Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung

Sebastian Zagaria

A Pull-Multicast for HAMcast

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einleitung

Beliebte Internetanwendungen wie z.B Video- und Audio-Konferenzen, IPTV, Chatprogramme und Filesharing basieren auf einer Art von Gruppenlogik. Die Multicast Unterstützung von IPv4 und IPv6 bietet einen effizienten Mechanismus für das Verteilen von Daten in Gruppen. Jedoch ist eine Nutzung für eine global verteilte Gruppenkommunikationsanwendung nicht möglich, da Multicast im globalem Routing nicht unterstützt wird. Entwickler von Gruppenkommunikationsanwendungen sind deshalb gezwungen eigene Verteilungsmechanismen, auf Kosten höherer Komplexität und geringerer Effizienz, zu entwickeln.

Hybride Multicast-Netzwerke kombinieren die Vorteile von Application-Layer-Multicast (ALM) und nativen Multicast-Technologien um die Lücken zwischen Multicast-Netzwerken zu füllen. Bei diesem Ansatz wird die optimalste zur verfügungsstehende Multicast-Technologie verwendet und über Multicast-Gateways eine Verbindung zu anderen abgegrenzten Multicast-Netzwerken hergestellt. Im Bereich des Filesharing und einiger kommerzieller IPTV Anwendungen haben sich unstrukturierte Chunk-Trading-Netzwerke als Verteilungstechnologie durchgesetzt. Als bekanntestes Beispiel für ein solches Chunk-Trading-Netzwerk gilt BitTorrent. Diese als unstrukturierte Peer-to-Peer-Netzwerke bezeichneten Technologien besitzen einen verringerten Managementaufwand, geringere Komplexität und eine höhere Bandbreitenausnutzung gegenüber strukturierten Verteilungsmechanismen.

Ziel der Masterarbeit ist die Entwicklung und Implementation eines Pull-Multicast für HAMcast. Die Verteilung der Daten basiert dabei auf einem Chunk-Trading Verteilungsmechanismus. Dies stellt eine völlig neue Kombination aus Publish/Subscribe und Chunk-Trading dar. Es soll die Performance und Skalierbarkeit mit anderen Multicast-Technologien verglichen werden.

Inhalt dieser Ausarbeitung ist das Konzept für den Pull-Multicast. Aufgrund der Längenbeschränkung dieser Ausarbeitung liegt der Fokus auf der Implementation der API und der Gruppenverwaltung für den Pull-Multicast. Als Erstes werden Grundlagen zu der Common-API und Distributed Hash Table vorgestellt. Danach wird die Implementation der API und das Konzept für die Gruppenverwaltung vorgestellt. Abschließend wird ein Fazit gegeben und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Grundlagen

2.1 Common API for Transparent Hybrid Multicast

Das Ziel der Common-API [WSV12] ist es, Anwendungsentwicklern die Möglichkeit zu bieten Gruppenkommunikationsanwendungen zu entwickeln die überall einsetzbar sind, unabhängig von Status der Netzwerkserviceverteilung. Anders als die bisherige Multicast-Socket API ist das Naming und Adressing Technologie unabhängig und universell. Die API definiert technologie-unabhängige Aufrufe für die Gruppenkommunikation. Dies ermöglicht es Gruppenanwendungen zu entwickeln ohne sich bei der Entwicklung auf eine bestimmte Technologie festzulegen. Die Auswahl der Verteilungstechnologie wird zur Laufzeit bestimmt oder vom Benutzer festgelegt. Die Anwendungen werden dadurch so flexibel das, die darunterliegende Verteilungstechnologie je nach Bedürfnis des Benutzers oder des Netzwerkstatus ausgetauscht werden kann, ohne Anpassungen an dem Programm vorzunehmen. Des weiteren ist eine solche API für die Implementierung von hybriden Multicast-Netzwerken geeignet, in der mehrere verschiedene Technologien für den Benutzer verborgen miteinander interagieren wie z.B bei HAMcast [MCSW10]. Zur Adressierung von Multicast-Gruppen werden Uniform Resource Identifier (URI) eingesetzt. Diese bieten eine technologie-unabhängiege Adressierung von Gruppen.

Aufbau der URI ***scheme :// group @ instantiation : port /sec-credentials***
Das Scheme gibt den Namensraum an und legt die Syntax und Semantik des nachfolgenden Teils fest. Das Group Argument identifiziert die Multicast-Gruppe. Die Instantiation identifiziert die Entität die, die Instanz der Gruppe generiert, zum Beispiel die Source bei Source Specific Multicast. Der Port identifiziert eine bestimmte Applikation auf einem Knoten. Die Sec-Credentials können benutzt werden um optionale Sicherheitsberechtigungen zu implementieren.

Die API ist in vier Kategorien von Methoden unterteilt, die Group-Management-Calls für das beitreten und verlassen von Multicast Gruppen, die Send/Receive-Calls. Die Socket-Options-Calls, über die z.B ein bestimmtes Technologieinterface ausgewählt werden kann. Die Service-Calls, die Informationen über den Aktuellen Netzwerkstatus des Knoten bieten.

2.2 Distributed Hash Table

Eine Distributed Hash Table (DHT) wird als strukturiertes Peer-to-Peer-Netzwerk bezeichnet. DHTs sind selbstorganisierende Netzwerke die keine zentrale Instanz besitzen. Alle Mitglieder des Netzwerkes sind gleichberechtigt für das Routing und Management verantwortlich. Damit wird eine hohe Skalierbarkeit erreicht, so dass Netzwerke eine große Menge von Teilnehmern unterstützen. Die Last für Management und Routing ist gleichmäßig zwischen den Teilnehmern verteilt. Des weiteren sind DHTs ausfallsicher so das, dass plötzliche ausfallen von Teilnehmern, das hinzukommen oder verlassen von Teilnehmern nicht zu Fehlern oder gar zum Zusammenbruch des Netzwerks führen.

Distributed Hash Table arbeiten mit einem Lookup-Service der ähnlich den Hash Tables ist wie man sie aus den gängigen Programmiersprachen kennt, in denen Schlüssel Werte Paare gespeichert werden. Jedem Knoten und allen Daten die in einer DHT gespeichert werden wird ein einzigartiger Schlüssel (Key) zugewiesen. Dieser Schlüssel repräsentiert die Adresse der Knoten oder auch Daten im Overlay-Netzwerk. Generell werden diese Adressen durch eine Hashfunktion generiert, häufig verwendet wird beispielsweise der Secure Hash Algorithm (SHA1). Zum Beispiel, wenn ein Knoten dem Overlay beitritt wird mithilfe der Hashfunktion und einem Wert, wie der IP-Adresse und Port des Knoten, die Adresse generiert. Die Länge der Adresse kann sich bei den verschiedensten Implementierungen unterscheiden, meistens haben die Adressen eine Länge von 128 bis 160 Bit. Die Adressen von Daten werden auf gleiche weise generiert indem die Hash Funktion z.B. auf den Namen der Datei angewendet wird. Die Hashfunktionen garantieren eine gleichmäßige Verteilung der Overlay-Adressen entlang des Adressbereiches. Folglich teilen sich Knoten und Daten denselben Adressraum. Der Adressraum abstrahiert dabei die Adressen der Knoten und Daten. Es wird ein virtuelles Netzwerk mit eigenem Adressraum über dem eigentlichen Netzwerk erstellt.

Die Abbildung 2.1 zeigt ein Beispielnetzwerk, indem der Routing-Algorithmus das Overlay-Netzwerk in einer Ringstruktur anordnet. Das unten dargestellte Netzwerk zeigt die Verbindungen der Knoten im realen Netzwerk. Jeder Knoten im Overlay ist verantwortlich für das pflegen von Routing Informationen und Daten in einen ihn zugewiesenen Bereich. Der verantwortliche Adressbereich ist dabei von der Adresse des Knoten abhängig. Ein Knoten ist verantwortlich für Daten deren Adresse am nächsten an der Adresse des Knoten sind. Zum Beispiel Knoten A hat die Adresse 0001, Knoten B die Adresse 1000 und Daten die Adresse 0010 dann ist der Knoten A für diese Daten verantwortlich da seine Adresse näher an der Adresse der Daten liegen. Die Daten werden direkt bei den verantwortlichen Knoten gespeichert oder auch als Referenz zu dem Knoten der die Daten hält. Analog zu den Daten, Pflegen die Knoten in einem Overlay auch

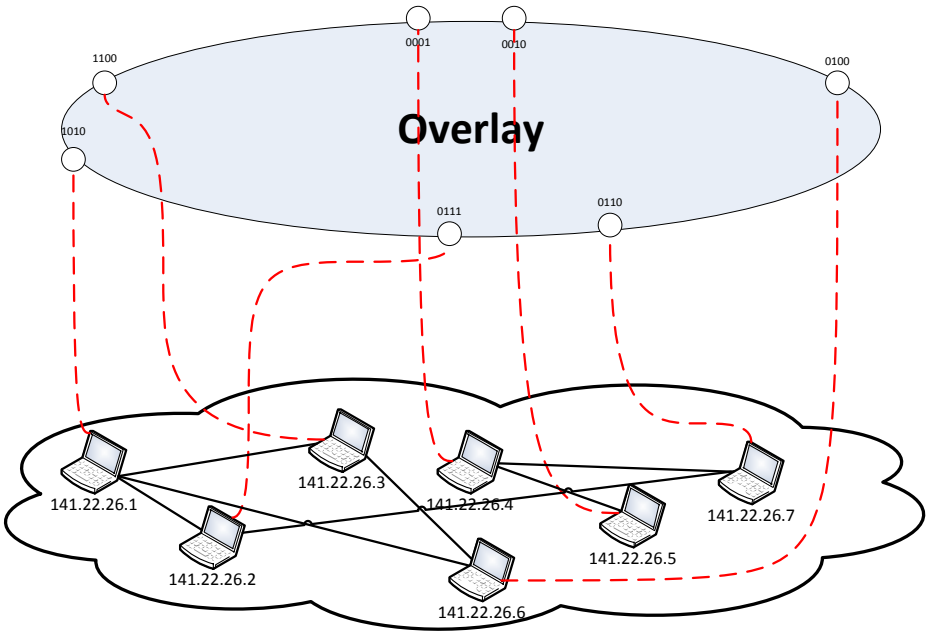

Abbildung 2.1: Overlay Beispiel Netzwerk

Routing-Informationen. Diese sind dafür da Knoten und Daten innerhalb des Overlays zu finden. Jeder Knoten besitzt teilweises Wissen über die gesamte Overlay-Topologie. Kennt ein Knoten nicht die direkte Route zu seinem Ziel, so kennt er zumindest einen Knoten, der dieses Wissen besitzt. Dabei werden Anfragen immer an den Knoten weitergeleitet, der am nächsten an der zu suchenden Adresse ist. Das Verfahren hängt hierbei vom verwendeten Routing Algorithmus ab. Es existiert eine Vielzahl verschiedener DHTs wie z.B Chord [SMLN+03], Pastry [RD01] oder Kademlia [MM02]. Die genannten DHTs unterscheiden sich in ihren Eigenschaften bezüglich der Skalierbarkeit, Robustheit und Netzwerkpartitionierung.

3 Pull Multicast Konzept

3.1 Übersicht

Das Ziel ist das Konzept und Implementierung einer Gruppenkommunikations-Technologie für HAMcast, die zur Verteilung der Daten einen Chunk-Trading Algorithmus verwendet. Diese Technologie wird auch als Pull-Multicast bezeichnet, da die an der Verteilung beteiligten Teilnehmer aktiv Daten anfordern und diese über Gruppennamen adressiert werden. Dieser Pull-Mechanismus unterscheidet sich dadurch deutlich von anderen Multicast-Technologien in denen die Daten direkt von der Quelle in das Netzwerk gesendet werden.

Für die Realisierung des Pull-Multicast wird ein Chunk-Trading Algorithmus zur Verteilung der Daten benötigt. Die verschiedenen Arten von Chunk-Trading Algorithmen [JX] [HLR08] [Coh03] wurden bereits in Anwendungen 2 vorgestellt. Weil Chunk-Trading Netzwerke unstrukturiert sind, wird für die Verwaltung der Netzwerke und benötigten Meta-Informationen, eine zweite Instanz erforderlich. Bei dem Pull-Multicast wird folglich auch eine zusätzliche Instanz benötigt, die für die Verwaltung der Gruppen und die Meta-Informationen zuständig ist. Des weiteren ist die Verwendung der Common-API unerlässlich für eine hybride Kommunikation wie sie bei HAMcast eingesetzt wird.

3.2 Benutzung der Common API

Die Nutzung einer universellen und gut definierten API schließt die Möglichkeit aus, dass die Funktionen an die Bedürfnisse der darunterliegenden Technologie angepasst werden können. Eine Veränderung oder sonstige Anpassungen der API sind nicht möglich. Betrachtet man die einzelnen Aufrufe ist anzunehmen das die Calls der API an die Bedürfnisse von Push-Verfahren angepasst sind. Um alle benötigten Funktionalitäten für den Pull-Multicast zu unterstützen können zusätzliche Argumente bzw. die Interpretation der vorhanden Argumente über das ßcheme"der Gruppenadresse codiert werden. Wie in abschnitt 2 beschrieben bietet die Common-API vier Kategorien von API Calls. In diesem Abschnitt liegt der Fokus auf der Implementation der vier Hauptaufrufe, die benötigt werden, um Daten zu veröffentlichen zu verteilen und zu empfangen. Die Aufrufe Join und Leave aus den Group-Management-Calls und die Aufrufe

aus der Send/Receive Kategorie. Alle anderen aufrufe müssen bei der späteren Implementation natürlich auch vorhanden und unterstützt werden, jedoch sind diese Aufrufe geradeaus zu implementieren und benötigen deshalb keine nähere Betrachtung für die eigentliche Realisierung.

3.2.1 Die Gruppenadresse

Das wichtigste Argument bei den Aufrufen ist die URI (Gruppenadresse), über die zusätzliche Argumente an das Technologiemodul weitergegeben werden können. Zuerst wird das Scheme angegeben was den Namensraum bestimmt, in diesem Fall mesh://. Darauf folgt der Gruppenname, dieser wird verwendet um die Gruppe zu identifizieren. Im Fall von Pull-Multicast ist der Gruppenname in zwei Abschnitte unterteilt, einen Pfad und einem Namen. Der Pfad gibt an, wo Daten gespeichert werden sollen. Alternativ kann dieser Pfad auch über die *Middleware.ini* von HAMcast (Config-Datei) an das Technologiemodul weitergegeben werden. Die Angabe des Pfades ist ebenfalls für das versenden von Daten wichtig, da beim Chunk-Trading ganze Dateien ausgetauscht werden und keine einzelnen Nachrichten wie es z.B bei anderen verfahren üblich ist. Der Name ist der Identifikator für die Gruppe, über den Namen werden die Daten adressiert. Ein Benutzer der diese Daten anfragen möchte benötigt also den Gruppennamen zur Identifikation. Ist in der Config-Datei ein Pfad angegeben so reicht der Gruppenname aus um die Daten zu speichern oder zu versenden, die Daten werden dann über den Pfad gespeichert oder abgerufen. Ist weder in der Config-Datei noch über die URI ein Pfad angegeben, werden die Daten nicht permanent gespeichert und nur an die Applikation weitergereicht. Um dennoch Daten weiter an andere Teilnehmer des Netzwerkes verteilen zu können, müssen diese temporär gespeichert werden. Folglich existieren diese temporären Daten so lange, wie die Session offenbleibt, durch ein Socket ***close()*** oder ein ***leave()*** der Gruppe wird die Session beendet. Das weglassen des Pfades ist nur möglich wenn ein Benutzer Daten Empfangen möchte. Zum versenden von Daten muss immer der Pfad zu den Daten angegeben werden.

3.2.2 Die Calls

join(in Socket s, in Uri groupName, out Int error)
Der Join-Call bietet die Funktion einer Multicast Gruppe beizutreten. Im Fall des Pull-Multicast bedeutet dies, das der Benutzer das Interesse an Daten bekundet die über diese Gruppe ausgetauscht werden. Sobald das Interesse an den Daten bekundet wurde, wird mit dem Datenaustausch begonnen. Möchte der Benutzer die Dateien herunterladen, die über die Gruppe ABC getauscht werden, so wird nach dem Sheme als erstes ein absoluter oder relativer Pfad zu dem gewünschten Speicherort angegeben. Dieses Argument ist optional und kann ebenfalls in der HAMcast

Config-Datei angegeben werden. Als nächstes folgt der Name der Gruppe dieser Identifiziert die Angeforderten Daten. Beispiel für den Aufbau einer Gruppenadresse:
mesh://meineDaten/ABC:1234

leave(in Socket s, in Uri groupName, out Int error)
Der Leave-Call resultiert in dem Verlassen der Gruppe. Wird ein Leave-Call ausgeführt werden die Daten der angegeben Gruppe nicht weiter ausgetauscht oder anderen Benutzern zur Verfügung gestellt. Anders als bei einem Join-Call muss bei der Gruppenadresse kein Pfad angegeben werden, da die Gruppe nur über den Namen identifiziert wird.
Beispiel für den Aufbau einer Gruppenadresse:
mesh://ABC:1234

send(in Socket s, in Uri groupName, in Size msgLen, in Msg msgBuf, out Int error)
Generell wird dieser Aufruf dazu verwendet, einzelne Datenpakete an eine Gruppe zu senden. Bei dem Pull-Multicast soll das Senden jedoch vollkommen autonom von dem Benutzer ablaufen. Der Benutzer soll lediglich angeben, welche Daten in einer Gruppe getauscht werden sollen. Da für das Verteilen der Daten die Generierung von Metadaten notwendig ist und andererseits Daten nicht aktiv verteilt werden, sondern von Benutzern angefragt werden. Die Komplexität dieser Aufgabe soll vom Benutzer getrennt werden, so das, dem Applikation Entwickler die Möglichkeit geboten wird, durch geringe Anpassungen, auch eine alternative Technologie zu verwenden. Die Argumente msgLen und msgBuf vom Pull-Multicast Modul ignoriert. Benötigt wird der Pfad dessen Inhalt an andere Benutzer zur Verfügung gestellt werden soll. Mit dem ausführen des Send-Calls werden die Daten anderen Benutzern zur Verfügung gestellt.

Beispiel für den Aufbau einer Gruppenadresse:
mesh://meineDaten/ABC:1234

receive(in Socket s, out Uri groupName, out Size msgLen, out Msg msgBuf, out Int error)
Dieser Call dient zum Empfangen von Daten. Ist ein Speicherpfad angegeben, setzt das Technologiemodul die Daten automatisch zu Dateien zusammen. Handelt es sich jedoch um zeitkritische Daten wie z.B. bei Video-on-Demand, möchte der Benutzer nicht warten, bis die Dateien komplett heruntergeladen wurde. Zu diesem Zweck werden alle empfangenen Daten über den Receive-Call auch an den Benutzer weitergereicht. Beim Pull-Multicast hängt die Reihenfolge, in der die Pakete empfangen werden, vom Scheduling-Algorithmus ab. Es ist deshalb nötig das der Benutzer Meta-Informationen über das Empfangene Chunks erhält, wie z.B. die Chunk Nummer,

Länge des Chunks und einen Hash über den Daten des Chunks. Die Chunk Daten werden in den msgBuf geschrieben und die Länge des Chunks in die msgLen. Über die URI können weitere Informationen wie die Chunk-Nummer und der Hash ausgelesen werden.

Beispiel Uri eines Receive-Call:
mesh://ABC:1234/63/0F16K

3.3 Gruppenverwaltung

3.3.1 Daten für die Gruppenverwaltung

Für das Chunk-Trading werden Meta-Informationen und Peer-Listen benötigt die Auskunft über die Daten und den Netzwerkstatus geben.

Meta-Informationen Die Meta-Informationen setzten sich zusammen aus dem Namen der Datei und einer MD5 Prüfsumme um die Datenintegrität zu prüfen. Des weiteren müssen Informationen über die Chunks bereit gestellt werden. Dazu zählt, in wie viele Chunks die Datei aufgeteilt wurden ist, die Größe der Chunks und ein SHA1-Hash Code über den Inhalt der Chunks der genutzt wird um die Datenintegrität der Chunks zu prüfen. Handelt es sich um einen Ordner über mehrere Dateien so müssen diese Daten für jede einzelne Datei vorhanden sein, zusätzlich muss noch ein relativer Pfad für die einzelnen Dateien Angegeben sein. Diese Meta-Informationen werden automatisch vom Technologie Modul erstellt wenn ein Benutzer seine Daten veröffentlichen möchte.

Peer-Listen Zusätzlich zu den Meta-Informationen muss eine Liste geführt werden die den Netzwerkstatus der Peers pflegt. Die Liste muss Informationen darüber enthalten welche Teilnehmer am Datenaustausch teilnehmen und wie diese zu erreichen sind (IP-Adresse und Port).

3.3.2 Verwaltung der Informationen

Im Allgemeinen werden die Meta-Informationen und die Peer-Listen von einer zentralen Instanz zur Verfügung gestellt und gepflegt. Die Grafik 3.1 zeigt dies am einem Beispielnetzwerk für einen Video-on-Demand Dienst. Wie in der Zeichnung dargestellt sind zwei zentrale Instanzen, der Tracker und der Channel-Server, für die Verwaltung der Informationen zuständig. Der Tracker dient zum Überwachung des aktuellen Netzwerkzustandes. Zu jedem Channel pflegt der Server eine Liste von Peers die Daten über diesen Channel austauschen. Der Channel Server dient zum verwalten der Channels möchte ein Peer Informationen über einen Channel und über die

Daten die über diesen Channel ausgetauscht werden erhalten, so kann er eine anfrage an den Channel-Server stellen und die Meta-Daten herunterladen.

Für den Pull-Multicast verwenden wir eine DHT als dezentralen Speicherort für die Meta-Informationen und das Pflege der Peer-Listen. Die Verwendung einer DHT besitzt gegenüber einer Zentralen Lösung den Vorteil das, die Kosten für den Serverbetrieb und das Risiko eines Single-Point-of-Falure entfallen. Zusätzlich ist der Pull-Multicast dadurch leichter und flexibler einsetzbar, da kein Server aufgesetzt und instandgehalten werden muss. Jedoch erhöht sich der Managementaufwand für die Teilnehmer, da diese Last nicht mehr von dem Server übernommen wird.

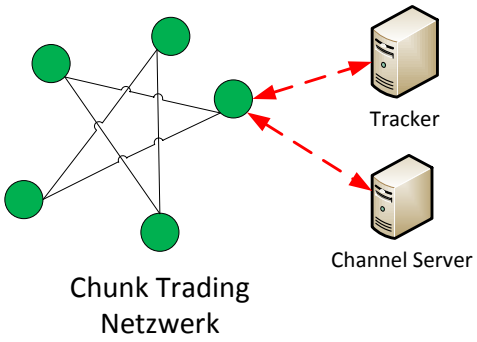

Abbildung 3.1: Server basiertes Chunk-Trading

Gruppenverwaltung über DHT

Die Zeichnung in Abbildung 3.2 zeigt den generellen Aufbau der Gruppenverwaltung über eine DHT. Um Daten auszutauschen bilden alle Benutzer ein Overlay-Netzwerk. In diesem Netzwerk werden die Meta-Informationen und Peer-Listen für die Gruppen gespeichert und gepflegt. Möchte ein Benutzer Daten veröffentlichen so werden die Meta-Informationen mit Hilfe des Routings in der DHT gespeichert. Um die Daten abzurufen wird über ein Lookup-Verfahren der Speicherort der Informationen und ein Einstiegspunkt für das Chunk-Trading ermittelt.

Die verteilte Datenspeicherung Um die Meta-Informationen in der DHT zu speichern wird ein SHA-1 Hash Code über den Gruppennamen erstellt. Anschließend wird eine Request-Nachricht erstellt und zu dem Knoten geroutet dessen ID die größte Übereinstimmung mit diesem Hashwert besitzt. Nachdem die Request-Nachricht den Knoten erreicht hat erstellt dieser eine Accept-Nachricht die, die IP-Adresse und Port des Knoten enthält. Die Accept-Nachricht wird

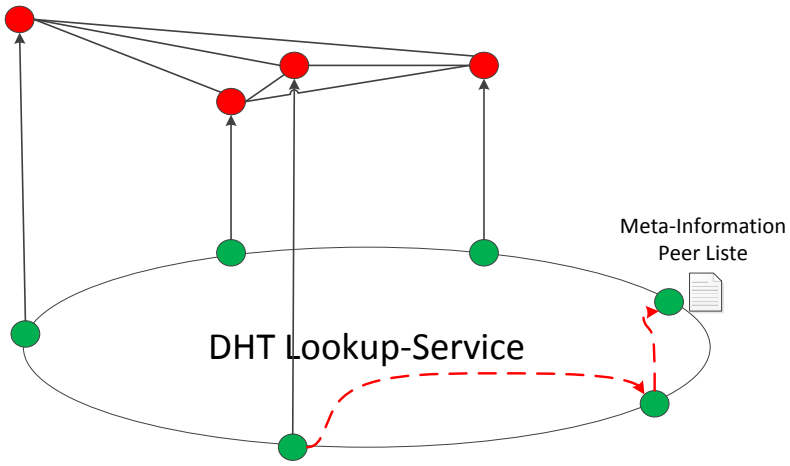

Abbildung 3.2: Gruppenverwaltung über DHT

anschließend an die Quelladresse zurück geroutet. Nach dem Empfangen der Accept-Nachricht können nun die Meta-Informationen direkt an den ermittelten Knoten gesendet werden. Eine weitere Aufgabe des Knoten der die Meta-Informationen hält ist das pflegen der Peer Liste. Zu jeder Gruppe die ein Knoten verwaltet wird eine Liste geführt die, die IP-Adressen und Ports aller Teilnehmer der Gruppe enthält. Jeder Knoten der die Meta-Informationen erhalten hat und der Knoten der diese veröffentlicht hat werden in der Liste gespeichert. Periodisch wird nun eine Alive-Nachricht an diese Knoten gesendet und mit einer AliveReply-Nachricht geantwortet. Sollte ein Knoten nicht antworten wird ein Zähler für diesen Knoten hochgezählt. Überschreitet der Zähler einen definierten Wert so wird der Knoten als nicht erreichbar betrachtet und aus der Liste entfernt. Um sicherzustellen das die gespeicherten Informationen nicht verloren gehen wenn der Knoten unerwartet ausfällt, werden die Daten auf einen weiteren Knoten Repliziert. Der Knoten auf dem die Daten Repliziert werden ist der Knoten dessen ID um eine Stelle mehr mit dem Hashcode übereinstimmt. Zum Beispiel, der Hashcode der Datei ist 1000, es existieren die Knoten 1000, 1100 und 0010, die Meta-Informationen sind auf dem Knoten 1000 gespeichert. Der Knoten 1000 erstellt nun eine Sicherheitskopie auf dem Knoten 1100 da dieser den längsten gemeinsamen Präfix +1 (LCP) mit dem Hashcode über den Gruppennamen aufweist. Sollte nun der Knoten der die Meta-Informationen gespeichert hat ausfallen kann der Knoten mit dem LCP+1 seine Aufgabe übernehmen und eine neue Kopie der Daten erstellen. Die Abbildung 3.3 zeigt den Nachrichtenaustausch und das Routing. Möchte ein Benutzer einer Gruppe beitreten so wird der Hash über den Gruppennamen gebildet und eine Subscribe-Nachricht an den Knoten

mit dem längsten gemeinsamen Präfix geroutet. Nachdem Empfangen dieser Nachricht antwortet der Knoten mit einer Reply-Nachricht die, die IP-Adresse und Port des Knoten enthält. Danach können die Meta-Informationen und Peer-Liste direkt ausgetauscht werden, zusätzlich wird ein Eintrag in der Peer-Liste für diesen Knoten erstellt.

Verbesserung der Lookup-Zeiten Um die Lookup-Zeiten zu verbessern, könnte zusätzlich über eine Publisch-Nachricht die Adresse des Knotens, der die Meta-Informationen und Peer-Liste pflegt, als Referenz in der DHT hinterlegt werden. Hierzu würde der Knoten der die Meta-Informationen veröffentlicht, nach einer erfolgreichen Veröffentlichung eine Publisch-Nachricht mit dem Hasch über den Gruppennamen und der IP-Adresse und Port des Speicherknotens erstellen und diese über einen Broadcast im Netz verteilen. Jeder Knoten der diese Nachricht empfängt erstellt daraufhin einen Eintrag für die Gruppe mit der IP-Adresse und Port des Speicherknotens als Referenz, so erhält jeder Knoten im Netzwerk Informationen darüber welche Gruppen existieren und wo Informationen über die Gruppen zu finden sind. Diese Listen werden über die Alive-Nachrichten wie oben beschrieben gepflegt. Sollte der Speicherknoten das Netzwerk verlassen übernimmt ein neuer Knoten seine Aufgabe und überschreibt den alten Eintrag mit einer neuen Publish-Nachricht. Knoten die der DHT neu beitreten können diese Gruppeninformationen über ihre Routing-Nachbarn erhalten.

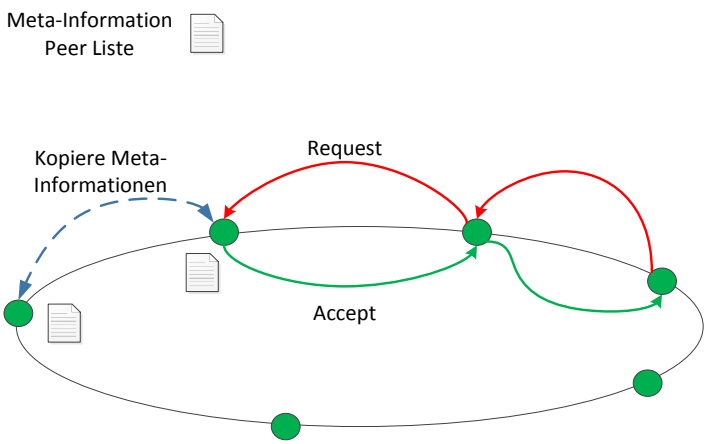

Abbildung 3.3: DHT Request Accept

4 Zusammenfassung und Ausblick

Inhalt der Ausarbeitung war die Implementierung der Common-API für das Pull-Multicast Modul und die Kozeptionierung einer Gruppenverwaltung für einen auf Chunk-Trading basierenden Pull-Multicast. Die hier vorgestellte Verwendung der Common-API erfüllt alle Anforderung des Pull-Multicast, so können über die Gruppenadressen und Argumente alle benötigten Informationen bezogen werden. Zusätzlich bietet die Common-API eine wohl definierte Schnittstelle so das Änderungen an der Technologie ohne Einfluss auf die Schnittstelle vorgenommen werden können. Die hier vorgestellt Gruppenverwaltung bietet eine dezentrale Lösung für das Verwalten und Speichern von Gruppeninformationen für den Pull-Multicast, welcher durch die Verwendung einer DHT auch bei großen Anzahlen von Benutzern gut skaliert und eine hohe Ausfallsicherheit bietet. Des weiteren bietet dieser Ansatz eine flexiblere und kostengünstigere Variante als eine Server basierte Lösung. Um dies zu beweisen sind ausführliche Messungen nötig sowie eine Gegenüberstellung mit einer Server basierten Lösung. Zusätzlich muss überprüft werden wie viel Nutzen eine Verbesserung des Lookup-Dienstes durch die Verteilung von Gruppeninformationen mittels Broadcast hat, da ein erhöhter Nachrichten und Managementaufwand für das Pflegen und Verteilen der Listen nötig ist.

Literaturverzeichnis

[Coh03] Bram Cohen. Incentives build robustness in bittorrent, 2003.

[HLR08] Xiaojun Hei, Yong Liu, and K.W. Ross. Iptv over p2p streaming networks: the mesh-pull approach. *Communications Magazine, IEEE*, 46(2):86 –92, february 2008.

[JX] Victor O.K. Li Jialing Xu. Request-driven swarming scheme for P2P data streaming. page 1410.

[MCSW10] Sebastian Meiling, Dominik Charousset, Thomas C. Schmidt, and Matthias Wählisch. System-assisted Service Evolution for a Future Internet – The HAMcast Approach to Pervasive Multicast. In *Proc. of IEEE GLOBECOM 2010, Workshop MCS 2010*, pages 913–917, Piscataway, NJ, USA, Dec. 2010. IEEE Press.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of the 1st Int. Workshop on Peer-to Peer Systems (IPTPS '02)*, pages 53–65, Cambridge, MA, USA, 2002.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *LNCS*, pages 329–350, Berlin Heidelberg, November 2001. Springer–Verlag.

[SMLN+03] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[WSV12] Matthias Wählisch, Thomas C. Schmidt, and Stig Venaas. A Common API for Transparent Hybrid Multicast. IRTF Internet Draft – work in progress 04, IRTF, January 2012.