile das bietet.

C. Problemstellungen durch unterschiedliche Erwartungen an das Zeitverhalten

Weitere Komplikationen bei kombinierten Future-Internet-Architekturen bilden 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 sein, 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 mehrere Prozesse) und Fehlerquellen (z.B. durch Deadlocks) zu minimieren. Wenn eine untere Ebene nichts verwirft (z.B. weil sie vom Konzept her zuverlässig ist) und alle Anfragen asynchron annimmt, müssen alle diese Versuche ausgeführt werden. Wenn die obere Ebene einen, im Vergleich kürzeren, Timeout mit anschließendem Neuversuch durchführt, führt das zu einer Überlastung der unteren Ebene und verschlimmert das ursprüngliche Problem.
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. Eine andere Möglichkeit wäre, die Zeitfenster für Timeouts zu erhöhen (ähnlich zu [10]), sobald auffällig viele Verbindungen/Aktionen von einem Timeout unterbrochen werden, und dieses Zeitfenster langsam zu verringern sobald die Verbindungen alle stabil sind.
Das führt zu den Fragestellungen,

- wie sich die Future-Internet-Architekturen unter großer Sende-Last verhalten,
- wie die Zeitfenster in der Overlayverwaltung von Ariba und MCPO umgesetzt sind und wie oft es zu Timeouts und Sende-Neuversuchen kommt,
- welche Schnittstellen synchron, bzw asynchron sind und welche Folgen dies mit sich bringt
- und wie MCPO sowie das Adaptermodul, konzeptionell mit einer gesteigerten Anzahl von direkten Joins/Leaves umgeht (Anwendungsfall: durch verschiedene Fernsehsender durchschalten und bevor der Join-Prozess bei einem Sender abgeschlossen ist, wird auf den nächsten umgeschaltet).

D. Netzwerk-Schichten als Software-Architektur

Die Komplexität steigt mit durchbrochenen Netzwerkschichten an (vgl. III-A). 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. Dieser liegt eigentlich, als Teil von Ariba, eine Ebene tiefer liegt.

Der Aufbau des Moduls über der Ariba/MCPO-Architektur

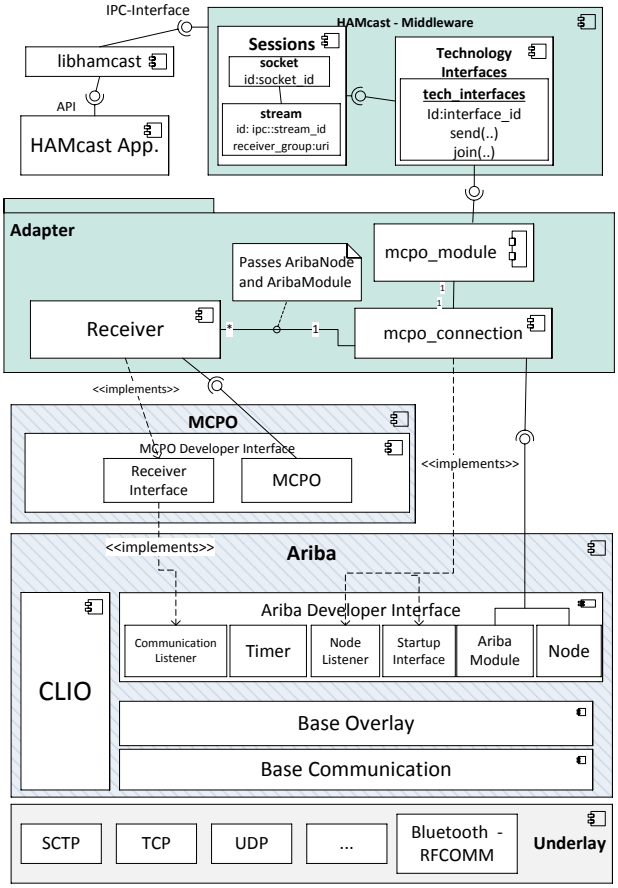


Abbildung 4. Kombinierter Netzwerk-Stack von H∀Mcast und Ariba/MCPO

ergibt ein komplexes Bild an Abhängigkeiten (vgl. Bild 4). H∀Mcast stellt genau 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 ist interessant, inwieweit H∀Mcast, Ariba und MCPO dem Benutzer eine einfach benutzbare Schnittstelle zur Verfügung stellen. Es wird ausgewertet, inwiefern das Adapter-Modul zwischen H∀Mcast und Ariba/MCPO die Konzepte der Frameworks berücksichtigt, und an welchen Stellen die Konzepte nicht weitergeführt werden.

Auswirkungen gestapelter Overlay-Netze auf den Gruppenbeitritt: Ariba und MCPO bauen unabhängig zwei Overlaynetze übereinander auf. Damit verbunden sind in beiden Komponenten jeweils Initialisierungsphasen. Bei MCPO wird der Knoten anfangs in den Verteilbaum eingegliedert und durchläuft dabei mehrere Phasen/Stati 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 Verbindungen mit Keepalive-Nachrichten offengehalten.
Das Adaptermodul für H∀Mcast nutzt die MCPO-Objekte so, 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.

E. Einflussfaktoren auf Komplexität

Insgesamt kann man die Komplexität eines Netzwerk-Stacks an drei logischen Aspekten betrachten. Die Komplexität der Control-Operationen beinhaltet die Komplexität, die Netzwerkarchitektur aufzubauen und instand zu halten. Die Komplexität des Data-Forwardings beinhaltet die Weiterleitung der Daten an den einzelnen Knoten und den Weg der Nachrichten durch den Netzwerk-Stack. Bei komponierten Netzwerk-Stack sollte auch die Komplexität der Anwendungsschnittstelle und damit die Konfiguration und das Management des Stacks aus Sicht des Anwenders beachtet werden. Zusammenfassend kommt man bei der Untersuchung von kombinierten Future-Internet-Architekturen zu folgenden Faktoren, welche die Komplexität beeinflussen.

Die Komplexität der Control-Operationen

- Die Anzahl der Cross-Layer-Elemente (vgl. III-A)
- Die Anzahl der Konfigurationsparameter, die in der Abbildung zweier Future-Internet-Architekturen nicht belegt werden können (vgl. III-B)

Die Komplexität der Data-Forwarding-Ebene

- Namensgebung und Adressierung innerhalb der einzelnen Ebenen, sowie die Abbildung der Identifikatoren der Future-Internet-Anwendungen (vgl. IV-A)
- Die Eigenschaften der Datenübertragung, die von den Ebenen garantiert werden (vgl. IV-B2)

Die Komplexität der Anwendungsschnittstelle

- Die Anzahl der logisch verschiedenen Funktionalitäten einer Schnittstelle (vgl. III-B)
- Die Anzahl der Cross-Layer-Elemente, die auf die Anwendungsschnittstelle ausgelagert werden (vgl. III-A, III-C)

V. Implementierung

Das Adaptermodul verbindet die H∀Mcast-Middleware mit der Ariba/MCPO-API. In der Komposition sollen die Konzepte der einzelnen Architekturen erhalten bleiben und und in ihrer vollen Breite genutzt werden (vgl. IV) Das Modul besteht aus drei Hauptkomponenten (vgl. Abbildung 4).

a) mcpo_module: Die MCPO-Module Klasse implementiert das Interface, welches die H∀Mcast-Middleware benutzt, um mit dem Modul zu kommunizieren. Es beinhaltet Multicast-Aufrufe wie join(Group), leave(Group), getParents(...), etc. Weiterhin wertet es die Konfigurationsdaten Aribas aus, die zur Initialisierungszeit benötigt werden und gibt sie an Ariba weiter (z.B. der SpoVNet-Name, Bootstrapknoten etc). Die mcpo_module-Klasse sorgt für eine klare Trennung zwischen den beiden Future-Internet-Architekturen, da sie nur H∀Mcast-Elemente implementiert.

b) mcpo_connection: Die MCPO-Connection-Klasse ist für die Verwaltung des Ariba-Knotens zuständig. Dafür implementiert sie die Schnittstelle zu Ariba, während die `Receiver`-Klasse die Anbindung an MCPO übernimmt. Ariba wird bereits zur Initialisierungszeit von H∀Mcast gestartet, damit die Verbindung zum SpoVNet hergestellt werden kann.

c) Receiver: Die Receiver-Klasse verwaltet ein MCPO-Objekt und implementiert dessen Schnittstelle. Im Gegensatz zur mcpo_connection-Klasse, wird ein Receiver-Objekt erst zum Zeitpunkt des ersten *Join*s oder *Send*s erstellt, da erst zu diesem Zeitpunkt die Informationen über die Gruppen zur Verfügung stehen. Die Receiver-Klasse übergibt Funktionsaufrufe, die es über die MCPO-Connection-Klasse erhält, an das MCPO-Objekt und implementiert die passiven Funktionen wie *"receiveData"* des MCPO-Receiver-Interfaces.

Um die Konzepte der einzelnen Architekturen beizubehalten, muss die Abbildung der H∀Mcast-URI auf die Ariba/MCPO-Identifikatoren so erfolgen, dass Multicast auch bei mehreren wechselnden Gruppen möglichst effizient läuft. Im Ariba/MCPO-Modul wird eine Gruppe durch zwei Identifikatoren beschrieben, SpoVNet-Name und die Service-ID zur Anwendungsidentifikation in Ariba. Innerhalb eines MCPO-Objektes gibt es noch ein weiteres „Gruppenkonzept“, welches ankommende Nachrichten an den Endknoten nach beigetretenen „Gruppen“ (ebenfalls durch Service-IDs dargestellt) filtert. Da MCPO in dieser Form nur einen Verteilbaum pro MCPO-Objekt (also pro Service-Identifikation in Ariba) erzeugt, entspricht diese Filterung genau dem allgemeinen Portkonzept. Eine Alternative bestünde in der Abbildung der H∀Mcast-Gruppen auf diese MCPO-„Gruppen“. Dies würde zu einer verminderten Skalierung führen, da alle Daten durch den kompletten Verteilbaum geflutet werden. Aus diesem Grund wird im Adapter zwischen H∀Mcast und Ariba-MCPO der Port auf die MCPO-Gruppen abgebildet. Die Abbildung der `group @ instantiation` der H∀Mcast-URI erfolgt mittels Hash auf die Service-ID, die ein Receiver-MCPO-Objekt in Ariba beschreibt.

Um Cross-Layer-Informationsfluss zu vermeiden, sollen im Adaptermodul möglichst viele Informationen, die für Ariba/MCPO benötigt werden, direkt aus der H∀Mcast-URI abge-

leitet werden. Dies funktioniert auch für gruppenspezifischen Informationen. Die Information darüber, welchem SpoVNet beigetreten werden soll, und wo dessen Bootstrap-Knoten liegen, kann jedoch nicht mit der H∀Mcast-URI übergeben werden (z.B. als `Credentials`). Würde man diese Informationen in der H∀Mcast-URI übergeben, wäre ein Beitritt des unterliegenden SpoVNet erst möglich, sobald eine Funktion aufgerufen wird, welche die H∀Mcast-URI beinhaltet. Dies würde den Gruppenbeitrittsprozess wesentlich verlangsamen. Aus diesem Grund wird die Information, welchem SpoVNet beigetreten werden soll, von der Anwendung erwartet. Diese Information wird während der Initialisierungsphase von H∀Mcast dem Modul aus einer Initialisierungsdatei übergeben. Dieser Cross-Layer-Informationsfluss ist notwendig, da das Adaptermodul (nach dem Ende-zu-Ende-Prinzip) keine Annahme über das anwendungsabhängige SpoVNet machen kann. Hier entsteht ein Widerspruch zwischen der Einhaltung des Ende-zu-Ende-Prinzip mit der klaren Trennung der Ebenen.

In Ariba/MCPO werden Multicast-Nachrichten mit einem Empfänger adressiert und an alle Gruppenmitglieder weitergeleitet. Die H∀Mcast-URI selbst unterstützt auch Source-Specific-Multicast (nachfolgend SSM). Im SSM-Konzept leiten die Forwarder bestimmte Nachrichten nur weiter, wenn der Empfänger auch die Gruppendaten von genau diesem Sender abonniert hat. Deswegen wird bei SSM für jede Nachricht ein Tupel (Sender, Empfänger) benötigt. In Ariba/MCPO gehört der Sender konzeptionell nicht zur Gruppenbezeichnung, da SSM in Ariba/MCPO nativ nicht unterstützt wird.
Um diese Logik bei der Servicekomposition dennoch zu umzusetzen, wird im Adapter für jedes Tupel (S,E) eine Gruppe angelegt (durch die Bildung eines Hash-Werts über `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 verwendet wird, 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.

Ein weiteres Problem, welches durch die Abbildung der Gruppenbezeichner (`host@instantiation`) auf die Service-IDs entsteht, ist ein Widerspruch des Ariba-Konzeptes mit dieser Abbildung der Adressierung. Im Ariba-Konzept ist eine Service-ID anwendungsspezifisch. Im Adaptermodul ist die Service-ID gruppenspezifisch, damit das NICE-Multicast-Konzept aufgeht. Das bedeutet, dass andere Ariba-Knoten (im gleichen SpoVNet), die die Service-ID 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 betrieben werden soll, in dem noch weitere 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 Service-IDs der Anwendungen (die ja auch frei gewählt werden können), mit den Service-IDs für die Gruppen überschneiden. Durch die Benutzung eines eigenen SpoVNets nur für eine Multicast-Anwendung, können diese Probleme vermieden werden.

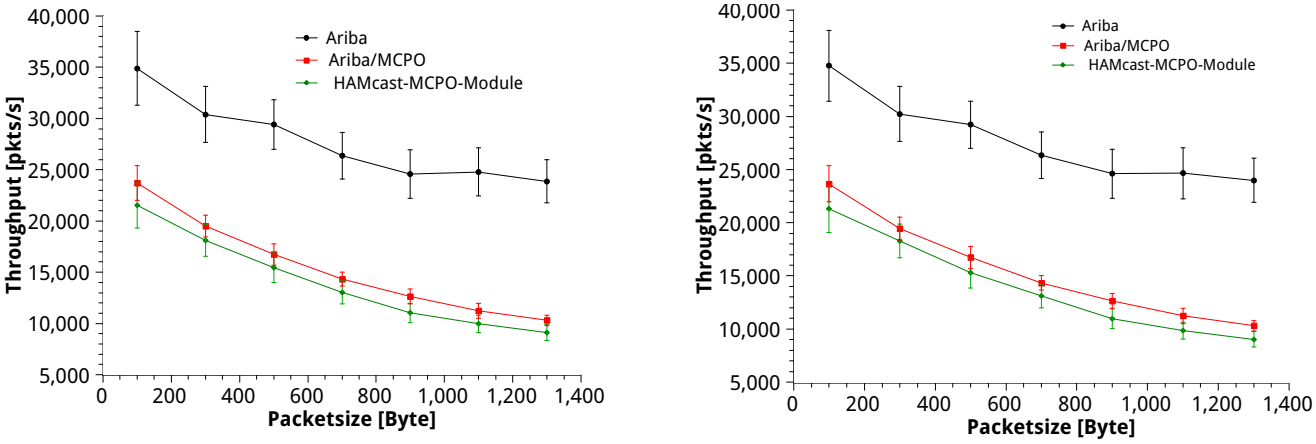
Ein Nachteil dieser Abbildung der Konzepte ist, dass das MCPO-Konzept nicht eins-zu-eins erhalten bleibt. Im nativen MCPO ist es nicht möglich die Service-ID zu bestimmen, mit der sich MCPO bei dem Ariba-Objekt registriert. Dies wurde nachimplementiert, damit ein Verteilbaum pro Gruppe ermöglicht wird. Diese Abbildung der Konzepte ist dann nur noch unter der ursprünglichen Service-ID mit dem ursprünglichen MCPO-Konzept kompatibel. Durch diese Abbildung wird eine bessere Skalierung der Multicast-Verteilbäume erreicht, allerdings zu dem Preis, dass bei jedem Gruppenbeitritt ein Knoten in einen Verteilbaum eingegliedert werden muss. Dies verlängert die Gruppenbeitrittsverzögerung im Vergleich zu dem nativen MCPO-Gruppenkonzept Insgesamt wurde der ID/Locator-Split Aribas erhalten und es wird keine Funktionalität Aribas eingeschränkt.

VI. Evaluierung

A. Test: Durchsatz und CPU-Auslastung

Getestet wurde das Adapter-Modul auf Paketdurchsatz und CPU-Belastung im Vergleich mit Ariba, sowie im Vergleich zu Ariba/MCPO. Es handelt sich um drei identische Testprogramme, welche so viele Pakete wie möglich mit der jeweiligen Technik (Ariba, Ariba/MCPO oder H∀Mcast-Middleware + Modul) an eine Gruppe (bzw. in Aribas Fall, direkt an einen zweiten Knoten) versenden. Die Empfängerseite prüft, ob die Daten in der richtigen Reihenfolge ankommen und zählt verlorene Pakete. Der Durchsatz von Ariba ohne MCPO bildet als unterliegendes Netzwerk die absolute Obergrenze, die mit MCPO und und dem Adapter-Modul erreicht werden kann. Aus diesem Grund wird der maximale Durchsatz den Ariba erreichen kann, mitgemessen.

Mit einem Bash-Skript wird sekündlich die CPU-Belastung gemessen. Geschrieben wurden die Durchsatztests in C++ und auf zwei identischen Rechnern (CPU: Intel Core i7-870 8M Cache, 2.93 GH, Speicher: 8GB, Betriebsystem: Ubuntu 12.04, Ariba v0.7.1b, MCPO v0.5.1, H∀Mcast v0.5) ausgeführt. Um mögliche anderweitige Einflüsse zu vermeiden, sind die Testrechner während Testlaufzeit direkt mittels eines lokalen Netzwerkkabel verbunden und werden ausschließlich für die Tests verwendet, sodass neben den Testkomponenten einzig das Betriebssystem läuft. Die Tests werden in Abhängigkeit von Paketgrößen zwischen 100 und 1300 Byte in 200er Schritten gemessen. Pro Paketgröße entstanden die Ergebnisse aus ca. 700 Testsekunden, aufgeteilt in jeweils 10 Testläufe. Die Testdaten enthalten weder den Verbindungsaufbau, noch den Verbindungsabbau, um die Messungen nicht zu verfälschen.



(a) Durchsatz Sender

(b) Durchsatz Receiver

Abbildung 5. Durchsatz in Abhängigkeit von der Paketgröße

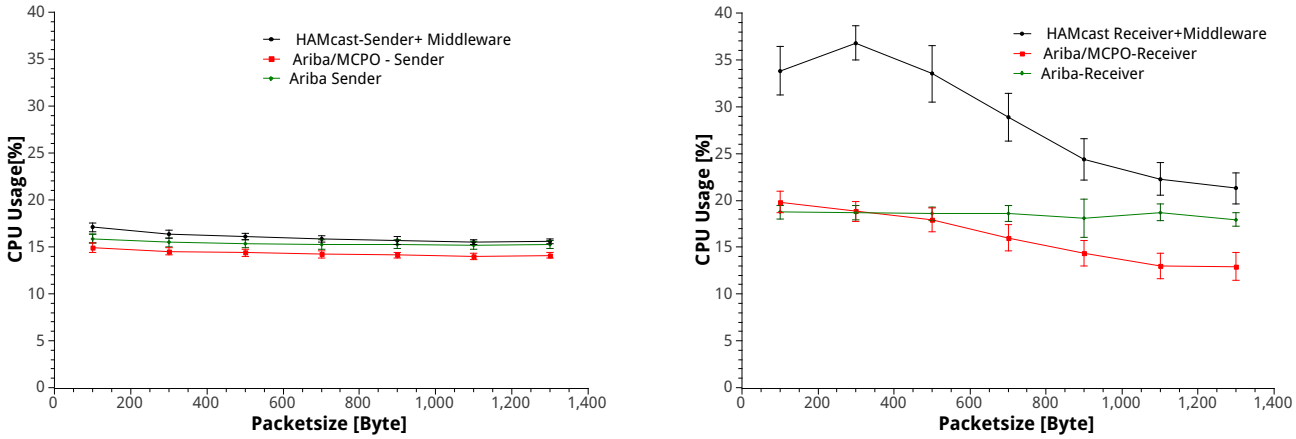
1) Durchsatz: Der Durchsatz bestimmt sich durch Pakete, die pro Sekunde versendet werden, in Abhängigkeit von der Paketgröße (Abbildung 5). Ariba versendet wesentlich mehr Pakete pro Sekunde als Ariba/MCPO, da dieses zusätzlich das Overlay-Multicast-Netz verwaltet, sowie einen weiteren Header vor den Ariba-Header einfügt. Als weitere Abstraktionsschicht verringert das Adapter-Modul zusammen mit der Middleware ebenfalls den Durchsatz, da in der H∀Mcast-Middleware eine weitere Kopieroperation durchgeführt wird. Der Durchsatz für das Modul ist durch die maximale Geschwindigkeit von Ariba/MCPO begrenzt. Im Vergleich zu einem H∀Mcast Modul, welches ein Scribe [11] Overlay benutzt (bis zu 250.000 Pakete/s), oder einem Modul welches ein BIDIR-SAM-Overlay [12] benutzt (bis zu 290.000 Pakete/s), ist Ariba/MCPO deutlich langsamer (siehe auch [1]). Diese Differenz erklärt sich durch den Einsatz des TCP- bzw. SCTP-Protokolls seitens Aribas, um eine zuverlässige Verbindung zur Verfügung zu stellen. Die meisten Multicast-Anwendungen (wie auch Scribe) verwenden verlustbehaftete Protokolle und sind dementsprechend schneller, da keine gesicherte Übertragung stattfinden muss. Zusätzlich zu dem Aufwand den Ariba für das SpoVNet betreibt, werden in MCPO Maintenance-Nachrichten verschickt und verarbeitet. Weiterhin wird der NICE-Verteilbaum aufgebaut und über die Ariba-Nachrichten-Verzögerungen die Wahl für den besten Clusterleader getroffen. Die Overlayverwaltung sorgt für zusätzliche Last auf dem Netzwerk und einzelnen Knoten und verlangsamt damit den Versende- und Empfangsprozess. Es zeigt sich, dass mit jeder Ebene in einem kombinierten Netzwerk-Stack neue Verwaltungsaufgaben hinzukommen, und der Durchsatz der Anwendung immer von der langsamsten Zwischenebene abhängt.

2) Paketverlust: Obwohl Ariba, als unterliegende Abstraktionsschicht für MCPO, zuverlässige Verbindungen zur Verfügung stellt, kann es zu gelegentlichen Nachrichtenverlust in Ariba/MCPO und dem Adapter-Modul kommen. Dieser Nachrichtenverlust ist gering, beläuft sich in einem Intervall von einer Sekunde durchschnittlich auf ein Paket und tritt sehr unregelmäßig auf (im Mittel alle 19,2 Sekunden).
In einem Testaufbau, der über die Middleware und das Adapter-Modul schnellstmöglich genau 100.000 Nachrichten (200 Byte groß) versendet, wurden, an verschiedenen Stellen des entstandenen Netzwerk-Stacks, Zähler implementiert, um die Zuverlässigkeit der Komponenten zu überprüfen und zu bestimmen, an welcher Stelle die Pakete verloren gehen. So verschwinden von den durch das Modul versendeten 100.000 Nachrichten, im Schnitt 2 pro Messung. Die Durchsatztests von Ariba ohne MCPO zeigen, dass Ariba eine zuverlässige Nachrichtenübertragung bietet (welches im Ariba-Konzept so vorgesehen ist), hierbei ist kein Paket verloren gegangen.
Das sendende Ariba/MCPO-Modul in der Middleware hat ebenfalls keinen Nachrichtenverlust zwischen Sendeapplikation und dem MCPO-Modul feststellen können.
Empfängerseitig wurde im Ariba/MCPO-Modul der Middleware jedoch der beschriebene Paketverlust festgestellt. Daraus lässt sich schließen, dass die MCPO-Komponente diese Nachrichten verliert und damit keine zuverlässige Verbindung bietet. Der Paketverlust ist eine Folge der widersprüchlichen Ziele Aribas und MCPOs bezüglich der Zuverlässigkeit (vgl. III-C, IV-B2). Da der Paketverlust insgesamt sehr gering ist, wirkt sich dieses Verhalten kaum auf die Durchsatz-Messungen aus und die Messungen der Nachrichtenanzahlen sind bei Sender und Empfänger annähernd gleich (vgl. Abbildung 5).

3) CPU-Auslastung: Die Beobachtung der Endknoten zeigt die CPU-Auslastung, welche die Sende- und Empfangsprozesse maximal bei vollem Durchsatz der Testprogramme auf den Testrechnern erzeugen (Abbildung 6).

Hier wird deutlich, dass alle Sender zu ähnlicher CPU-Belastung führen, egal welche Paketgröße der Sender mit welchem Test sendet. Trotz Overhead benötigt der MCPO-Test weniger CPU-Zeit als Ariba, da er wesentlich weniger Pakete versendet (vgl. Abbildung 5).

Am Empfängerknoten sorgt die sinkende Anzahl der Pakete/s bei steigender Paketgröße für eine Verminderung der CPU-Last, da weniger Pakete kopiert werden müssen. Dieser Zusammenhang gilt für Ariba/MCPO sowie H∀Mcast mit Ariba/MCPO-Modul. Der Aufwand der reinen Ariba-Knoten unterscheidet sich nicht zwischen den verschiedenen Paketgrößen. Der Aufwand des Ariba-Empfängers ist demnach nicht in



(a) CPU-Belastung Sender

(b) CPU-Belastung Receiver

Abbildung 6. CPU-Auslastung der Testknoten bei maximalem Durchsatz

dem Maße von der Paketanzahl abhängig wie MCPO, sondern mehrheitlich von der absoluten Datenmenge. Beim Empfänger ist die CPU-Auslastung des MCPO-Moduls zusammen mit der H∀Mcast-Middleware bei kleinen Paketen sehr hoch (vgl. akkumulierter Graph in Abbildung 7).

Grund für die hohe CPU-Auslastung ist einerseits die Middleware, welche eine Kopieroperation zusätzlich zur Technologie darunter (in diesem Fall Ariba/MCPO) pro Paket benötigt, um die Pakete zuverlässig an die überliegende Applikation weiterzuleiten. Die Applikation benötigt weiterhin etwas CPU-Zeit, um die Pakete über IPC zu empfangen. Je mehr Pakete empfangen werden, desto mehr Kopieroperationen werden benötigt. Dies erklärt die erhöhte CPU-Belastung bei kleineren Paketgrößen.

B. Test: Auswirkung komplexer Netzwerk-Stacks auf Zeitverhalten bei Gruppenbeitritt

In einem komplexen, komponierten Netzwerk-Stack muss eine Nachricht, bevor sie letztendlich abgeschickt wird, durch eine Menge von Netzwerkschichten gereicht werden. Diese Ebenen fügen den Nachrichten neue Header hinzu, serialisieren sie, und verwalten ihr eigenes Netzwerk. Der Aufwand, der für einen bestimmten Multicast-API-Aufruf betrieben wird, ist immer gleich. Je mehr Abhängigkeiten im Netzwerk-Stack

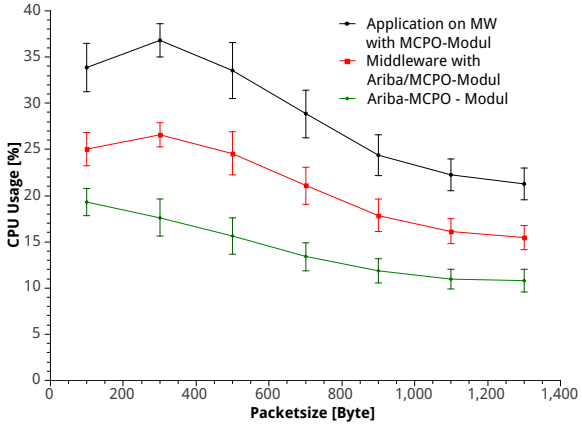


Abbildung 7. akkumulierte CPU-Last der Komponenten einer Ariba/MCPO-H∀Mcast-Anwendung

vorhanden sind, desto mehr Einstellungen müssen überprüft und gegebenenfalls hergestellt werden (vgl. III-B).

Im Falle des H∀Mcast-Ariba/MCPO-Stacks 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. In letzterem Fall ist die „Join“ Operation schnell, weil es sich um eine lokale Operation handelt. Der Gruppe wurde schon beigetreten, und die Pakete an diesen Port werden nicht weiter gefiltert. Wurde der Gruppe noch nicht beigetreten, müssen verschiedene Operationen im lokalen Stack sowie über das Netzwerk abgewickelt werden, was einige Zeit in Anspruch nehmen kann.

- Erstellen eines lokalen Receiver-Objektes, Hash-Bildung der `group @ instantiation`, und Erstellen des MCPO-Objektes mit der neuen Service-ID
- Registrierung des MCPO-Objektes mit Service-ID beim Ariba-Node-Objekt (lokal)
- Broadcast Rendezvous-Point-Lookup-Message in SpoVNet für diese Service-ID (MCPO)
- Bootstrapping-Phase zur Eingliederung in den MCPO-Verteilbaum (MCPO)

Nachfolgend wird die Zeit gemessen, die eine Anwendung warten muss, bis nach einem Gruppenbeitritt die ersten Daten ankommen. Um eine realistische Einschätzung zu erhalten, ist es notwendig, einen Testaufbau zu wählen, der kein Trivialbeispiel ist und dabei handhabbar bleibt. Damit der Aufbau des SpoVNets den Test nicht verfälscht, treten alle Knoten zuerst dem gleichen SpoVNet bei, und erst, wenn das für alle Knoten abgeschlossen ist, der Gruppe selbst. Weiterhin treten die Knoten, an denen die Beitrittsverzögerung gemessen wird einer Multicast-Gruppe bei, die schon stabil existiert. Auf diese Weise wird verhindert, dass die Messung durch den initialen Gruppen-Aufbau verfälscht wird.

Der Testaufbau (in Anlehnung an [13]) besteht aus einer festen Gruppe mit 10 Knoten, die alle derselben Gruppe beigetreten sind (nachfolgend: „Sender“, vgl. Abbildung 8). Einer dieser Knoten sendet regelmäßig Daten (1000 Nachrichten /s). 11 weitere Testknoten treten der Gruppe bei und messen dabei

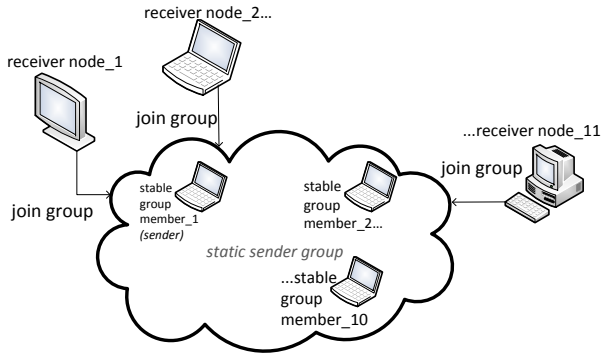


Abbildung 8. Testaufbau zum Testen der Beitrittsverzögerung. Statische, fertig initialisierte Gruppe, sowie diverse Knoten, welche der Gruppe beitreten.

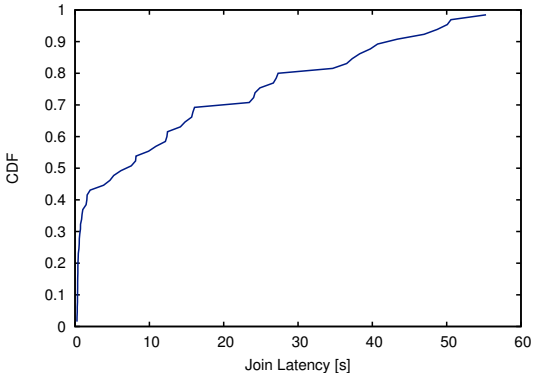


Abbildung 9. Verzögerung bei Gruppenbeitritt

die Verzögerung, bis die erste Nachricht ankommt.

Für diesen Test wurden 21 Knoten verwendet, die in verschiedenen Orten Europas (Planetlab) stehen. Auf den Knoten läuft Fedora 12, und es wurde Ariba v0.8.2 mit einer leicht angepassten Version MCPO v0.5.1 verwendet. MCPO wurde verändert, um die leicht veränderte Schnittstelle von Ariba, im Vergleich zur Vorgänger-Version, zu verwenden. Weiterhin wurde angepasst, dass das Adapter-Modul verschiedene MCPO-Objekte auf einem Knoten starten kann (variable Service-ID in Konstruktor).
Die Rollen der Knoten als Sender 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 verwendet wurden.

1) Testergenisse: Join-Latency: Die Verzögerung bei Gruppeneintritt (vgl. Abbildung 9) beträgt bei 44 % der Knoten unter einer Sekunde, und damit in einem Intervall, welches den meisten Anwendungen gerecht wird. Auffällig ist die Menge an Knoten, die mehr 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.

2) Gründe für lange Gruppenbeitrittsverzögerung bei spezifischen Knoten: Die Gründe für die wiederholte hohe Gruppenbeitrittsverzögerung 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.

a) 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 Routingnachbarn, 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 Overlaynetzen 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), nachdem 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 Routingknoten an entstandenen Engpässen von der Menge der Daten überfordert werden. Wenn Ariba in einen Timeout läuft, wird versucht diese Daten noch einmal zu verschicken. Wenn ein Zwischenknoten die Daten nicht weitergeschickt hat, weil er zu stark 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. In Ariba/MCPO werden alle Nachrichten, Anwendungsdaten sowie Maintenance-Nachrichten gleichrangig behandelt. Ist ein Netz stark ausgelastet, ist es möglich, dass auch „Join"-Nachrichten im normalen Datenfluss untergehen. Mit einer Priorisierung der Maintenance-Nachrichten könnte man diesem Problem entgegenwirken. In diesem Test reicht die Sendelast des Netzes wahrscheinlich nicht als Grund für die lange Beitrittsverzögerung, da das Netz nicht ausgelastet war.

Ariba ist zuverlässig und stellt deswegen Rückgabewerte zur Verfügung, ob eine Nachricht versendet wurde oder nicht. MCPO, als unzuverlässiger Multicast, wertet diese Rückgabewerte nicht aus. Dies macht beim Senden einer einfachen Applikationsnachricht Sinn, da das Auswerten dieser Daten beim schnellen Senden die Anwendung verzögert und unzuverlässiger Multicast von vornherein mit Nachrichtenverlust rechnet. Beim Versenden einer Join Nachricht in der Initialisierungsphase, wie auch bei vielen anderen Maintenance-Funktionen, kann das Wissen darüber, ob eine Nachricht von Ariba versendet wurde, dazu benutzt werden, den „Join"-Prozess zu optimieren (z.B. muss nicht auf ein MCPO-Timeout gewartet werden, wenn das Senden der Maintenance-Nachricht direkt auf Ariba-Ebene fehlschlägt und Ariba dies zurückmeldet).

VII. Zusammenfassung

Future-Internet-Architekturen behandeln bestimmte Probleme im derzeitigen Internet. Dafür stellen sie bestimmte Konzepte bereit, die diese Probleme lösen sollen. Anhand der Kombination von zweier Architekturen wurde untersucht wo die Komplexität entsteht und wie sie sich auswirkt. Dafür wurden Merkmale gezeigt, die die Komplexität beeinflussen. Zu ihnen gehören Cross-Layer-Komponenten, Abbildung der Konfigurationsparameter und die Auflösung konzeptioneller Widersprüche. Die Abbildung der Future-Internet-Konzepte funktioniert selten ohne Einschränkungen. Der Durchsatztest hat gezeigt, dass die Kombination von einem verbindungslosen Multicast-Konzept über einem verbindungsorientierten Overlay relativ viel Leistungseinbußen hinnehmen muss, da es den Verbindungsabbau und die zuverlässige Übertragung von TCP bzw. SCTP immer mitbezahlt. Weiterhin kommen mit jeder Ebene in kombinierten Architekturen neue Maintenance-Aufgaben hinzu, welche die Komponente komplexer machen. Es wurde gezeigt, wie die Abbildung der Konzepte die Leistung und Geschwindigkeit der einzelnen Funktionen beeinflusst. So ist der Datendurchsatz der Anwendung immer von der langsamsten Ebene abhängig.
Der Test der Gruppenbetrittsverzögerung zeigt, wie die Wahl der Abbildung einer Funktion der Control-Plane maßgeblich deren Geschwindigkeit bestimmt.

Die Kombination von Future-Internet-Architekturen bringt einige Fehlerquellen mit sich, die ein Anwendungsprogrammierer nicht unbedingt überschauen kann. Dies ist vor allem der Fall, wenn sich die Systeme gegenseitig beeinflussen oder vom Anwender Konfigurationen für tiefer liegende Schichten verlangt werden.

VIII. Ausblick

Im Bezug auf den Gruppenbeitrittsverzögerungstest wäre ein Vergleichstest mit einem anderen Overlay notwendig, um die gemessene Gruppenbeitrittsverzögerung richtig einzuschätzen. Weiterhin wäre zu testen, wie sich variable Zeitfenster für die Timeouts in Ariba und MCPO auswirken, und wie sich der Beitritt in mehrere Gruppen bei der Komposition auswirkt.

Im Abschnitt über Komplexität und Problemstellungen wurden weitere Themen angeschnitten, die bisher nicht weiter untersucht wurden. Dazu gehören die Future-Internet-Architekturen unter großer Sendelast, in einem größeren Verteilbaum, sowie die Auswertung, wie sich die Vorteile von synchronen und asynchronen Schnittstellen bemerkbar machen, oder wie das System mit einer gesteigerten Anzahl von „Join" und „Leave" umgeht.

Literatur

[1] S. Meiling, D. Charousset, T. C. Schmidt, and M. Wählisch, "HAMcast – Evaluierung einer systemzentrierten Middleware-Komponente für einen universellen Multicast-Dienst im Future Internet," *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, vol. 35, Mai 2012.

[2] C. Hübsch, C. P. Mayer, S. Mies, R. Bless, O. P. Waldhorst, and M. Zitterbart, "Reconnecting the internet with ariba: self-organizing provisioning of end-to-end connectivity in heterogeneous networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, Jan. 2010.

[3] K. I. f. T. Institut für Telematik, "Documentation/mcpo - ariba - overlay-based virtual network substrate." http://ariba-underlay.org/wiki/Documentation/MCPO, May 2013.

[4] M. Wählisch, T. C. Schmidt, and S. Venaas, "A Common API for Transparent Hybrid Multicast," IRTF Internet Draft – work in progress 08, IRTF, April 2013.

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

[6] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of SIGCOMM '02*, (New York, NY, USA), ACM Press, 2002.

[7] R. Bush and D. Meyer, "Some Internet Architectural Guidelines and Philosophy," RFC 3439, IETF, December 2002.

[8] B. E. Carpenter, "Architectural Principles of the Internet," RFC 1958, IETF, June 1996.

[9] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Trans. Comput. Syst.*, vol. 2, Nov 1984.

[10] V. Jacobson, "Congestion Avoidance and Control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, August 1988.

[11] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, 2002.

[12] M. Wählisch, T. C. Schmidt, and G. Wittenburg, "BIDIR-SAM: Large-Scale Content Distribution in Structured Overlay Networks," in *Proc. of the 34th IEEE Conference on Local Computer Networks (LCN)* (M. Younis and C. T. Chou, eds.), (Piscataway, NJ, USA), IEEE Press, October 2009.

[13] D. Stopp and B. Hickman, "Methodology for IP Multicast Benchmarking," RFC 3918, IETF, October 2004.

Downloads

Ariba & MCPO: `http://www.ariba-underlay.org/`
H∀Mcast: `http://www.realmv6.org/hamcast.html`