Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sebastian Zagaria

Visualisierungs- und Monitoring-Software für eine hybride Multicast Architektur

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Zagaria

Visualisierungs- und Monitoring-Software für eine hybride Multicast Architektur

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
Department Informatik
in der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas C. Schmidt
Zweitgutachter : Prof. Dr.-Ing. Franz Korf

Abgegeben am 7. September 2010

Sebastian Zagaria

Thema der Bachelorarbeit

Visualisierungs- und Monitoring-Software für eine hybride Multicast Architektur

Stichworte

IP-Multicast, hybride Multicast Architekturen, HAMcast, Traceroute, Monitoring, Visualisierung, Netzwerk-Topologie

Kurzzusammenfassung

Diese Bachelorarbeit befasst sich mit der Entwicklung und der Implementierung einer Visualisierungs- und Monitoring-Software für eine hybride Multicast-Architektur. Es werden die Grundlagen von IP- und Application-Layer-Multicast dargelegt und existierende hybride Multicast Ansätze präsentiert. Verfahren zum Auffinden von Multicast Typologien und deren Visualisierung werden im Anschluss besprochen. Anschließend wird die HAMcast Architektur und API, sowie die Implementierung und Weiterentwicklung der Visualisierungs- und Monitoring Software, vorgestellt.

Sebastian Zagaria

Title of the paper

Visualization and monitoring Software for a hybrid multicast architekture

Keywords

IP-Multicast, hybrid multicast Architecture, HAMcast, Traceroute, monitoring, visualization, networktopology

Abstract

This bachelor thesis deals with the development and implementation of a visualization and monitoring software for a hybrid multicast architecture. First, the basics for IP and application layer multicast will be presented. Second, an overview on existing hybrid multicast approaches will be given, and we discuss several approaches to discover the multicast topology and the visualization for such data. As core contribution, the HAMcast-Architecture and API, as well as the monitoring concept, the implementation and further development for a specific visualization and monitoring software will be elaborated in detail.

Inhaltsverzeichnis

1 Einleitung

1.1 Einleitung

Für die Wartung und den Aufbau von komplexen Netzwerken ist es erforderlich, dass Software-Werkzeuge zur Verfügung stehen, die einen Überblick über den Aufbau und Zustand eines Netzwerks bieten. Die Informationen, die durch solche Werkzeuge gewonnen werden, können Netzwerk-Administratoren dabei helfen, das Netzwerk zu verwalten, zu überwachen und auftretende Probleme zu erkennen.

Diese Bachelorarbeit behandelt die Entwicklung und Implementierung einer Visualisierung- und Monitoring-Software für eine hybride Multicast-Architektur. Die Aufgabe einer Visualisierungs- und Monitoring-Software ist es, die Netzwerktopologie zu ermitteln und Informationen über den Zustand des Netzwerks bereit zu stellen. Die Visualisierung dieser Daten bietet dem Benutzer eine verständliche Darstellung der Netzwerk-Topologie und die Möglichkeit auftretende Probleme zu identifizieren.

Die im Rahmen dieser Arbeit entwickelte Software basiert auf dem *Hybrid Adaptive Mobile Multicast* (HAMcast) [20] Projekt und dem QT UI-Framework.HAMcast stellt einen universellen Multicast Dienst bereit. HAMcast implementiert einen hybriden Multicastansatz, der es ermöglicht, Multicast-Domains verschiedener Multicast-Technologien miteinander zu verbinden. Dieser Multicast-Dienst kann über die *Common Multicast API* [28] zur gruppenkommunikation genutzt werden. Die Funktionalität der API wird durch eine Middleware realisiert, die dem Benutzer eine Schnittstelle zur Verfügung, die es ermöglicht die Funktionalitäten der HAMcast-Architektur nutzen zu können. Die Visualisierungs- und Monitoring-Software verwendet diese Benutzerschnittstelle um Topologie-Informationen über die HAMcast-Hosts eines Netzwerks zu ermitteln.

Das QT UI-Framework stellt Bibliotheken zur Verfügung, z.B. für das Erstellen von grafischen Oberflächen, der Netzwerk-Programmierung und generische Datentypen. Bei der Visialisierungs- und Monitoring-Software wird das QT UI-Framework eingesetzt um die Benutzeroberfläche zu erstellen und die Informationen, die über die *Common Multicast API* ermittelt wurden, zu visualisieren.

1.2 Problemstellung

Ziel der vorliegenden Arbeit ist es eine Visualisierungs- und Monitoring-Software zu entwickeln die, die Netzwerk-Topologie eines HAMcast Netzwerks mit Hilfe der HAMcast API ermittelt und visuell darstellt. Die Software soll sowohl unter Linux, als auch Windows Systemen lauffähig sein. Die Software soll ein Netzwerkgraphen ermitteln und zeichnen, sowie zusätzliche Informationen über Hosts anzeigen, wie z.B. alle beigetretenen Multicastgruppen oder die verwendete Verbindungstechnologie.

1.3 Übersicht

Die vorliegende Bachelorarbeit gliedert sich wie folgt. In dem Kapitel Grundlagen 2 wird zuerst auf die generelle Funktionsweise von IP- und Application-Layer-Multicast eingegangen. Anschließend wird eine Übersicht über existierende hybride Multicast Ansätze dargestellt. Danach wird anhand von zwei Beispielen das Ermitteln einer Multicast-Netzwerk-Topologie vorgestellt. Zum Schluss des Kapitels werden zwei Netzwerk-Visualisierungs-Programme vorgestellt. Im dem Kapitel HAMcast 3 wird die HAMcast-Architektur und API vorgestellt. In dem Kapitel Implementation 4 wird das Konzept der Visualisierungs- und Monitoring-Software und deren Implementation genauer behandelt. Zum Schluss werden die möglichen Erweiterungen und die Weiterentwicklung der Software besprochen.

2 Grundlagen

In diesem Kapitel werden die allgemeinen Voraussetzungen beschrieben, die für das Verständnis der entwickelten Visualisierungs- und Monitoring-Software benötigt werden. Im ersten Abschnitt des Kapitels wird die Funktionsweise und Problematik von IP-Multicast beschrieben. Danach folgt die Beschreibung des Application-Layer-Multicast, sowie eine Beschreibung existierender Verfahren für hybride Multicast-Architekturen. Die letzten beiden Abschnitte befassen sich mit Verfahren zur Topologie Auffindung von Multicast-Netzwerken und deren Visualisierung.

2.1 IP Multicast

IP-Multicast [6] ist ein verbindungsloser Übertragungsdienst, welcher Daten an eine Untergruppe von Hosts in einem Netzwerk überträgt. Multicast ist ein effizientes Vorgehen bei der Verteilung von Echtzeitdaten an eine Vielzahl von Empfängern. Somit ist es besonders gut für Anwendungen wie z.B. Audio/Video Streaming, Multiplayer Games und Videoconferencing geeignet.

Die beiden Abbildungen 2.1 und 2.2 zeigen ein Szenario, in dem ein Sender Daten an eine Gruppe von Empfängern versendet. Die Abbildungen verdeutlichen die Unterschiede zwischen IP-Multicast und Unicast aufzeigen. Wie in Abbildung 2.1 dargestellt, möchte der Sender S eine Nachricht an eine Gruppe von Empfängern senden. Die Daten werden per Unicast übertragen. Um die Nachricht an alle Empfänger senden zu können, müssen diese repliziert und nacheinander, also zeitverzögert, an die Empfänger gesendet werden. Durch das Anwenden von Multicast kann diese Replikationslast deutlich reduziert werden. Abbildung 2.2 zeigt das gleiche Szenario mit IP-Multicast. Der Sender S sendet genau eine Multicast-Nachricht an eine Gruppe von Empfängern. Die Nachricht wird von den Routern dann an den benötigten Stellen repliziert und weitergeleitet. Durch eine gruppenspezifische Adresse muss der Sender kein Wissen über Empfänger besitzen. Das Senden an unbekannte Teilnehmer ist durch eine solche gruppenspezifische Adresse möglich. Ein Aktualisieren von Empfängerlisten beim Sender, da die IP-Adressen der Empfänger sich über die Zeit ändern können, muss ebenfalls nicht mehr berücksichtigt werden. Die Replikationslast des Senders wird durch Multicast reduziert und in das Netzwerk verlagert.

Beim Multicast kann zwischen zwei grundsätzlichen Verfahren unterschieden werden: Any-Source-Multicast [6] (ASM) und Source-Specific-Multicast [10] (SSM). Any-Source-Multicastgruppen haben die Eigenschaft, dass jeder Host einer Gruppe (G) Multicastpakete an die Gruppe senden kann. Als Sender ist es nicht nötig, aber möglich, Teil

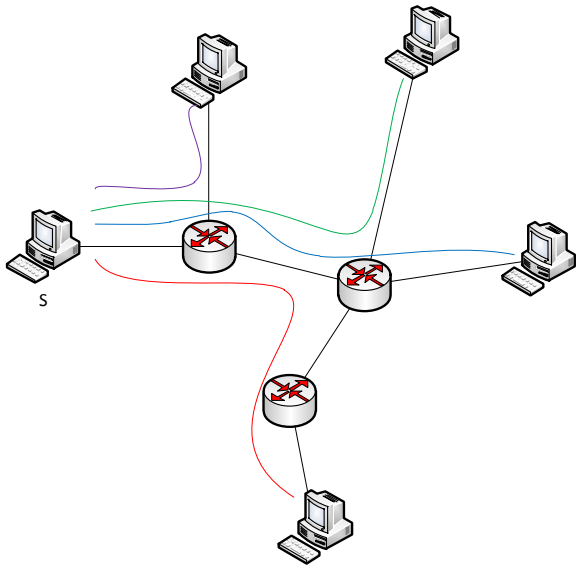

Abbildung 2.1: Versenden einer Nachricht an eine Gruppe von Empfängern über Unicast

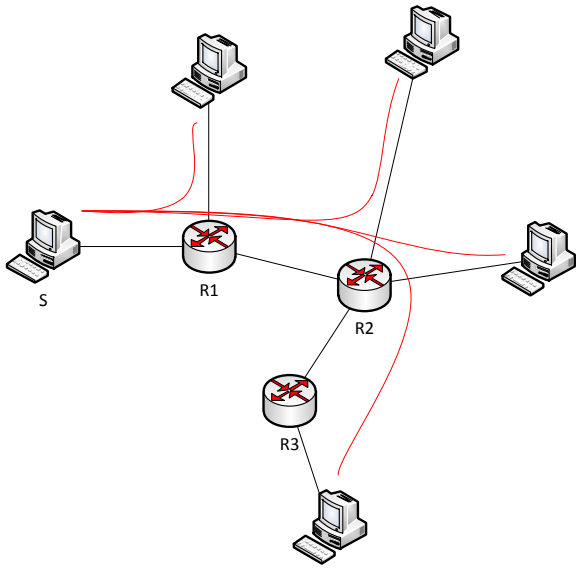

Abbildung 2.2: Versenden einer Nachricht an eine Gruppe von Empfängern über Multicast

der Multicastgruppe zu sein. Somit unterstützt ASM sowohl one-to-many als auch many-to-many Gruppenkommunikation. One-to-many bedeutet, dass es einen Sender und eine Vielzahl von Empfängern gibt. Many-to-many Gruppenkommunikation bedeutet, dass es eine Vielzahl von Quellen und eine Vielzahl von Empfängern geben kann. Source-Specific-Multicast hat die Eigenschaft, das es nur einen spezifizierten Sender für einen Multicast-Channel gibt. Ein Multicast-Channel wird durch ein Paar, bestehend aus Sender und Gruppe (S,G), gebildet. Somit unterstützt SSM nur eine one-to-many Gruppenkommunikation. Pro Gruppe können aber durchaus mehrere solcher Multicast-Channels existieren z.B. für Sender S1 die Gruppe (S1,G) und für Sender S2 die Gruppe (S2,G).

Multicast beruht auf dem Publish/Subscribe Prinzip. Es existieren ein oder mehrere Publisher, die Daten an eine spezifische Gruppe verteilen. Der Publisher besitzt dabei kein Wissen darüber, welche und ob es überhaupt Empfänger gibt, die Interesse an diesen Daten haben. Ein Subscriber bekundet sein Interesse an den Daten einer spezifischen Gruppe, ohne zu wissen wer der Publisher ist und ob überhaupt einer existiert. Aus diesem Grund muss es einen Mechanismus geben, über den sich teilnehmende Router und Hosts austauschen können. Informationen über Gruppen und ihre Zustände, sowie das Bekunden von Interesse und Desinteresse an einer Gruppe, müssen bereitgestellt werden. Aus diesem Grund wird ein Gruppenmanagement-Protokoll benötigt, das diese Aufgaben erfüllen kann. Existierende Gruppenmanagement-Protokolle sind u.a. Internet Group Management Protokoll [3] (IGMP) für IP-Version 4 (IPv4) Systeme (Hosts und Router) und Multicast Listener Discovery [25] (MLD) für IP-Version 6 (IPv6) Systeme (Hosts und Router).

Das effiziente Weiterleiten von gruppenspezifischen Multicastpaketen, das Forwarden, sowie das Auffinden von optimalen Pfaden und der Aufbau von Multicast-Bäumen, werden durch die Multicast-Routing-Protokolle realisiert. Multicast- Routing-Protokolle lassen sich in zwei Typen von Protokollen aufteilen, Dense Mode und Sparse Mode. Dense Mode ist ein Routing-Protokoll das unter der Annahme arbeitet, dass es eine Vielzahl von Empfängern gibt. Das heißt Multicastdaten werden an alle Subnetzwerke zugestellt, es sei denn, es wurde explizit angegeben das kein Interesse an den Daten besteht. Häufig angewandte Dense Mode Routing-Protokolle sind Distance Vector Multicast Routing Protocol [29] (DVMRP) und Protocol Independent Multicast - Dense Mode [1](PIM-DM). Bei Sparse Mode Routing-Protokollen wird angenommen, dass wenige Multicast-Empfänger existieren d.h. Multicastdaten werden nur zugestellt, wenn sie explizit beantragt wurden. Eines der am weit verbreitetsten Routing-Protokolle dieser Art ist Protocol Independent Multicast - Sparse Mode [8](PIM-SM). Das Paket-Forwarding und das Finden der optimalen Pfade wird bei DVRMP und PIM-DM über einen Minimal Spanning Tree realisiert, welcher den kürzesten Pfad ausgehend von der aktuellen Quelle ermittelt. Bei PIM-SM und anderen Sparse Mode Protokollen wird das Forwarding über einen Shortest-Path-Tree [8](SPT) realisiert, ausgehend von einer bekannten Instanz, dem Rendezvous Point, werden die kürzesten Pfade ermittelt. Bei DVRMP wird der Multicast-Traffic an alle Subnetze weitergeleitet, anschließend werden „prune messages“ verschickt um die Pfade, welche nicht zu einen Gruppenmitglied führen, aus dem Spanning Tree zu entfernen. PIM-SM nutzt einen unidirektionalen Baum, bei dem Multicastpakete zuerst an einen Rendezvous Point geschickt werden und anschließend von diesem aus, über einen Multicast Shortest-Path-Tree, an die Empfänger weitergeleitet wird.

Eine allgemeine Verbreitung und Bereitstellung von IP-Multicast durch Internet Service

Provider (ISP) ist zum jetzigen Zeitpunkt noch nicht umgesetzt worden. Dies hat zur Folge, dass eine allgemeine Verfügbarkeit von IP-Multicast im Internet nicht gegeben ist [7]. Die Gründe dafür sind unter anderem, das Fehlen von Autorisierungsmechanismen für das Erstellen von Gruppen, dem Beitreten oder das Senden an eine Gruppe. Zudem stellt IPv4-Multicast keine Mechanismen zur Verfügung, die das Problem der Allokation von Multicast-Gruppenadressen lösen, z.B. das Feststellen, ob eine Gruppe bereits existiert oder ob diese Gruppe überhaupt noch in Verwendung ist. Des Weiteren mangelt es an Netzwerk-Managementprogrammen und Funktionen, die Multicast-Router und Gruppen vor Angriffen schützen. Die hohen Kosten bei der Platzierung von multicastfähigen Routern für einen internetweiten Einsatz, stellen ebenfalls ein Problem bei der Verbreitung von IP-Multicast dar.

Es existieren Ansätze wie Multicast Backbone (Mbone) [2], die Multicast-Subnetzwerke über IP-Tunnel miteinander verbinden. Da Mbone aber nicht die Probleme der Authentifizierung und Sicherheit von Multicast-Netzwerken lösen kann, bleiben die generellen Probleme, die von IP-Multicast, weiterhin bestehen. Wegen den vorher bereits angesprochene Problemen von IP-Multicast, wurden Multicast Mechanismen auf Anwendungsebenen entwickelt.

2.2 Application Layer Multicast (ALM)

Die als Application-Layer-Multicast (ALM) [11] bezeichneten Protokolle bieten einen Dienst an, der es End-Hosts erlaubt, an einem internetweiten Multicast-Netzwerk teil zu nehmen. Die Skalierbarkeit von Anwendungen, wie z.B. Audio/Video Streaming, Filesharing, Videoconferencing oder Multiplayer Games, kann durch einen solchen Application-Layer-Multicast deutlich verbessert werden.

Die Nachteile des ALM gegenüber dem IP-Multicast sind, dass die Replikation von Paketen Aufgabe der Hosts ist und die zeitliche Verzögerung beim Versenden und Empfangen solcher Pakete zunimmt. Des Weiteren findet das Routing auf der Anwendungsebene statt, weshalb die Wege nicht optimal sind. Dies führt dazu, dass die Pakete mit einer höheren Verzögerungzeit, als bei IP-Multicast, übertragen werden. Daraus ergibt sich ein ineffizienteres Netzwerk als bei IP-Multicast. Einen Ausgleich dieser Nachteile schafft der ALM aber durch eine einfache Verteilung im Internet, da jeder Host am Netzwerk teilnehmen kann. Weitere Vorteile sind die Wartbarkeit, die Möglichkeit einer leichteren Aktualisierung des Systems und das Zuschneiden eines ALMs für eine bestimmte Applikation. Der übliche Ansatz einer ALM ist es, ein virtuelles Netzwerk aus Unicast-Verbindungen zu erzeugen, das Overlay. Auf diesem Overlay-Netzwerk können dann Multicast-Bäume erstellt werden. Im Gegensatz zum IP-Multicast müssen die Funktionalitäten für das Gruppenmanagement und das Weiterleiten der Daten von den End-Hosts übernommen werden, was eine zusätzliche Belastung der Hosts zur Folge hat.

Es kann zwischen zwei Arten von ALM-Systemen unterschieden werden, Proxy-based und End-system-based ALMs. Die Abbildungen 2.3 und 2.4 zeigen den beispielhaften Aufbau solcher Systeme. Die schwarzen Linien sind die Netzwerk-Verbindungen zwischen den Systemen und die roten Linien stellen die Overlay-Verbindungen dar. Proxy-based ALMs sind Overlay-Multicast-Netzwerke, bei denen durch Server/Proxys

Multicast-Funktionalität zur Verfügung gestellt wird. Die Server/Proxys können dabei die Vorteile von separierten Multicast-Netzwerken nutzen, indem sie als Repräsentant für ein solches Netzwerk eingesetzt werden, wodurch die Effizienz für ein Subnetzwerk erhöht werden kann. Die Verfügbarkeit einer Proxi-based ALM ist im Vergleich zu einer End-System ALM höher, da Server generell eine längere Lebenszeit im Bezug auf die Erreichbarkeit im Internet haben. Durch das Auslagern der Multicast- Funktionalität auf die Server/Proxys, wird die Verantwortlichkeit von Gruppenmanagement und Datenweiterleitung von den Server/Proxys übernommen. Die Komplexität der Anwendungen, die einen solchen Multicast nutzen möchten, wird dadurch reduziert. Der Nachteil von Proxy-basierten-Systemen ist, dass sie weniger flexibel auf eine bestimmte Anwendung angepasst werden können, da sie eine generelle Multicast-Funktionalität bereitstellen. Die Verteilung von Servern/Proxys und die damit verbunden Kosten für eine hohe Erreichbarkeit des Dienstes, sind ebenfalls zu berücksichtigen. End-System ALM-Protokolle bieten eine wesentlich flexiblere Anpassung an Anwendungen und eine leichte Verteilung des Dienstes an, da keine Server benötigt werden. Die Skalierbarkeit solcher Systeme kann bei großen Netzwerken und einer großen Anzahl an Multicast-Gruppen, je nach Overlay- und Multicast-Protokoll, deutlich schlechter sein als bei Proxy-based Ansätzen. Das liegt daran, dass End-System-Hosts über eine geringere Bandbreite verfügen und die Verantwortlichkeit von Gruppenmanagement und Forwarding von den End-Systemen übernommen werden müssen.

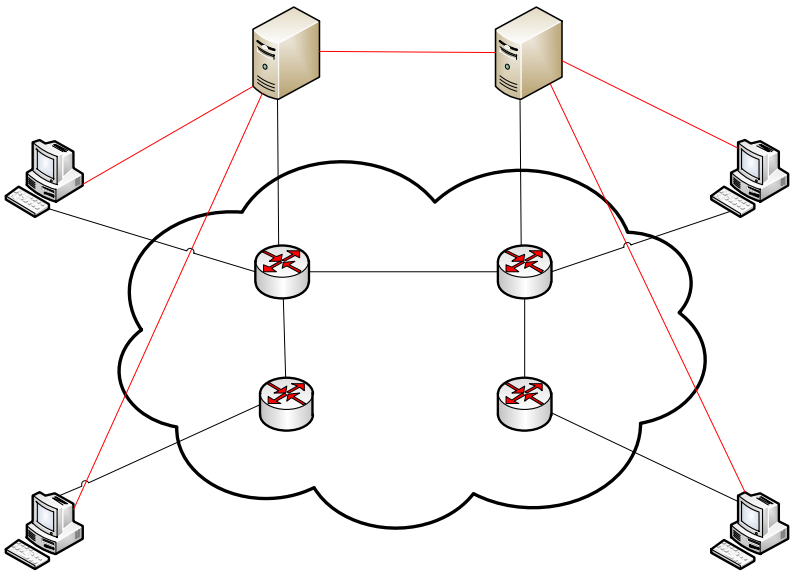

Abbildung 2.3: Proxy-based-System

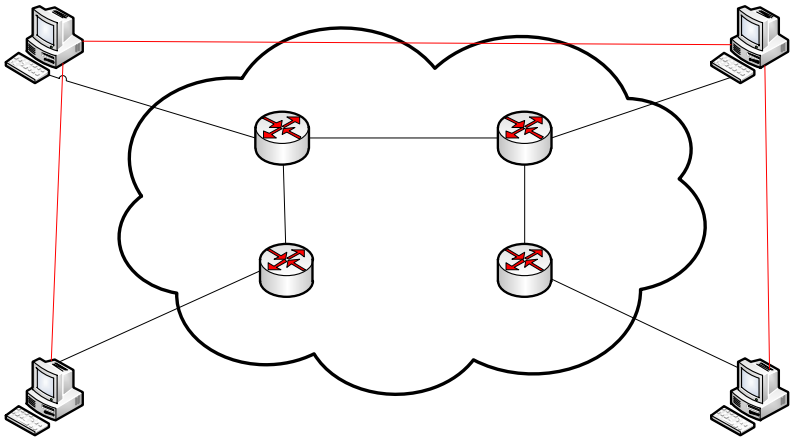

Abbildung 2.4: End-System-based

2.3 Hybrider Multicast

Die Idee einer hybriden Multicast-Architektur ist es isolierte Multicast-Netzwerke, so genannte „walled gardens“ und Multicast-Inseln, über deren Grenzen hinaus miteinander zu verbinden. Das Ziel ist es, einen Multicast Dienst zur Verfügung zu stellen, der die Vorteile von IP-Multicast mit denen von Application-Layer-Multicast kombiniert.

Geschlossene Multicast-Netzwerke werden als Inseln bezeichnet. Eine Insel ist eine Netzwerk Domäne beliebiger Größe, aber nur einer Multicast-Technologie. Die Grenze einer Insel ist der Bereich, indem sich eine Multicastnachricht in einer Technologie verteilen lässt.

Hybride Multicast-Architekturen verfügen generell über zwei Ebenen. In der unteren Ebene befinden sich die Multicast-Inseln. In der darüber liegenden zweiten Ebene werden die Multicast-Inseln über Edge Nodes, die Teil eines globalen Application-Layer-Multicasts sind, miteinander verbunden.

Das Beispiel 2.5 zeigt eine hybride Multicast-Architektur, in der drei Inseln miteinander verbunden sind. Die Hosts H5 (IP Multicast), H3 (IP Multicast) und H7 (ALM) dienen dabei als Edge Nodes. Sie verbinden die einzelnen Inseln über einen globalen Application-Layer-Multicast. Architekturen, die einen solchen oder ähnlichen Ansatz vertreten, sind u.a. „Scalable and Backbone Topology-Aware Hybrid Multicast“ (SHM) [16], „Universal IP multicast delivery“ (UM) [30], „Island Multicast: Combining IP Multicast With Overlay Data Distribution“ (IM) [15] und die „Hybrid Shared Tree“ (HST) Architektur [26] [27].

SHM: Scalable and Backbone Topology-Aware Hybrid Multicast

SHM stellt ein Verfahren vor, das IP-Multicast-Inseln und Application-Layer-Multicast-Inseln über ein zentralisiertes Application-Layer-Multicast Netzwerk verbindet. Wie in Abbildung 2.6 zu sehen ist, unterscheidet SHM [16] zwischen zwei Arten von Inseln, Node-Domain und IP-Multicast-Island. Node-Domain sind Inseln bei denen Hosts über

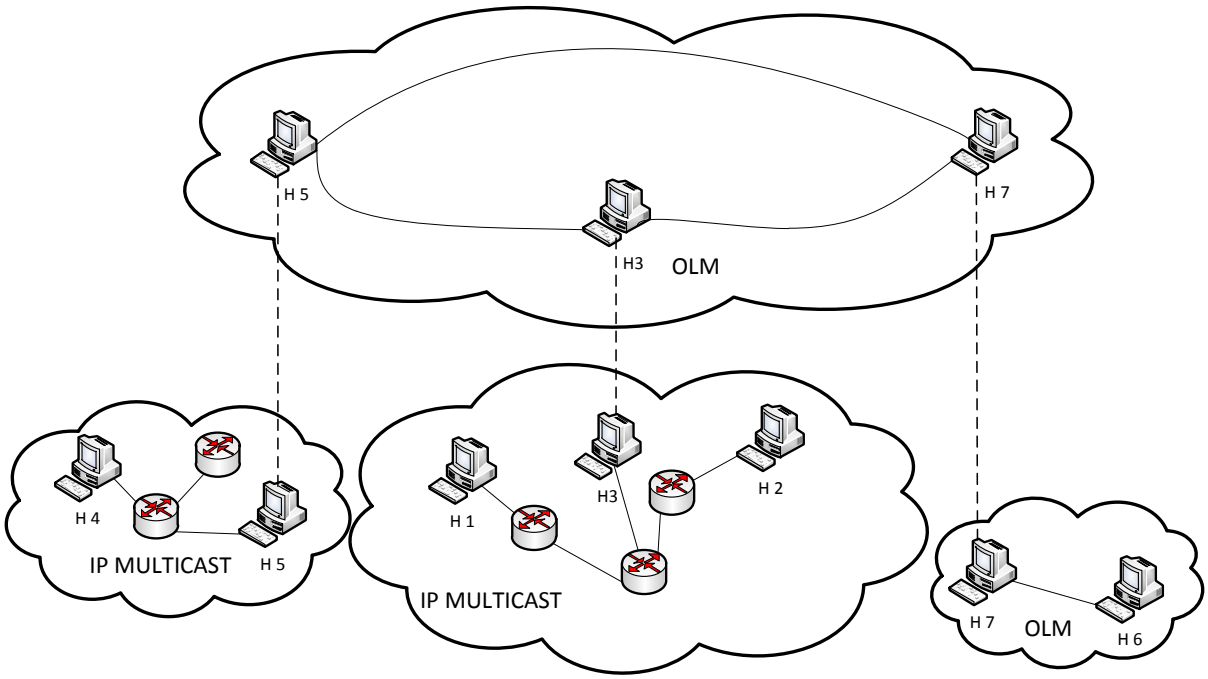

Abbildung 2.5: Beispiel für eine hybride Multicast-Architektur

einen Application-Layer-Multicast miteinander verbunden sind. IP-Multicast-Island sind Inseln, bei denen eine Kommunikation über IP-Multicast möglich ist. Jede Insel verfügt über einen Domain-Agent. Die Domain-Agents (DA) dienen dazu, die Inseln über ein zentralisiertes Application-Layer-Multicast Netzwerk zu verbinden. Das zentralisierte Application-Layer-Multicast Netzwerk und die Domain-Agents werden von einem globalen Rendezvous Point (RP) koordiniert und überwacht. Der RP ist ein für alle Teilnehmer bekannter Host. Er hält alle Informationen über die Netzwerk-Topologie bereit und verwaltet alle Domain-Agents. Das Gruppenmanagement wird von dem RP verwaltet. Möchte ein Host einer Gruppe beitreten, diese verlassen oder eine Gruppe erstellen, so muss der RP kontaktiert werden. Die Entscheidung, ob ein Host Domain-Agent für eine Multicast-Island ist, wird von RP getroffen.

Island Multicast: Combining IP Multicast With Overlay Data Distribution

Die Idee von Island-Multicast [15] ist es, IP-Multicast und Application-Layer-Multicast miteinander zu kombinieren, um so IP-Multicast-Inseln miteinander verbinden zu können.

Die Abbildung 2.7 zeigt ein Beispiel für Island-Multicast. Hosts die eine Verbindung zwischen Multicast-Inseln herstellen, werden als Bridge-Nodes bezeichnet. Es gibt zwei Arten von Bridge-Nodes, die Egress-Bride-Nodes und die Ingresss-Bridge-Nodes. Egress-Bridge-Nodes sind Hosts, die Datenpakete an andere Multicast-Inseln senden und Ingress-Bridge-Nodes sind Hosts, die Datenpakete von anderen Multicast-Inseln empfangen. Island-Multicast gibt es in zwei Varianten, die jeweils für spezielle Arten von Multicast-Anwendungen entwickelt wurden. Centralized Island Multicast (CIM) ist

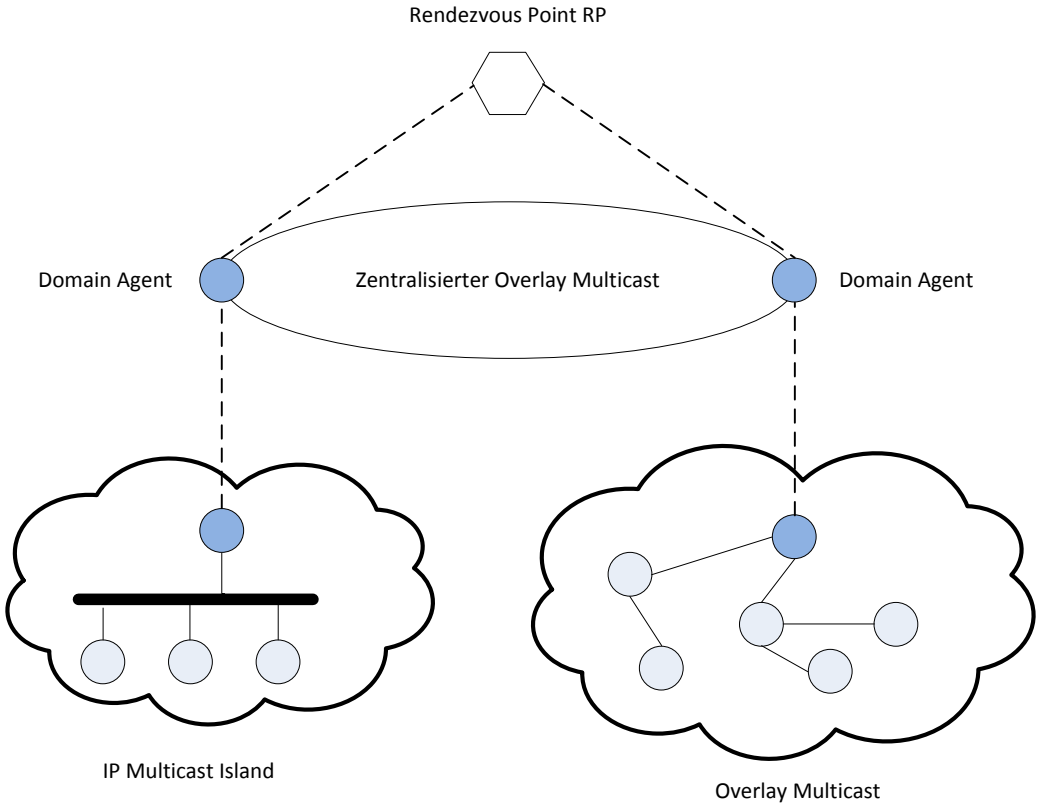

Abbildung 2.6: SHM Beispiel

für Anwendungen mit einer kleinen Multicast-Gruppengröße und hoher Bandbreiten-Anforderung geeignet. Ein zentraler Server erzeugt und verwaltet einen Verteilungsbaum, der zur Datenübertragung dient. Der Verteilungsbaum wird anhand von globalen Informationen generiert. Jeder bride-node einer Multicast-Island verwendet diesen Verteilungsbaum, um die Hosts zu erfahren, an die er Daten weiterleiten muss. Distributet Island Multicast (DIM) ist das zweite Protokoll. Es ist für Anwendungen geeignet, die über eine große Anzahl von Benutzern verfügen. DIM besteht aus zwei Schichten. In der ersten Schicht werden alle Multicast-Inseln über ein auf Unicast basiertem Overlay-Baum miteinander verbunden. In der zweite Schicht befinden sich die Multicast-Inseln. Die interne Verteilung von Daten wird bei den Multicast-Inseln über IP-Multicast durchgeführt. Jede Multicast-Insel wählt einen Anführer aus. Die Anführer sind über ein Overlay miteinander verbunden. Die Inseln werden in Paare aufgeteilt, die Anführer zweier Inseln wählen dann zwei Hosts aus, wobei jeweils einer zum Ingress-Bridge-Node und einer zum Egress-Bridge-Node bestimmt wird.

Universal IP multicast delivery

Das Universal Multicast Verfahren (UM) [30] verbindet, wie auch bereits Island Multicast, die Multicast-Inseln über einen ALM. Bei UM werden in jeder Multicast-Insel ein oder mehrere Designated Members (DM) gewählt. DMs sind Hosts die dynamisch Unicast-Tunnel zwischen den Multicast-Inseln herstellen, um diese miteinander zu verbinden. Multicast-Datenpakete werden innerhalb der Multicast-Inseln über IP-Multicast

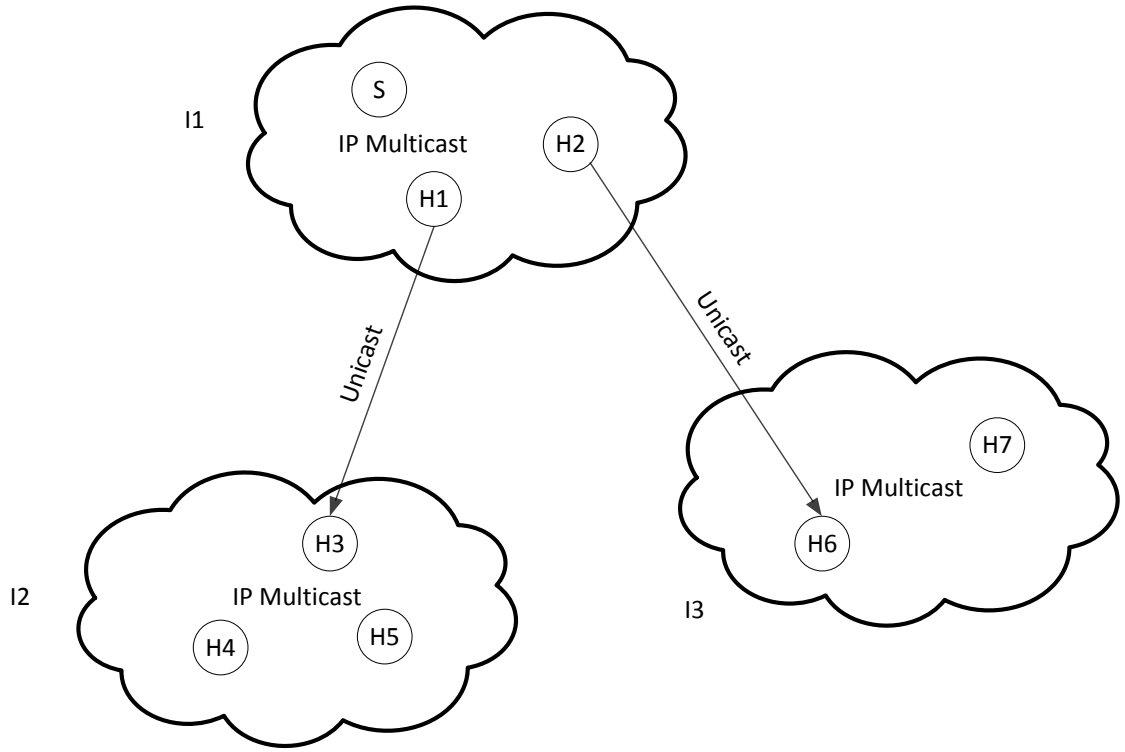

Abbildung 2.7: Island Multicast Beispiel Quelle [15]

gesendet und durch die DMs über Unicast-Tunnel an die anderen Multicast-Inseln übermittelt. Die DMs sind dafür verantwortlich, über ein Application-Layer-Multicast ankommende und ausgehende Daten an andere Multicast-Inseln weiterzuleiten. Für jede Gruppe gibt es einen Rendezvous-Point, den Universal Multicast Rendezvous-Point (UMRP). Dieser hat die Aufgabe, neuen DMs die Verbindung in das globale Overlay zu ermöglichen. Der UMRP besitzt protokollspezifische Informationen z.B welches Application-Layer-Multicast-Protokoll von der Gruppe verwendet wird. Sollte der UMRP ausfallen, ist es neuen DMs nicht mehr möglich dem Netzwerk beizutreten. Das bereits vorhandene Multicast-Netzwerk bleibt trotzdem noch funktionsfähig. Die Abbildung 2.8 zeigt beispielhaft den Aufbau eines Universal-Multicast-Netzwerks.

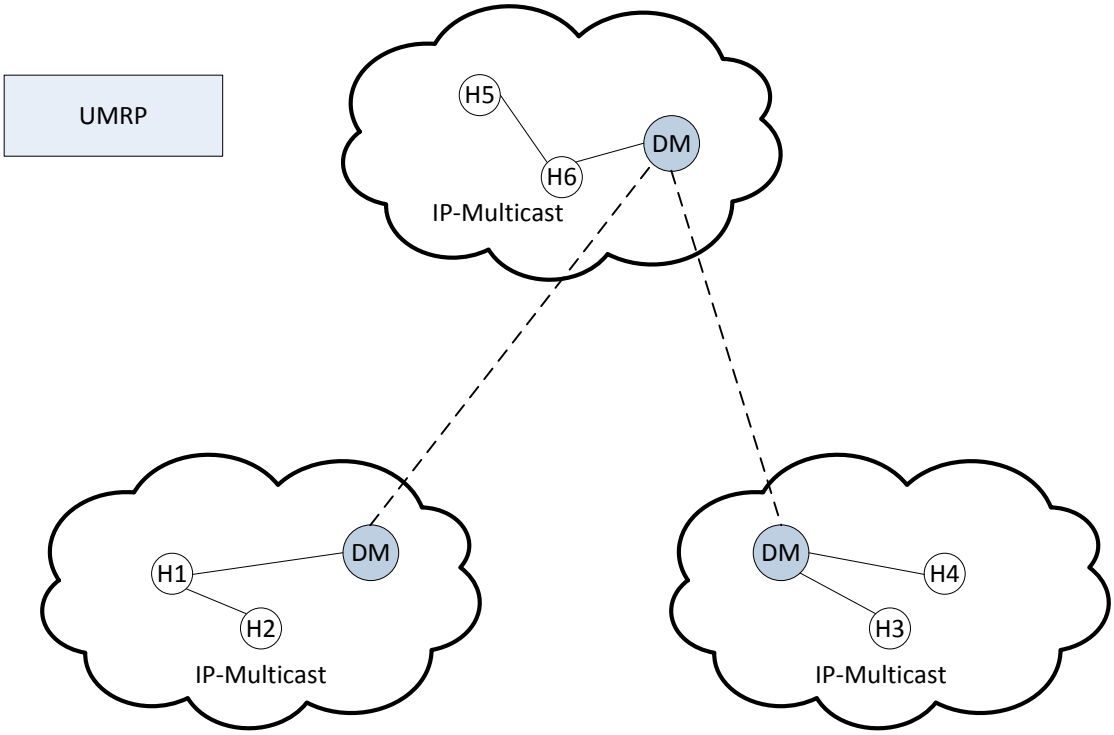

Abbildung 2.8: Universal Multicast Netzwerk

2.4 Ermitteln der Multicast-Topologie

Multicast-Topologie Informationen können Netzwerk Administratoren dabei helfen, das Netzwerk zu konfigurieren und mögliche Forwarding Probleme zu identifizieren. Um die Multicast-Topologie Informationen bereitstellen zu können, müssen Werkzeuge wie der Unicast-Traceroute [19] für Multicast entwickelt werden.

Das Unicast-Traceroute [19] Verfahren nutzt eine bereits vorhandene Funktionalität (ICMP) von Routern aus, die es ihm ermöglicht, von allen Routern unterstützt zu werden. Dieses Traceroute Verfahren kann jedoch bei Multicast-Netzwerken nicht angewandt werden, da als Reaktion auf Multicastpakete mit zu geringer TTL keine ICMP Pakete als Antwort generiert werden [18]. Dies ist nötig, da sonst der Sender eines Traceroute, wenn er Teil eines großen Multicast-Netzwerks ist, mit Nachrichten überflutet werden würde. Aus diesem Grund wird ein anderes Verfahren benötigt, um die Traceroute Funktionalität auch für Multicast-Netzwerke nutzen zu können. Eine Erweiterung der Funktionalitäten von Multicast-Routern ist dafür erforderlich, da die durch Router zur Verfügung gestellten Informationen nicht ausreichen, um die Multicast-Topologie zu bestimmen.

Die in diesem Kapitel vorgestellten Algorithmen ermitteln die Multicast-Topologie basierend auf den Multicast-Forwarding-States. Man kann generell zwischen zwei Ansätzen unterscheiden, dem Empfänger zur Quelle und dem Quelle zum Empfänger Verfahren.

Empfänger zur Quelle

Um die Netzwerktopologie zu ermitteln, wird bei diesem Ansatz der Pfad jedes Empfängers bis hin zur Quelle verfolgt und aus der Summe der Informationen der Baum gebildet. Um einen solchen Ansatz durchführen zu können, ist es nötig alle Empfänger der Multicast-Gruppe zu kennen.

Quelle zum Empfänger

Bei diesem Ansatz wird zum Auffinden der Baum-Topologie die Quelle als Ausgangspunkt gewählt und der Traceroute Algorithmus bis zu den Empfängern durchgeführt. Es ist kein Wissen über die Empfänger nötig.

2.4.1 Mtrace

Der Multicast-Traceroute (Mtrace) [9] basiert auf dem Empfänger zur Quelle Ansatz, um die Multicast-Topologie zu bestimmen. Möchte man lediglich den Pfad zu einem bestimmten Empfänger ermitteln, ist es einfacher den Weg vom Empfänger zur Quelle zu nehmen. Denn wird der Pfad von der Quelle zu mehreren Empfängern bestimmt, kann nicht ermittelt werden, über welchen Zweig des Multicast-Baumes der Empfänger zu erreichen ist. Deshalb wäre es nötig, den ganzen Baum zu fluten. Das Verfolgen eines Pfades auf dem Weg vom Empfänger zur Quelle hingegen betrachtet nur die Router, die auf dem direkten Pfad zwischen den beiden Knoten liegen. Es ist nicht nötig Teil der Multicast-Gruppe oder des Multicast-Netzwerkes zu sein, um einen Multicast-Traceroute auszuführen.

Zum Ausführen des Multicast-Traceroutes wird eine Query-Nachricht an den Router des Empfängers gesendet, dessen Pfad zur Quelle ermittelt werden soll. Die Query-Nachricht enthält die Adresse von der Quelle, die Multicast-Adresse und die Response-Adresse. Der Empfänger wandelt diese in eine Request-Nachricht um, indem er einen Request-Datenblock hinzufügt. Der Request-Datenblock enthält Informationen über die Zeit, wann das Paket eingetroffen ist, alle ausgehenden Interface Adressen, die Anzahl der bisher ausgesendeten Pakete des Routers und die Forwarding TTL. Die Request-Nachricht wird dann an den vorherigen Hop Router mittels Unicast weitergeleitet. Ermittelt wird dieser Router, indem derselbe Mechanismus angewandt wird, den der Router verwenden würde, wenn ein Paket von der Quelle weitergeleitet werden soll. Nach Empfang einer Request-Nachricht fügt der Router einen neuen Response-Block zu dem Paket hinzu und leitet das Paket, wie oben beschrieben, an den vorherigen Hop Router weiter. Ist der Router erreicht worden, von dem angenommen wird, dass er über eine seiner Netzwerkverbindungen direkt mit der Quelle verbunden ist, wird aus der Request-Nachricht eine Response-Nachricht erzeugt und an die Response-Adresse gesendet.

2.4.2 Tracetree

Um die Topologie-Informationen eines Multicast-Netzwerkes zu ermitteln, wird bei Tracetree [23] der Quelle zu Empfänger Ansatz angewendet. Um das Verhalten des Tracetree Algorithmus zu beschreiben, wird das Beispiel in Abbildung 2.9 verwendet.

Der Im Beispiel dargestellte Querier Q hat Interesse, die Baumtopologie der Gruppe (S,G) zu entdecken. Zur Ermittlung der Topologie generiert Querier Q eine Tracetree Query-Nachricht. Diese wird dann an den “first hop router“ A gesendet. Nach Empfang der Nachricht wandelt A diese in eine Tracetree Request-Nachricht um und leitet sie an die Multicast-Gruppe G, die Router B und C, weiter. Als Quell-Adresse für die Nachricht verwendet A die Adresse der Session Source S.

Die Request-Nachricht enthält einen Protokoll Header, dieser verfügt über zwei Einträge, den Request-Forwarder und den Response-Destination Eintrag. In den Request-Forwarder Eintrag wird die IP Adresse des Routers eingetragen, der die Nachricht weiterleitet. Der Response-Destination Eintrag enthält die IP-Adresse des Querier. Damit der Router Request-Nachrichten von normalen Multicast-Datenpaketen unterscheiden kann, wird die “IP Router altert Option“gesetzt, diese zwingt den Router dazu, Pakete genauer zu analysieren anstatt sie direkt weiterzuleiten. Nach Empfang einer Tracetree Request-Nachricht, wird eine Antwort an die im Response-Destination Eintrag stehende IP-Adresse gesendet. Bevor die Nachricht weitergeleitet wird, ersetzt der Router die IP Adresse im Request-Forwarder Eintrag durch seine eigene.

Der gesamte Baum lässt sich aus der Summe der Tracetree Response-Nachrichten bestimmen. Der oben beschriebene Algorithmus eignet sich ebenfalls für das Bestimmen von Unterbäumen einer Gruppe(S,G). Zum Bestimmen eines Unterbaumes wird eine Query-Nachricht an einen geeigneten Router, der sich in der Multicastgruppe (S,G) befindet, gesendet.

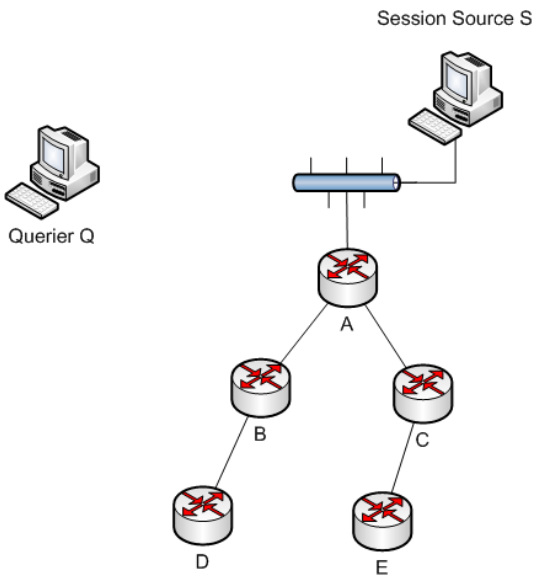

Abbildung 2.9: Multicast Baum für die Gruppe (S,G) Sarac und Almeroth [23]

2.5 Visualisierung der Multicast-Topologie

Die Visualisierung von Multicast-Topologie Informationen verbessert die Übersichtlichkeit und erleichtert das Erfassen von Informationen für den Benutzer. Im folgenden Abschnitt werden zwei Programme vorgestellt, die einen Überblick darüber geben, wie Topologie Informationen sinnvoll dargestellt werden können.

2.5.1 MHealth

MHealth [18] ist ein Multicast-Monitoring Tool, das sowohl Anwendungs- als auch Routing-Informationen kombiniert, um ausführliche Informationen über ein Multicast-Netzwerk bereitstellen zu können. Die Routing Informationen werden mit Hilfe von Mtrace (siehe Abschnitt 2.4.1) ermittelt. Mtrace ist nicht Teil des Monitoring Tools, es wird eine bereits existierende Implementation genutzt. Zum Auffinden der Topologie startet MHealth das externes Mtrace Programm und liest die Ergebnisse der Ausgabe ein. Daten wie Jitter, Paket loss, delay und Gruppenzugehörigkeit werden über das RTCP Protokoll [21] erlangt.

Nachdem mit Hilfe von RTCP alle Gruppenmitglieder ermittelt wurden, wird durch Mtrace die Multicast-Topologie aufgedeckt. Es werden alle Informationen zusammengetragen und aus der Summe der einzelnen Informationen eine Baumstruktur erstellt. Informationen, wie die Anzahl der Multicast Pakete pro Hop, gruppenspezifischer Hop Count, Netzwerk Topologie und Paket loss pro Hop werden von MHealth erfasst und grafisch dargestellt.

Die Voraussetzungen für die Ausführung von MHealth sind Mtrace fähige Router und Hosts, auf denen eine RTCP Anwendung ausgeführt wird.

Visuelle Darstellung der Daten

In MHealth werden Router durch Rechtecke visualisiert und die Netzwerk-Verbindungen zwischen den Routern durch Linien dargestellt. Die IP-Adressen der Router werden in das Rechteck platziert. Weitere Informationen über die Router werden durch Farben symbolisiert. So bedeutet z.B. die Farbe rot zehn Prozent oder mehr Loss.

2.5.2 Otter

Otter [12] ist eine Visualisierungs-Software mit der beliebige Netzwerkdaten visualisiert werden können. So kann z.B. mit Hilfe von Mrinfo [14] , Mbone Netzwerkdaten ermittelt werden. Mrinfo ist ein Programm das Multicast Router Informationen über IGMP (ASK_NEIGHBORS) Anfragen ermittelt und Informationen, wie die Multicast Version, Multicast Protokoll oder Routing Nachbarn abruft. Die daraus gewonnenen Daten können dann mit Otter visualisiert werden. Otter arbeitet datenunabhängig. So können Daten, die durch externe Tools ermittelt wurden, in das Otter Format umgewandelt und dargestellt werden.

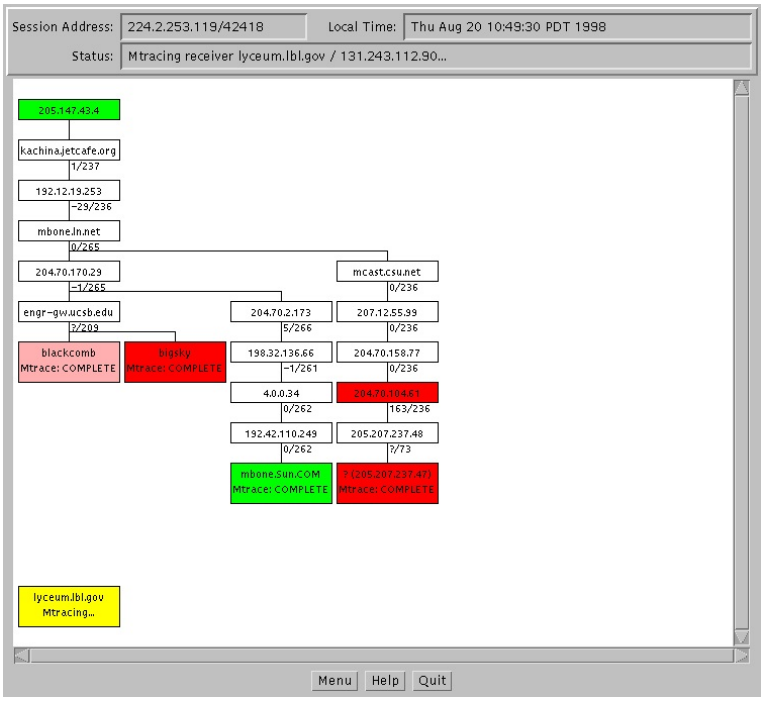

Abbildung 2.10: Mhealth Screenshot Quelle [17]

Visuelle Darstellung der Daten

Otter bietet zwei Graphenlayouts zur Visualisierung, semi-geographical-layout (siehe Abbildung 2.11) und topological-layout (siehe Abbildung 2.12). Die Layout Algorithmen positionieren den Netzwerkgraphen und können zur Laufzeit, durch das Verschieben von einzelnen Elementen, durch den Benutzer korrigiert oder angepasst werden. Gezeichnet werden Vierecke, die durch Linien miteinander verbunden sind. Die Vierecke stellen Multicast-Router dar. Rot gekennzeichnete Vierecke sind Roots. Die Root-Nodes können durch die eingegebenen Daten oder zur Laufzeit interaktiv bestimmt werden. Die Linien stellen Verbindungen zwischen den Routern dar. Knoten können mit einem Domain Lable versehen werden (siehe Abbildung 2.13) und wahlweise können alle Router Attribute durch Farben dargestellt werden, z.B. die Domain Zugehörigkeit (siehe Abbildung 2.14), ihre Verfügbarkeit oder Multicast Version.

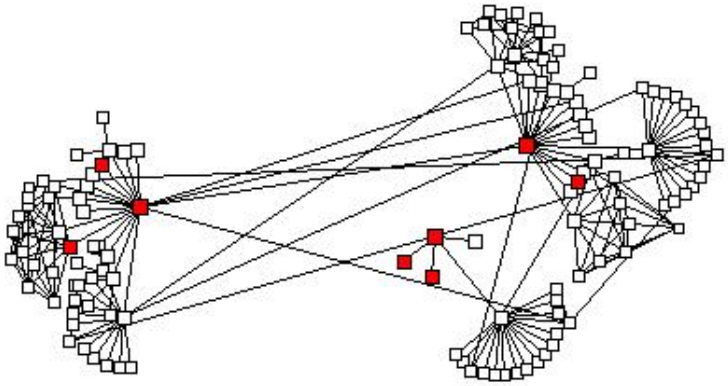

Abbildung 2.11: Semi-geographical-layout (Quelle [13])

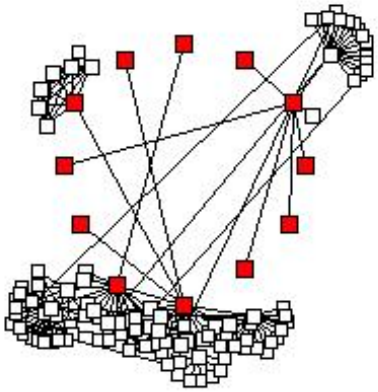

Abbildung 2.12: Topological layout (Quelle [13])

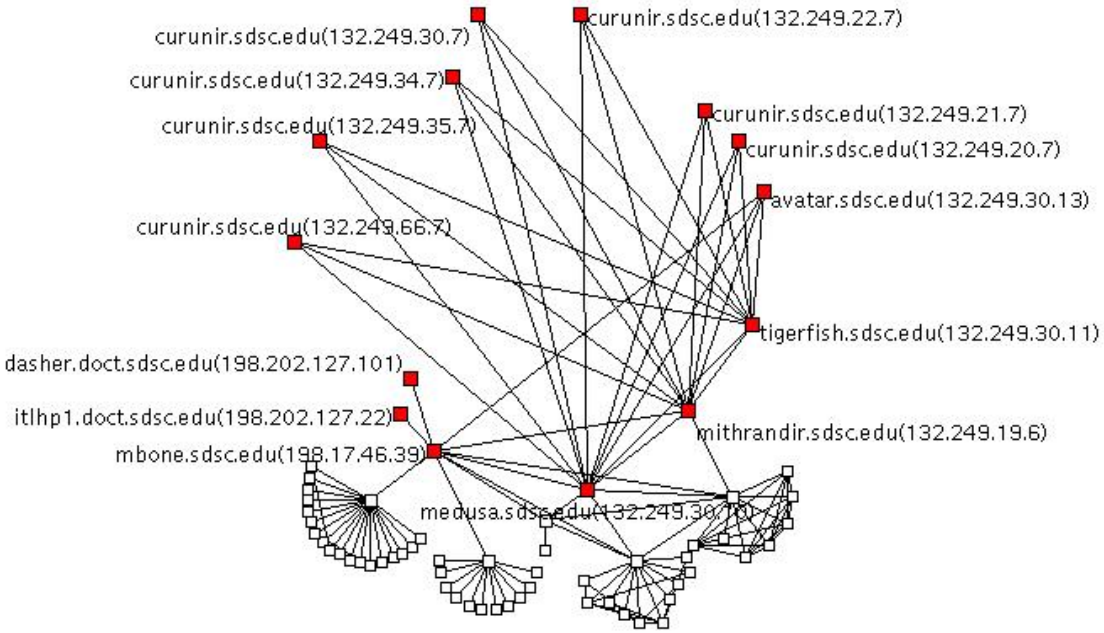

Abbildung 2.13: Node label (Quelle [13])

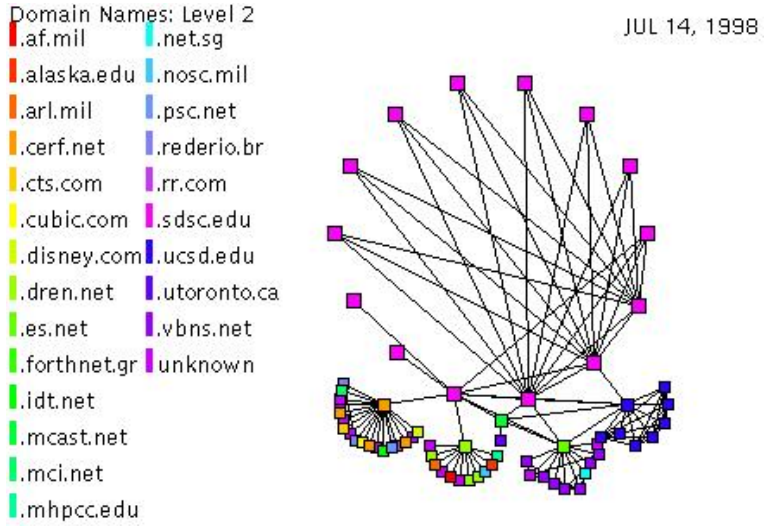

Abbildung 2.14: Farbiege Domains (Quelle [13])

3 HAMcast

Multicast existiert in einer Vielzahl von Varianten und Technologien wie z.B. IPv4, IPv6, ASM, SSM und Application-Layer-Multicast. Zurzeit gibt es keine einheitliche Programmierschnittstelle, die den Multicast Dienst von der Technologie abstrahiert. Somit ist es den Softwareentwicklern überlassen, die eine Gruppenkommunikations-Anwendung implementieren wollen, festzulegen welche Technologie die Anwendung zur Laufzeit benötigt. Im Fall von Application-Layer-Multicast muss in den meisten Fällen das Overlay und der Multicast-Algorithmus auf die Anwendung angepasst werden. Daraus ergibt sich, dass heutige Multicast-Anwendungen mit zwei Problemen zu kämpfen haben. Entweder sind sie technologieabhängig und funktionieren nur unter bestimmten Netzwerk Vorrausetzungen oder es werden Leistungseinbußungen in Kauf genommen, um die Anwendung für eine breitere Benutzergruppe zur Verfügung stellen zu können.

3.1 HAMCast Architektur

Das HAMcast Projekt[20] stellt einen universellen Multicast bereit, ohne dabei Veränderungen an vorhandenen Technologien und Protokollen vornehmen zu müssen. Vielmehr werden die bereits vorhandenen Technologien und Protokolle in ihrer Funktionalität durch HAMcast erweitert. HAMcast verfolgt das Konzept, das ein Netzwerkdienst nicht durch seine Technologie oder der Netzwerkschicht, auf der er implementiert wurde, sonder durch seine Funktionalität definiert wird. Die HAMcast-Architektur bietet Funktionalitäten, die es dem Softwareentwickler ermöglichen Gruppenkommunikations-Anwendungen zu entwickeln und zu implementieren, welche unabhängig von Netzwerktechnologie und Protokollen operieren.

Um einen technologieunabhängigen Dienst zur Verfügung stellen zu können, muss die Adressierung und Namensgebung von der verwendeten Technologie abstrahiert werden. HAMcast basiert auf dem Identifier-Locator Konzept, d.h. Gruppennamen werden von technologieabhängigen Adressen separiert. HAMcast-Anwendungen verwenden Gruppennamen zur Identifikation der Gruppe. Durch diese Namen kann die verwendete Technologie für das Verteilen der Daten vor der Anwendung verborgen werden. Ein Gruppenname wird in HAMcast durch eine URI repräsentiert.

Beispiel für eine URI:

scheme :// group @ instantiation : port / sec-credentials

Das „scheme“ bezeichnet den Namensraum, z.B. ip, sip oder scribe. „group“ gibt die Gruppen ID an. „instantiation“ identifiziert die Entität, die eine Instanz der Gruppe generiert, z.B. bei SSM. „port“ identifiziert eine bestimmte Applikation und sec-credentials ist

optional für die Implementierung von Sicherheitsmechanismen. Zulässige Gruppen IDs sind z.B. ip://239.0.0.1:1234, scribe://239.0.0.1:1234 oder sip://snoopy@peanuts.com.

Das Adress-Mapping kann dadurch realisiert werden, dass kleinere IDs in größere Namensräume eingebunden werden oder eine willkürliche unbenutzte ID im Zielnamensraum ausgewählt wird. Die Beziehungen zwischen logischen IDs, den Gruppennamen und technischen IDs können durch Mapping Funktionen verwaltet werden. Diese können entweder zustandslos oder zustandsbehaftet sein.

Dazu wird ein Late-Binding von technologiespezifischen Adressen und Namen zur Laufzeit vorgenommen. Das Binding von Adressen und Namen zur Laufzeit ermöglicht es Softwareentwicklern, Anwendungen zu implementieren ohne eine vorherige Festlegung auf eine bestimmte Technologie.

HAMcast unterscheidet zwischen zwei Arten von Hosts. Hosts, die sich in einer Multicast-Insel befinden und Hosts, die sich in einem Netzwerk ohne Multicast-Unterstützung befinden. Hosts, die sich in einem nicht multicastfähigem Netzwerk befinden, können sich zu einem Application-Layer-Multicast-Netzwerk über Unicast verbinden.

Multicast-Inseln werden über Interdomain-Multicast-Gateways (IMG) ([26], [27]) miteinander verbunden. IMGs verbinden Multicast-Inseln über ein Application-Layer-Multicast-Netzwerk. Dazu muss ein IMG Multicastdaten von einer Technologie auf eine andere übersetzen. Um das durchführen zu können, benötigt ein IMG Mechanismen um Gruppennamen zwischen technologiespezifischen Gruppenadressen übersetzen zu können. Informationen über Gruppenzugehörigkeit und Quellen werden dazu benötigt.

Die Abbildung 3.1 zeigt ein Beispiel, in dem vier Multicast-Inseln, die alle über verschiedene Multicast-Technologien verfügen, über IMGs miteinander verbunden werden.

3.2 HAMcast Middleware

Um auf den HAMcast Multicast-Dienst zugreifen zu können, wird als Teil des HAMcast Projektes ein Applikation-Programming-Interface (API) zur Verfügung gestellt, die „common multicast API“ [28]. Die API wird durch eine Middleware realisiert. Abbildung 3.2 zeigt den Aufbau der Middleware. Die Middleware verfügt über Mechanismen zur Service-Discovery, Service-Selection und Name-Address-Mapping. Die unterstützten Multicast-Technologien sind in Module aufgeteilt. Diese Aufteilung ermöglicht eine hohe Erweiterbarkeit, Anpassung und Flexibilität der Middleware. So können neue Technologien hinzugefügt werden, ohne eine Veränderung der Anwendung. Die Anwendungen kommunizieren über einen Socket-Stub mit der Middleware, der Socket-Stub ist mittels Inter-Prozess-Kommunikation (IPC) mit der Middleware verbunden.

Das Service-Discovery-Modul ermittelt die vorhanden physikalischen Multicast-Technologien und analysiert diese. Unter Beachtung der, von der Middleware unterstützten Multicast-Technologien, kann dann die optimale zur Verfügung stehende Technologie ermittelt werden.

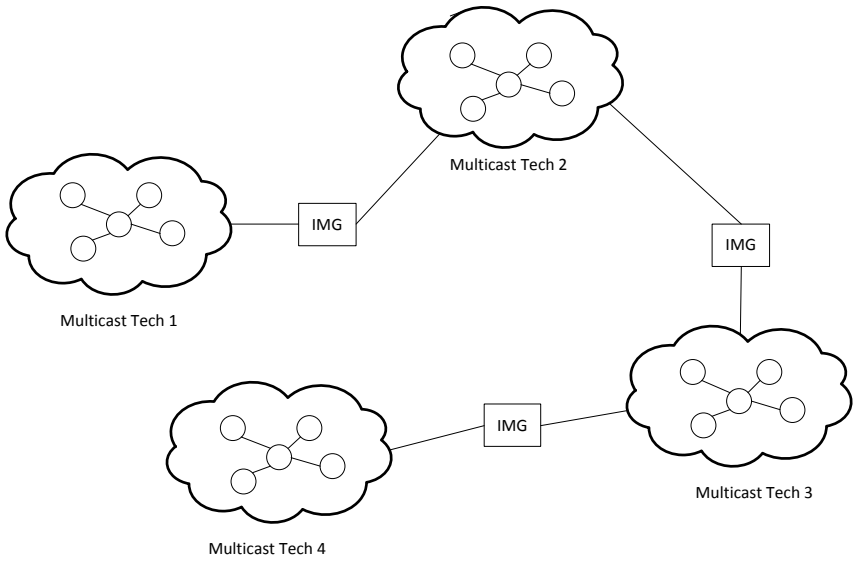

Abbildung 3.1: IMGs die Multicast-Inseln miteinander verbinden

Das Service-Selection-Modul wählt Service-Interfaces aus, unter Berücksichtigung der Service-Discovery, um sie mit einem HAMcast-Socket zu verbinden. Service-Interfaces sind virtuelle Schnittstellen zwischen dem HAMcast-Socket und dem Technologie Modul.

Das Name-Address-Mapping löst eine Gruppen ID so auf, dass sie einer technologiespezifischen Adresse zugeordnet werden kann.

Zum Ausführen des HAMcast Multicast-Dienstes muss eine Instanz der Middleware auf jedem Host, der den Dient nutzen möchte, initiiert werden.

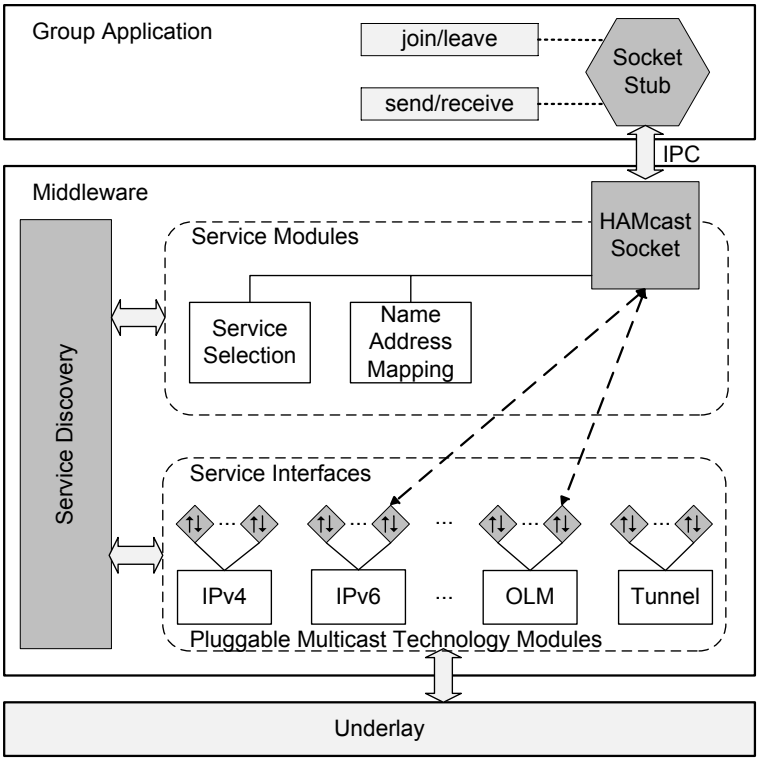

Abbildung 3.2: Aufbau der HAMcast Middleware

3.3 Common Multicast API

Die „Common Multicast API“ [28] stellt Funktionen bereit, die es dem Softwareentwickler ermöglichen, die Architekturvorteile von HAMcast über eine definierte Schnittstelle nutzen zu können. Funktionen wie das Erstellen von Multicast-Sockets, Join Group und Leave Group werden von der HAMcast API bereit gestellt.

3.3.1 Interface

Die Funktion *if_prop(void)* gibt eine Liste von Interface-Datenstruktur aller existierender Interfaces zurück. Die Funktion benötigt keine Argumente.

*struct if_prop *if_prop(void);*

Aufbau der Interface-Datenstruktur :

- if_index : die Identifikationsnummer für das Interface.
- if_name : der Name des Interface.
- if_addr : die technologiespezifische Adresse.
- if_tech : die Technologie des Interface.

3.3.2 Group Management Calls

Create

Die Funktion Create dient zum Erstellen eines Multicast-Socket.

*int createMSocket(uint32_t *if);*

Argument *if ist eine Liste über Interface Kennungen. Wenn kein Interface angegeben ist, wird ein als Standard definiertes Interface mit dem Multicast-Socket verbunden.

Destroy

Destroy entfernt den Multicast-Socket.

int destroyMSocket(int s);

Das Argument s ist eine Kennung für einen Multicast-Socket.

Join

Mit dem Join Aufruf kann einer Multicast-Gruppe beigetreten werden.

int join(int s, const uri group_name);

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument group_name identifiziert die Gruppe.

Leave

Der Leave Aufruf ermöglicht es, eine Multicast-Gruppe zu verlassen.

int leave(int s, const uri group_name);

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument group_name identifiziert die Gruppe.

Source Register

Der Source Register Aufruf registriert einen Sender für eine Gruppe auf alle aktiven Interfaces des Multicast-Sockets s. Dieser Aufruf wird nicht von allen Multicast-Technologien benötigt.

*int srcRegister(int s, const uri group_name, uint_t num_ifs, uint_t *ifs);*

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument group_name identifiziert die Gruppe. Das Argument num_ifs gibt die Anzahl der Interfaces an, die über das Array ifs zurückgegen wurden. Das Argument ifs zeigt auf eine Liste von Interface Kennungen.

Source Deregister

Der Source Deregister Aufruf gibt an, dass ein Sender nicht länger beabsichtigt, Daten an eine Gruppe zu senden. Source Deregister wird auf alle aktiven Interfaces für den Socket s ausgeführt. Dieser Aufruf hat nicht auf alle Multicast-Technologien Einfluss.

*int srcDeregister(int s, const uri group_name, uint_t num_ifs, uint_t *ifs);*

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument group_name identifiziert die Gruppe. Das Argument num_ifs gibt die Anzahl der Interfaces an, die über das Array ifs zurückgegen wurden. Das Argument ifs zeigt auf eine Liste von Interface Kennungen.

3.3.3 Send and Receive Calls

Send

Der Send Aufruf ermöglicht das Senden auf einen Multicast-Socket.

*int send(int s, const uri group_name, size_t msg_len, const void *buf);*

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument group_name identifiziert die Gruppe, msg_len gibt die Länge der Nachricht an, die versendet wird. Das Argument *buf zeigt auf den Nachrichten-Buffer.

Receive

Ermöglicht das Empfangen von Daten.

*int receive(int s, const uri group_name, size_t msg_len, msg *msg_buf);*

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument group_name identifiziert die Gruppe, msg_len gibt die Länge der Nachricht an, die empfangen wurde. Das Argument *buf zeigt auf den Nachrichten-Buffer.

3.3.4 Socket Options

Get Interfaces

Der Get Interfaces Aufruf gibt eine Liste aller Aktiven Interfaces zurück, die im Zusammenhang mit dem Socket s stehen.

*int getInterfaces(int s, uint_t num_ifs, uint_t *ifs);*

Argument s ist eine Kennung für einen Multicast-Socket. Das Augment num_ifs gibt die Anzahl der Interfaces an, die über das Array ifs zurückgegeben wurden. Das Argument ifs zeigt auf eine Liste von Interface Kennungen.

Add Interface

Der Aufruf Add Interface fügt einen weiteren Verteilungskanal (Interface) zum Socket hinzu.

int addInterface(int s, uint32_t if);

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument if identifiziert das Interface, das hinzugefügt werden soll.

Delete Interface

Der Aufruf Delete Interface entfernt ein Interface von dem Socket.

int delInterface(int s, uint32_t if);

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument if identifiziert das Interface, das entfernt werden soll.

Set TTL

Der Aufruf Set TTL konfiguriert den maximalen Hop Count für einen Socket. Dieser gibt an, wie viele Hops eine Multicast-Nachricht durchführen kann.

*int setTTL(int s, int h, uint_t num_ifs, uint_t *ifs);*

Argument s ist eine Kennung für einen Multicast-Socket. Das Argument h gibt die Anzahl der Hops an. Das Augment num_ifs gibt die Anzahl der Interfaces an, die über das Array ifs zurückgegeben wurden. Das Argument ifs zeigt auf eine Liste von Interface Kennungen.

3.3.5 Service Calls

Group Set

Der Aufruf Group Set gibt alle registrierten Gruppen eines Interfaces zurück.

*int groupSet(uint32_t if, uint_t *num_groups, struct groupSet *groupSet);*

Argument if ist eine Kennung für ein Interface. Das Argument num_groups gibt die Anzahl der Gruppen im groupSet Array an. Das Augment groupSet ist eine Liste über groupSet Datenstrukturen.

Aufbau der Datenstruktur groupSet :

- uri group_name : die Group ID
- int type : gibt den Zustand der Gruppe an. 0 = listener state, 1 = sender state, 2 = sender und listener state

Neighbor Set

Der Aufruf gibt eine Liste über Nachbar-Knoten zurück.

*int neighborSet(uint32_t if, uint_t *num_neighbors, const uri *neighbor_address);*

Argumente sind, if die Interface Kennung, *num_neighbors die Anzahl der Einträge in der Liste und *neighbor_address das Array, das auf die URIs der Nachbar-Knoten zeigt.

Children Set

Der Aufruf gibt eine Liste von Kind-Knoten zurück.

*int childrenSet(uint32_t if, const uri group_name, uint_t *num_children, const uri *child_address);*

Argumente sind, if eine Interface Kennung, group_name eine Group ID, *num_children die Anzahl der Einträge in der Liste und *child_address das Array, das auf die URIs der Kind-Knoten zeigt.

Parent Set

Der Aufruf gibt eine Liste von Eltern-Knoten zurück.

*int parentSet(uint32_t if, const uri group_name, uint_t *num_parents, const uri *parent_address);*

Argumente sind, if eine Interface Kennung, group_name eine Group ID, *num_parents die Anzahl der Einträge in der Liste und *parent_address das Array, das auf die URIs der Eltern-Knoten zeigt.

Designated Host

Der Aufruf Designated Host ermittelt, ob ein Host designated forwarder/querier ist oder nicht.

*int designatedHost(uint32_t if, const uri *group_name);*

Argumente sind, if eine Interface Kennung und group_name eine Group ID.

Update Listener

Der Aufruf Update Listener informiert den group service über eine Veränderung des Empfängerzustands.

*const uri *updateListener();*

Es werde keine Argumente benötigt.

4 Konzept und Implementierung

In diesem Kapitel wird das Konzept und die Implementierung der Visuallisirungs- und Monitoring-Software für die HAMcast-Architektur vorgestellt.

4.1 Konzept

4.1.1 Visualisierung der Daten

Die Aufgabe der Software ist es, HAMcast Topologie Informationen visuell darzustellen. Dabei muss zunächst berücksichtigt werden, welche Daten zur Verfügung stehen und wie diese am besten visualisiert werden.

Topologie Informationen

Die HAMcast API bietet Funktionen, die Topologie Informationen zur Verfügung stellen. Die Aufrufe sind in mehrere Kategorien unterteilt, Interface(siehe 3.3.1), Group Management Calls (siehe 3.3.2), Send and Receive Calls (siehe 3.3.3) und Service Calls (siehe 3.3.5). Aufrufe die Informationen über die Topologie geben können, befinden sich in den Kategorien Interface und Service Calls.

Die Interface Kategorie bietet die Funktion *if_prop* , über die Interface Informationen abgerufen werden können. Der Aufruf gibt eine Liste aller existierender Interfaces zurück. Diese enthält zu jedem Interface Informationen, wie die Interface Kennung, die Technologie, den Namen und eine technologiespezifische Adresse.

Die Service Calls Kategorie bietet zwei wichtige Aufrufe, über die Topologie Informationen gewonnen werden können, *groupSet* und *childrenSet* . Die Funktion *groupSet* gibt eine Liste aller registrierten Gruppen für ein Interface zurück. Diese enthält Informationen, wie die Gruppen ID und den Status der Gruppe. Die Funktion *childrenSet* gibt eine Liste von URIs aller Kind-Knoten zurück. Die URI enthält die technologiespezifische Adresse des Knoten.

Funktionsaufrufe:

- Interface: *struct if_prop *if_prop(void)*
- Service Call: *int groupSet(uint32_t if, uint_t *num_groups, struct groupSet *groupSet)*
- Service Call: *int childrenSet(uint32_t if, const uri group_name, uint_t *num_children, const uri *child_address)*

Zeichnen des Netzwerkgraphen

Aus den Daten, die durch die Funktionen *childrenSet* und *if_prop* gewonnen werden, kann ein Netzwerkgraph erstellt werden.

Bei der Darstellung des Netzwerkgaphen kann man generell zwischen zwei Elementen unterscheiden, den Knoten (Hosts/Router) und den Netzwerkverbindungen. Die Hosts und Router werden durch eine einfache geometrische Form, eine Ellipse, dargestellt. Diese einfache Darstellung macht eine Visualisierung von großen Netzwerken übersichtlicher als komplexere Formen, wie z.B. Piktogramme. Die Netzwerkverbindungen lassen sich am besten durch Linien darstellen.

Um einen solchen Netzwerkgraphen erstellen zu können, müssen die Informationen aller HAMcast-Hosts miteinander kombiniert und verglichen werden.

Die Abbildung 4.1 zeigt ein Beispiel, wie aus den Informationen ein Netzwerkgraph erstellt werden kann. Der Knoten A besitzt zwei Kinder B und C. Der Knoten B besitzt zwei Kinder D und E und der Knoten C besitzt ein Kind F. Als erstes wird die Wurzel bestimmt, dazu werden alle Kinder in eine Liste eingetragen. Der Knoten, der sich nicht in der Liste befindet, ist die Wurzel. Im Fall des Beispiels ist die Wurzel der Knoten A. Der Knoten A und seine Kinder werden als erstes gezeichnet. Anschließend werden die Kinder von A mit den anderen Knoteninformationen verglichen. Für die Knoten B und C stehen weitere Informationen zur Verfügung, somit können B und C, sowie ihrer Kinder unter A gezeichnet werden.

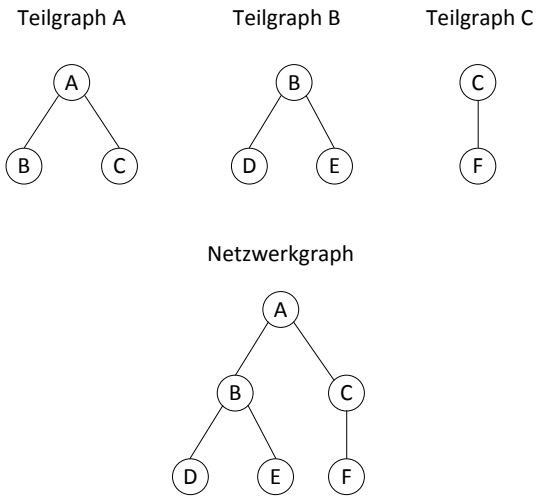

Abbildung 4.1: Beispiel für das Erstellen des Netzwerkgraphen

Darstellung weiterer Informationen

Weitere Informationen, wie die verwendete Technologie einer Netzwerkverbindung, können durch eine farbliche Kennzeichnung der Verbindung dargestellt werden. Komplexere Informationen, die durch die Funktionen *groupSet* und *if_prop* zur Verfügung gestellt werden, können auf die Anfrage des Benutzers in einer Tabelle abgerufen werden.

4.1.2 Datengewinnung

Für den Zugriff auf die HAMcast API wird auf den HAMcast-Hosts ein Programm benötigt, dass die Informationen, die über die API bereit gestellt werden, abruft. Daraus ergibt sich, dass die Visualisierungs- und Monitoring-Software in zwei Programme unterteilt werden muss, ein Programm für die Visualisierung (Monitoring Viewer) und ein Programm, um die Informationen auf den HAMcast-Hosts abzurufen (Monitoring Daemon).

Kommunikation zwischen Monitoring Viewer und Monitoring Deamon

Der Monitoring Viewer und der Monitoring Daemon tauschen die Informationen über eine TCP-Verbindung aus. Um eine TCP-Verbindung zwischen dem Monitoring Viewer und den Monitoring Daemons herstellen zu können, benötigen die Monitoring Daemons die IP-Adresse des Monitoring Viewers. Hierzu wird ein Mechanismus benötigt, der die IP-Adresse des Monitoring Viewers an die Monitoring Daemons überträgt.

Der Monitoring Viewer und die Monitorin Daemons werden über HAMcast-Sockets miteinander verbunden. Dadurch, dass die Programme über Multicast miteinander kommunizieren, ist es möglich, über eine als Standard definierte Multicastgruppe für das Monitoring, eine Anfragenachricht an die Monitoring Daemons zu schicken. Der Inhalt der Anfragenachricht ist die IP-Adresse und Port des Monitoring Viewers. Erhält ein Monitoring Daemon eine solche Anfragenachricht, kann er sich über TCP mit dem Monitoring Viewer verbinden. Auf Anfrage des Monitoring Viewers können nun Informationen ausgetauscht werden. Weitere Anfragenachrichten, wie das Ermitteln der Netzwerk-Topologie, werden über Multicast an die Monitoring Daemons versendet. Die Antwort der Monitoring Daemons erfolgt über die TCP-Verbindung.

4.1.3 Test Programm: Video Streamer

Der Video Streamer ist ein Programm, das die UDP Datenströme des VLC Media Players [22] über die HAMcast API an eine Multicast-Gruppe sendet und auf der Empfängerseite an den VLC Player weiterleitet. Das Programm dient dazu, die Funktionalität der HAMcast API zu testen und visuell zu präsentieren. Der Video Streamer kann in zwei Modi gestartet werden, dem Server und den Client Modus.

Im Server Modus 4.1 werden die Netzwerkpakete die von VLC für einen Video Stream generiert werden, über einen Lokalen UDP Socket empfangen und anschließend über einen HAMcast Socket, an eine vom Benutzer angegebene Multicast-Gruppe, gesendet. Die Netzwerkpakete können im Client Modus 4.2 durch einen HAMcast Socket empfangen werden und über einen lokalen UDP Socket an den VLC Media Player übergeben werden.

Der Video Streamer bietet damit eine einfache Möglichkeit einen Video Stream über eine Hybride Multicast Architektur zu senden.

Listing 4.1: Video Streamer im Server Modus

```
0  void startServer(string group, int vlcport){
       multicast_socket s; // Erstellen des HAMcast Multicast
           Sockets
       unsigned char sendbuffer[MAX_MSG];
       memset(sendbuffer,0x0,MAX_MSG);

5      int sd = initReceiveUDPSock(vlcport); // Initialisierung des
           UPD Sockets
       if(sd <0){
           s.close(); // HAMcast Socket schließen
                   return; // Programm beenden
       }
10     while(true){
           int rc = receiveFromSocket(sendbuffer,sd); // Empfangen
               der Datenpakete von VLC

           if(rc <0){
               continue
15         }
           s.send(group, rc, sendbuffer); // Weiterleitung der
               Datenpakete über HAMcast
       }
       return;
   }
```

Listing 4.2: Video Streamer im Client Modus

```
0  void startClient(string group, int vlcport){

       multicast_socket ms; // Erstellen des HAMcast Multicast-
           Sockets
   int sd = initSendUDPSock(vlcport);  // Initialisierung des
       UPD Sockets
   if(sd < 0){
5                  s.close(); // HAMcast Socket schließen
                   return; // Programm beenden
   }
   ms.join(group);
   while(true){
10       multicast_packet mp = ms.receive(); // Datenpakete über
             HAMcast empfangen
         const char* msg = reinterpret_cast< const char*>(mp.
             data());
         sendData(msg,mp.size(),sd); // Datenpakete an den VLC
             Player senden
   }
}
```

4.2 Implementierung

4.2.1 Programmiersprache und externe Bibliothek

Als Programmiersprache für die Implementation wird C++ verwendet. Aufgrund der derzeitigen Implementation der HAMcast API, die als C++ Implementierung vorliegt. Um dennoch ähnlich wie bei Java Plattform unabhängig zu bleiben, gibt es Bibliotheken, wie z.B. das Qt UI-Framework [5] oder GTK+ [24].

Die Entscheidung fiel auf das Qt UI-Framework, aufgrund der guten Dokumentation, den umfangreichen Code Beispielen und dem Graphics View Framework, das zur Visualisierung von 2d Grafiken entwickelt wurde.

Qt ist ein umfangreiches Framework, das neben einer Bibliothek zum Erstellen von grafischen Benutzeroberflächen noch weitere Funktionalitäten zur Verfügung stellt, u.a. für Netzwerk, Threading und generische Datentypen. Werden die von Qt zur Verfügung gestellten Bibliotheken verwendet, anstatt plattformspezifische Funktionen, wie z.B Netzwerk Sockets, lässt sich der Quellcode für Windows, Linux und Mac OS kompilieren.

4.2.2 Qt Graphics View Framework

Zur Visualisierung der Daten wird das Qt Graphics View Framework [4] verwendet. Das Graphics View Framework kann eine große Anzahl von 2D Grafik Objekten verwalten und managen.

Das Framework besteht aus drei Elementen:

- QGraphicsScene
- QGraphicsView
- QGraphicsItem

QGraphicsScene

Die QGraphicsScene bietet ein schnelles Interface für das Verwalten einer großen Menge von 2D Grafik Objekten. Ihre Aufgabe ist das Übertragen von Ereignissen an die Grafik Objekte, das Managen von Objektzuständen, wie das Auswählen oder Verschieben von Objekten und das Zeichnen der Grafikobjekte.

QGraphicsView

Die QGraphicsView visualisiert die Objekte, die von einer QGraphicsScene verwaltet werden. Die View bietet Scrollleisten oder Zoomfunktionen zur Navigation in einer Zeichnung. Maus und Tastatur Ereignisse werden von der View in Scene-Ereignisse übersetzt und an die Scene weitergeleitet.

QGraphicsItem

Das QGraphicsItem ist die Basisklasse für alle 2D Grafikobjekte. Es existiert eine Auswahl von vorgefertigten Objekten, wie z.B. Linien, Ellipsen, Rechtecken und Textklassen. Die QGraphicsItem Klasse besitzt Standard Eigenschaften, wie Maus und Tastatur Ereignisse, Drag and Drop und Kollisionserkennung. Zum Erstellen eigener Grafikklassen muss von QGraphicsItem abgeleitet werden und einige Standardfunktionen reimplementiert werden. Die Funktionen die reimplementiert werden müssen, sind Paint und BoundingRect. Die Paint Funktion wird von der QGraphics View aufgerufen, um das GraphicsItem zu zeichnen. Die BoundingRect Funktion gibt den Bereich an, indem ein GraphicsItem gezeichnet wird. Bewegt sich ein anderes GraphicsItem in diesem Bereich, führt dies zu einer Neuzeichnung des Items.

4.2.3 Graphical User Interface (GUI) des Monitoring Viewer

Das Graphical User Interface der Monitoring View dient als Schnittstelle zur Interaktion mit dem Benutzer. Abbildung 4.2 zeigt die Benutzeroberfläche der Software. Die Daten werden gruppenspezifisch abgerufen. Der Benutzer gibt über ein Textfeld die HAMcast Gruppen ID ein. Mit der Taste Draw wird das Ermitteln der Daten und das Zeichnen der Netzwerk-Topologie ausgelöst. Der Netzwerkgraph wird in das rechts liegende Feld (das QGraphicsView Widget) gezeichnet. Die Hosts/Router werden durch Ellipsen (Nodes) 4.4 dargestellt. Die Netzwerkverbindungen werden durch Linien (Edges) 4.5 zwischen diesen Ellipsen symbolisiert. Die technologiespezifischen Adressen von Hosts/-Router werden in textform unter den Ellipsen platziert. Die Interface Informationen über die Knoten werden in dem links stehenden Feld (QTabWidget) in einer Tabelle 4.3 dargestellt.

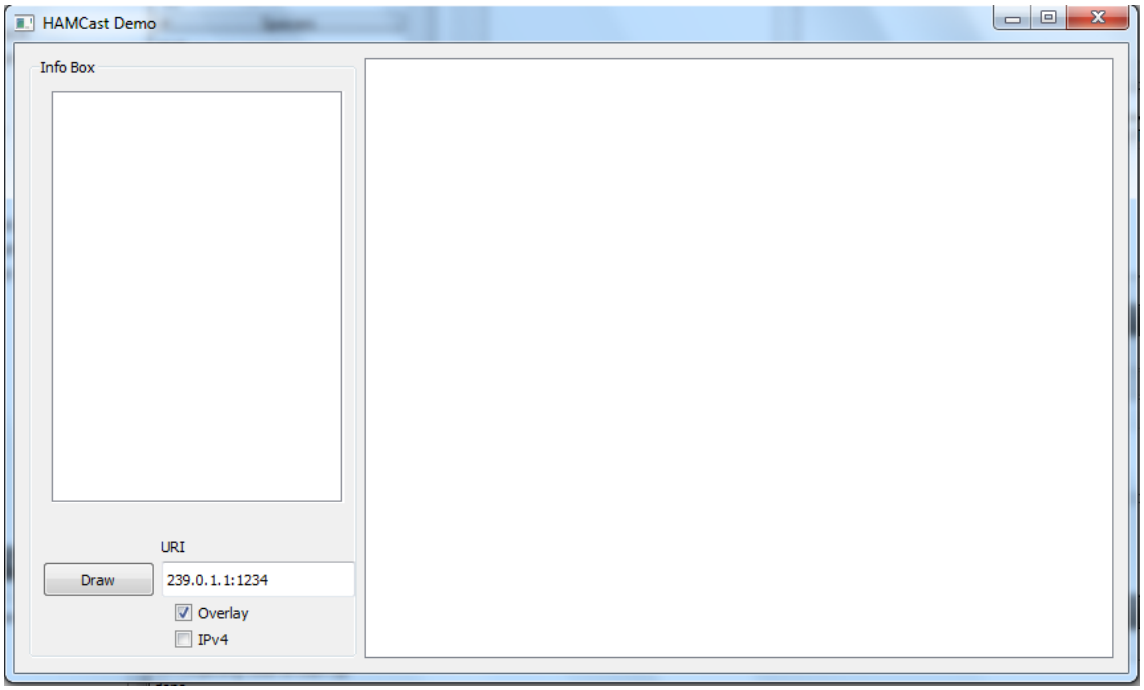

Abbildung 4.2: Monitoring Viewer GUI

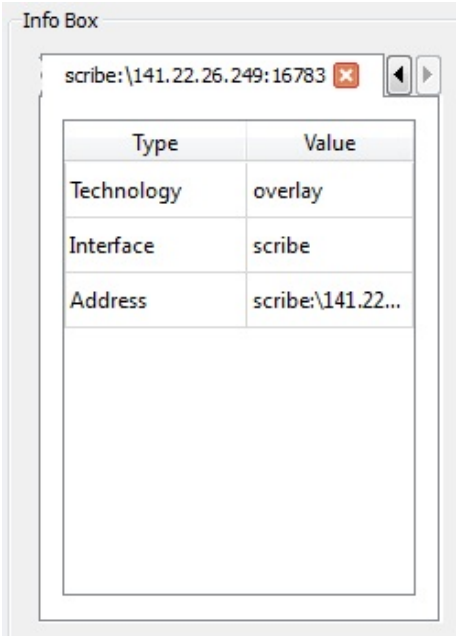

Abbildung 4.3: Tabelle eines Knoten

Abbildung 4.4: Node Item

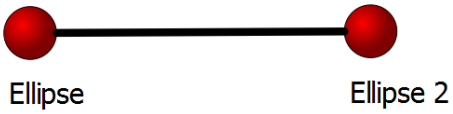

Abbildung 4.5: Edge, die Verbindung zwischen Node Items

4.2.4 Monitoring Daemon

Das im Konzept angesprochene Verfahren zur Kommunikation zwischen dem Monitoring-Daemon und der Monitoring-Software (siehe 4.1) wurde bei der derzeitigen Implementierung noch nicht umgesetzt. Der Grund dafür ist, dass zum Zeitpunkt der Implementation keine stabile Version der HAMcast API zur Verfügung stand. Deshalb nutzt der Monitoring Deamon zur Kommunikation mit dem Monitoring Viewer einen QTcpServer.

Verhalten

Zum Programmstart befindet sich der Monitoring Daemon in einem IDLE Zustand. Das Verhalten (siehe 4.6) des Monitoring Daemon wird durch den Empfang einer Anfragenachricht ausgelöst. Eine Anfragenachricht besteht aus einem Nachrichten Typ und anderen Informationen, die von der Art der Nachricht abhängen. Wird der Nachrichten Typ nicht erkannt, wird eine Error-Message an den anfragenden Host gesendet. Wenn die Nachricht vom Typ getChildrenInfo ist, werden die angefragten Daten ermittelt und an den anfragenden Host zurückgesendet. Der verwendete Nachrichten Typ einer Anfrage- und Antwortnachricht ist identisch.

QTcpServer Konfiguration

Im Codeausschnitt 4.3 ist die Konfiguration für den QTcpServer zu sehen. Mit der Funktion *listen(QHostAddress::Any,port)* wird der QTcpServer so konfiguriert, dass er alle Tcp-Verbindungen, die über den angegebenen Port eingehen, verarbeitet. Das Signal *newConnection()* wird ausgelöst, sobald ein Client sich mit dem QTcpServer verbinden möchte. Der Slot *processConnection()* ist eine Methode, die aufgerufen wird, sobald das Signal *newConnection* ausgelöst wird. Die Methode *processConnection()* dient zur weiteren Verarbeitung der neuen Verbindung.

Verarbeitung eingehender Nachrichten

Die Verarbeitung der eingehenden Anfragenachrichten wird in der Methode 4.5 durchgeführt.

Eine Anfragenachricht besteht aus drei Einträgen, den Nachrichten Typ, der Group ID und der Technologie. Über den Nachrichten Typ wird die Art der Nachricht und die damit verbundenen Informationen definiert. Die Group ID wird benötigt, um Informationen für eine bestimmte Multicastgruppe zu bekommen. Der Technologie Eintrag gibt an, über welches Interface Information gewünscht werden. Eine Switch-Kontrollstruktur wählt über den Nachrichten Typ die Methode aus, welche die Antwort vom Server generiert. Ist kein bekannter Nachrichten Typ vorhanden oder entspricht die Nachricht nicht der erwarteten Länge, wird eine Error-Message an den Monitoring Viewer gesendet.

Die Knoten Informationen, die vom Monitor Daemon an den Monitor Viewer geliefert werden, sind Interface Informationen und die Adressen der Kind-Knoten. Interface Informationen werden über die Methode *get_interfaces()* abgerufen. Die Adressen der Kinder

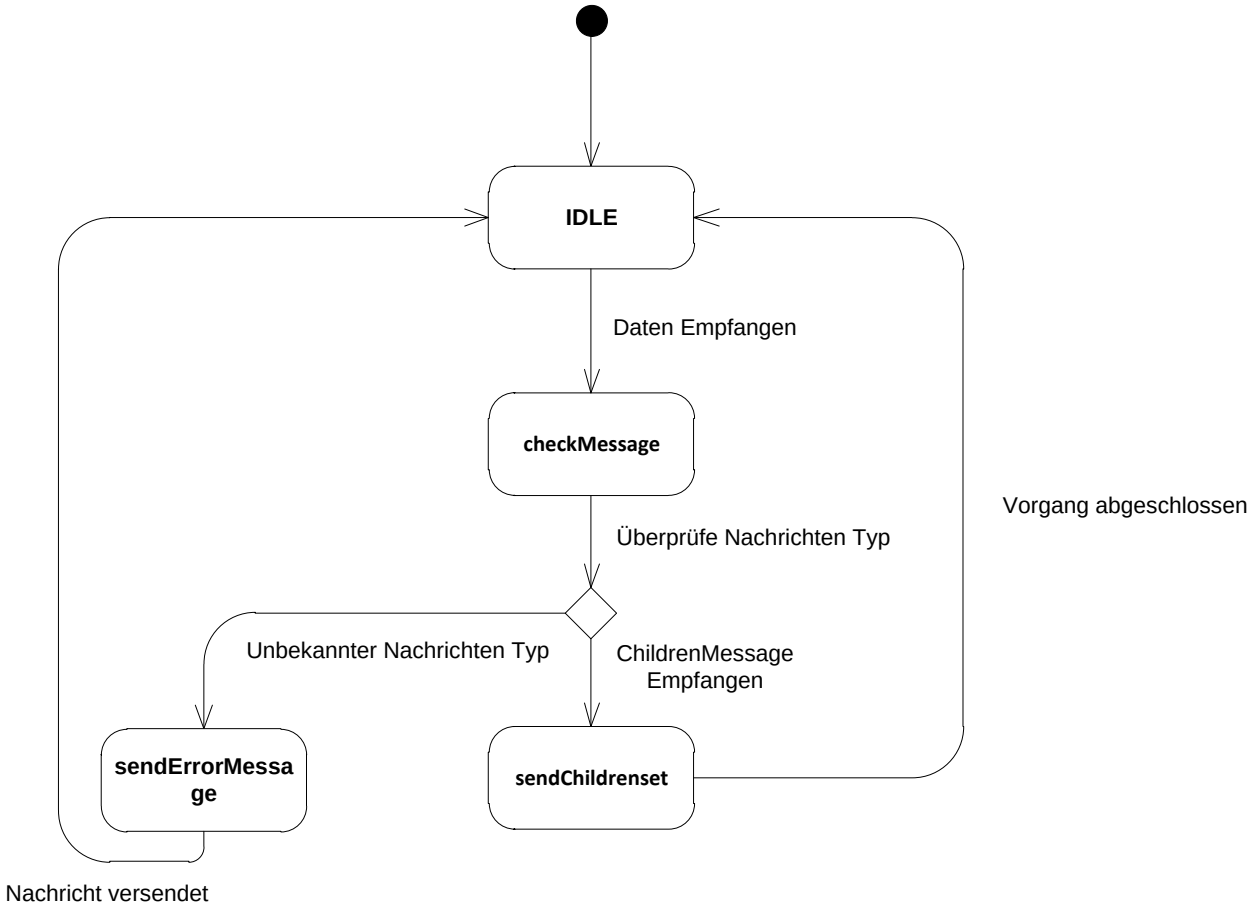

Abbildung 4.6: Verarbeitung empfangener Anfragenachrichten

Listing 4.3: Konfigurieren des QTcpServer

```
0 /* Konstruktor für den QTcpServer*/
  Server::Server(QObject *parent,quint16 port):QTcpServer(parent)
  {
      listen(QHostAddress::Any,port); // Server horcht auf dem
          angegebenen Port
      std::cout << "Server listen on port " << port << std::endl;
          // Ausgabe auf die Console
5     connect(this, SIGNAL(newConnection()), this, SLOT(
          processConnection())); // Signal newConnection() mit dem
          Slot processConnection() verbinden
  }
```

werden über die Methode *childrenSet()* abgerufen. Der Codeauschnitt 4.5 zeigt, wie bei dem Monitor Daemon die Informationen ermittelt werden.

Listing 4.4: Verarbeitung der Anfragenachrichten

```
/* Verarbeitung eingehender Nachrichten */
void Server:: receiveMessage(){
    QTcpSocket* socket = static_cast<QTcpSocket*>(sender());

    QByteArray data = socket->readAll();//lesen der Daten vom
        Socket

                if(data.length < 1 ){ // data länge < 1 sende
                    Error, keine gültige Nachricht
                        sendErrorMessage();
                }
    int c = data[0]; // Nachrichten Typ
    switch (c){
    case 0x2: sendChildrenSetData(data,socket); // 0x2
        getChildrenSet
         break;
    default:std::cout << "unknown Message recived" << std::endl;
         // Typ nicht erkannt sende error Nachricht
    sendErrorMessage();
    }
}
```

Listing 4.5: Ermitteln der Knoten Informationen

```
for(unsigned int i = 0; i < interface.size(); i++){
        prob = interface.at(i);
        .....
            if(technology == tech){
                //Schreibe Daten in den Sendebuffer, mit
                   Seperator ","
                sendData.append(QString::fromStdString(prob.name
                   ));
                sendData.append(",");
                                                          .......
                if(prob.name == "scribe"){
                // Erstelle URI aus Prefix und Group ID
                    const uri u("scribe://"+ip.toStdString());
                    // Aufruf von neighbor_set
                    // liefert uri´s von Kindern wenn vorhanden
                    vec = neighbor_set(prob.id, u);
                }
                else{
                // weitere mögliche Prefixe
                    .....
                }
                // Füge, wenn vorhanden Kinder URIs zu den
                   Sendedaten hinzu
                for(unsigned int n = 0; n < vec.size();n++){
                    uri a =vec.at(n);
                    QString s = a.c_str();

                    std::cout << "uri " << s.toStdString()<< std
                       ::endl;
                    if(n+1 == vec.size()){
                        sendData.append(s);
                    }
                    else{
                        sendData.append(s);
                        sendData.append(",");
                    }
                }
            }
        }
    }
```

4.2.5 Monitoring Viewer

Verhalten

Der Monitoring Viewer besitzt zwei Verhaltensweisen. Die erste Verhaltensweise (siehe 4.7) wird durch den Benutzer ausgelöst und sorgt dafür, dass Anfragenachrichten an die Monitoring Daemons versendet werden. Die zweite Verhaltensweise (siehe 4.8) ist das Zeichnen des Netzwerkgraphen. Sie wird durch den Empfang der Antworten auf eine Anfragenachricht ausgelöst.

Das Versenden der Anfragenachrichten :

Zum Programmstart befindet sich das Programm in einem IDLE Zustand und wartet auf Eingaben des Benutzers oder das Empfangen von Antwortnachrichten. Gibt der Benutzer eine Gruppen ID in das Textfeld ein und betätigt den Draw Button, wird das Verhalten zum Versenden der Anfragenachrichten ausgelöst. Als erstes wird geprüft, ob der Benutzer eine Adresse in das Textfeld geschrieben hat. Ist das der Fall, wird eine Anfragenachricht erstellt und an alle bekannten HAMcast-Hosts versendet.

Das Zeichnen des Netzwerkgraphen :

Nachdem eine Nachricht empfangen wurde, wird geprüft, um was für eine Nachricht es sich handelt. Entspricht der Nachrichten Typ der *getChildrenInfo* Nachricht, werden die Daten analysiert und die Knoten und Kanten Objekte für den Knoten und seine Kinder erstellt. Sind alle erwarteten Antwortnachrichten empfangen, werden die Positionen für die Knoten berechnet und vergeben. Zum Schluss werden alle Objekte zur QGraphicsScene hinzugefügt und gezeichnet.

Das Node Item

Das Node Item wird durch die Funktionen *boundingRect* 4.6 und *paint* 4.7 definiert. Die Funktion *boundingRect* gibt den Bereich an indem das Node Item gezeichnet wird. Zur Berechnung des umgebenden Rechtecks wird die Größe der Ellipse, die gezeichnet werden soll, verwendet. Da die Node Items alle die gleiche Größe besitzen, kann man feste Werte zur Berechnung des Rechtecks benutzen. Das Zeichnen eines Node Items in das Rechteck wird durch die Methode Paint implementiert. Gezeichnet wird eine Ellipse, die mit einem Radial Gradient ausgemalt wird. Wird ein Node Item von einer Maus angeklickt, ist das Flag State_Sunken gesetzt und der Radial Gradient wird mit einer anderen Farbe gezeichnet. Nodes können per Drag and Drop die Position ändern. Aus diesem Grund ist es nötig, wenn zwei Nodes über eine Edge miteinander verbunden sind, die Edge neu zu zeichnen. Das Neuzeichnen der Edge wird durch die Methode *itemChange* veranlasst.

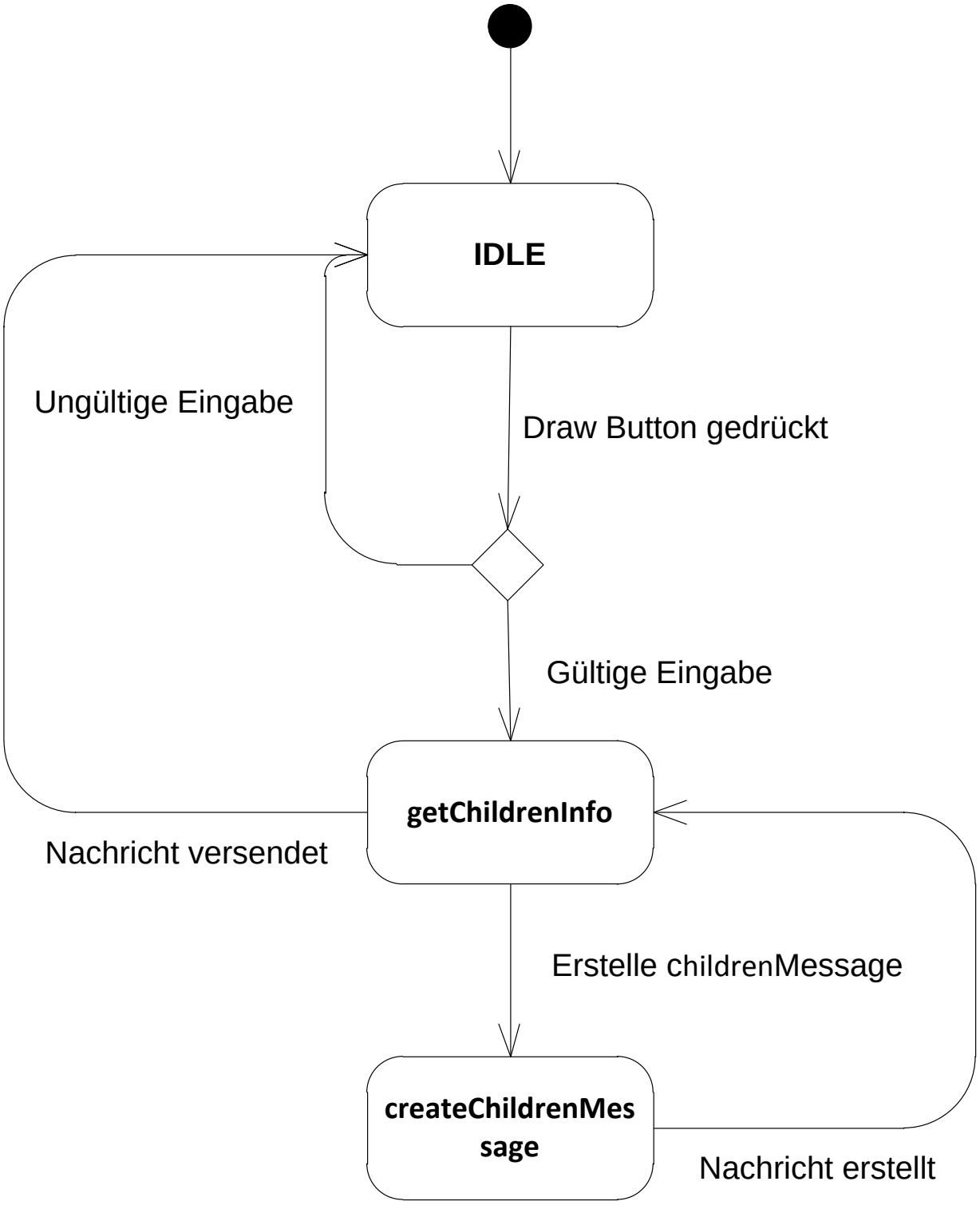

Abbildung 4.7: Versenden der Anfragenachricht

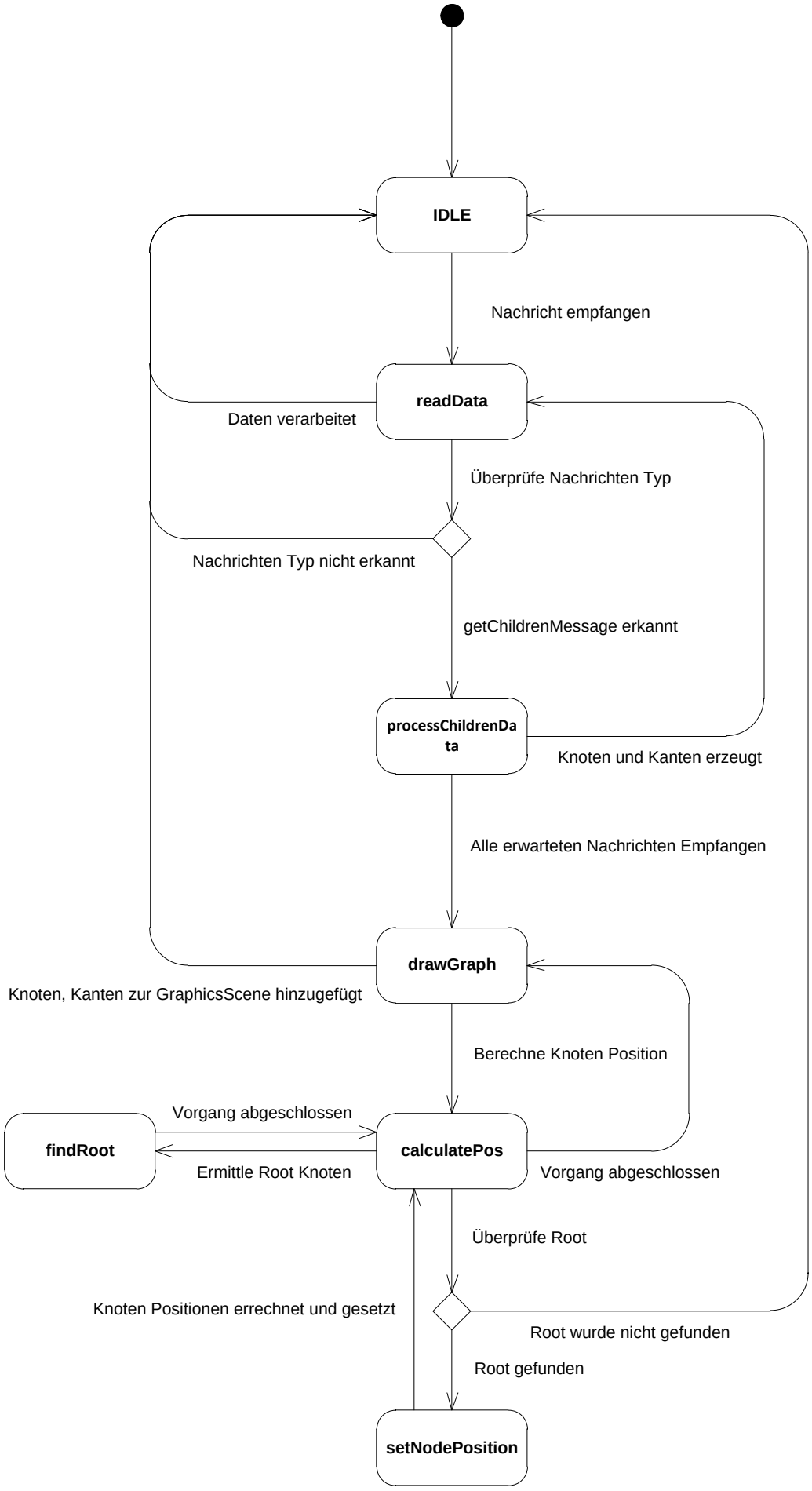

Abbildung 4.8: Zeichnen des Netzwerkgraphen

Listing 4.6: Berechnung des Rechtecks

```
QRectF Node::boundingRect() const
 {
     QRectF rectf =QRectF(-10, -10,
                          20 , 20 );
     return rectf;
 }
```

Listing 4.7: Die Paint Funktion

```
void Node::paint(QPainter *painter, const
   QStyleOptionGraphicsItem *option, QWidget *)
 {

     QRadialGradient gradient(4, 4, 10); // Position und Größe
        des Gradienten festlegen x,y,Radius
     if (option->state & QStyle::State_Sunken) {
         gradient.setColorAt(1, QColor(Qt::red).light(120)); //
            Äußerer berech
         gradient.setColorAt(0, QColor(Qt::darkRed).light(120));
             // Innerer Berech
     } else {
         gradient.setColorAt(0, Qt::red);// Innerer Berech
         gradient.setColorAt(1, Qt::darkRed);// Äußerer berech
     }
     painter->setBrush(gradient);
     painter->setPen(QPen(Qt::black, 0));
     painter->drawEllipse(-10, -10, 20, 20); // Ellipse zeichnen

 }
```

Das Edge Item

Das Edge Item ist eine Linie, die zwei Nodes miteinander verbindet. Als Ausgangspunkt zum Zeichnen des Rechtecks 4.9 wird die Quellkoordinate des Knotens genommen, der als SourceNode beim Erstellen des Edge Objekts übergeben wurde. Die Breite des Rechtecks lässt sich durch die Differenz der x Koordinate von Quell und Zielpunkt bestimmen, äquivalent dazu wird die Höhe berechnet. Dabei muss die Breite der Linie noch berücksichtigt und auf die Länge und Höhe des Rechtecks addiert werden. Die *Paint* Funktion zeichnet eine Edge nur dann, wenn die Node Items einen Mindestabstand zueinander haben. Die Farbe der Linie kann durch das Argument color verändert werden. Übergeben wird dieses Argument beim Erstellen des Edge Objekts. Durch das Argument Style kann die Art, wie eine Line gezeichnet wird, bestimmt werden. Das Argument wird beim Erstellen des Objekts übergeben.

Listing 4.8: Berechnung des Rechtecks

```
0  QRectF Edge::boundingRect() const
   {
       if (!source || !dest)
           return QRectF();

5      qreal penWidth = 3;
       qreal extra = (penWidth) / 2.0;
       //topleft, size //QsizeF w,h
       QRectF rectF = QRectF(sourcePoint, QSizeF(destPoint.x() -
           sourcePoint.x(),destPoint.y() - sourcePoint.y())).
           normalized()                                .adjusted(-extra,
            -extra, extra, extra);
       return rectF;
10 }
```

Berechnung der Knoten Koordinaten

Zur Berechnung der Knoten Koordinaten, wird der vorher erstellte Netzwerkgraph (siehe 4.1.1) rekursiv durchlaufen. Als Startpunkt für die Rekursion wird die Wurzel des Netzwerkgraphen genommen. Der Algorithmus ist im Codeauschitt 4.10 zu sehen.

Der Algorithmus verhält sich wie folgt. Besitzt ein Knoten keine Kinder, wird die Methode beendet. Besitzt ein Knoten ein Kind, so wird es direkt unter den Eltern-Knoten positioniert. Für die Abstände x und y sind default Werte festgelegt. Wenn ein Knoten zwei Kinder besitzt, so werden sie links und rechts unter dem Eltern-Knoten positioniert. Ist die Anzahl der Kinder eines Knotens größer als oder gleich 3, wird die x Koordinate für das erste Kind berechnet und anschließend alle weiteren Kinder rechts neben dem vorherigem Kind platziert. Zur Berechnung der x Koordinate für das Kind, welches als erstes platziert wird, wird die Anzahl der Kinder durch zwei geteilt und von der x Koordinate des Eltern-Knotens subtrahiert. Bei einer großen Anzahl von Knoten auf der selben Ebene, kann es dazu kommen, dass Knoten übereinander positioniert werden. Eine Korrektur durch den Benutzer ist deshalb bei der derzeitigen Implementation nötig.

Listing 4.9: Die Paint Funktion

```
0  void Edge::paint(QPainter *painter, const
       QStyleOptionGraphicsItem *, QWidget *)
   {
       if (!source || !dest)
           return;
                   // Abstand zu gering, zeichne keine Linie
5      QLineF line(sourcePoint, destPoint);
       if (qFuzzyCompare(line.length(), qreal(0.)))
           return;

                   // Tabellen Eintrag makriert, zeichne Rect
10     if(light){
           painter->setPen(QPen(Qt::black, 1, Qt::DashLine));
           painter->drawRect(boundingRect());

       }
15     // Line Farbe und Zeichenart auswählen
       switch(this->style){
       case 1:painter->setPen(QPen(color,3,Qt::SolidLine, Qt::
           RoundCap, Qt::RoundJoin));
           break;
       case 2:painter->setPen(QPen(color,3,Qt::DashLine, Qt::
           RoundCap, Qt::RoundJoin));
20         break;
       .......
       default :painter->setPen(QPen(color,3,Qt::SolidLine, Qt::
           RoundCap, Qt::RoundJoin));
           break;
                   }
25                 // Line Zeichnen
       painter->drawLine(line);
   }
```

Listing 4.10: Berechnung der Knoten Positionen

```
0       int len = e.size();
        // keine Kinder Vorhanden
        if(len == 0) return 1;
        // Ein Kind vorhanden, positioniere Knoten unter dem Eltern-
           Knoten
        if(len == 1){
5               e.at(0)->destNode()->setPos(x,y);
                setNodePosition(e.at(0)->destNode()->childEdges(),x
                   ,(y+ydistance),distance-20);
        }
        // Zwei Kinder vorhanden, positioniere Kinder links und
           rechts unter dem Eltern-Knoten
        if(len == 2){
10              e.at(0)->destNode()->setPos(x+distance,y);
                setNodePosition(e.at(0)->destNode()->childEdges(),(x
                   +distance),(y+ydistance),distance-20);
                e.at(1)->destNode()->setPos(x-distance,y);
                setNodePosition(e.at(1)->destNode()->childEdges(),(x
                   -distance),(y+ydistance),distance-20);
        }
15      else{
        // N Kinder vorhanden, positioniere Kinder von links nach
           rechts
            x -= (len / 2) *distance;
            for (int i =0; i < len; i++){
                        e.at(i)->destNode()->setPos(x,y);
20                      setNodePosition(e.at(i)->destNode()->
                            childEdges(),x,y+ydistance,distance-20);
                        x +=distance;
            }
        }
        return 1;
```

5 Zusammenfassung und Ausblick

Die in dieser Arbeit vorgestellte Visualisierungs- und Monitoring-Software wurde entwickelt, um Topologie-Informationen über ein HAMcast-Netzwerk zu ermitteln. Die Ermittelung des Netzwerkgraphen für die Application-Layer-Multicast Knoten, sowie deren Interface Informationen und Visualisierung, sind mit der derzeitigen Implementierung möglich. Um einen vollständigen Überblick über das Netzwerk erhalten zu können, müssen weitere Funktionalitäten hinzugefügt werden.

Da sich die HAMcast API ebenfalls noch in der Entwicklung befindet, war es eine besondere Herausforderung die entwickelten Konzepte umzusetzen. So gibt es bisher noch keinen Mechanismus, um den IP-Multicast Forwarding Tree aufzudecken.

Der Fokus liegt nun in der Weiterentwicklung, eines Mechanismus zum Ermitteln des IP-Multicast Forwarding Trees und den Algorithmus zur Positionierung von Knoten zu verbessern. Weitere noch zu implementierende Features sind das Versenden von Anfragen über einen HAMcast Multicast-Socket und die Ermittelung und Visualisierung aller Multicast-Gruppen und existierender Interfaces eines HAMcast-Hosts.

Weitere Features wie Paketstatistiken, die Ermittelung der Bandbreite von Hosts und ein Multicast Ping, könnten die Software noch nützlicher für die Konfiguration und das Management von HAMcast Netzwerken machen.

Literaturverzeichnis

[1] ADAMS, A. ; NICHOLAS, J. ; SIADAK, W.: Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised) / IETF. January 2005 (3973). – RFC

[2] ALMEROTH, K.C.: The evolution of multicast: from the MBone to interdomain multicast to Internet2 deployment. In: *IEEE Network* 14 (2000), Jan/Feb, Nr. 1, S. 10–20. – ISSN 0890-8044

[3] CAIN, B. ; DEERING, S. ; KOUVELAS, I. ; FENNER, B. ; THYAGARAJAN, A.: Internet Group Management Protocol, Version 3 / IETF. October 2002 (3376). – RFC

[4] CORPORATION, Nokia: *The Graphics View Framework*. 2010. – URL `http://doc.qt.nokia.com/4.6/graphicsview.html#graphicsview`. – abgerufen 16. August 2010

[5] CORPORATION, Nokia: *Qt UI-Framework*. 2010. – URL `http://qt.nokia.com/`. – abgerufen 16. August 2010

[6] DEERING, Stephen E. ; CHERITON, David R.: Multicast Routing in Datagram Internetworks and Extended LANs. In: *ACM Trans. Comput. Syst.* 8 (1990), Nr. 2, S. 85–110. – ISSN 0734-2071

[7] DIOT, Christophe ; LEVINE, Brian N. ; LYLES, Bryan ; KASSEM, Hassan ; BALENSIEFEN, Doug: Deployment Issues for the IP Multicast Service and Architecture. In: *IEEE Network Magazine* 14 (2000), Nr. 1, S. 78–88

[8] FENNER, B. ; HANDLEY, M. ; HOLBROOK, H. ; KOUVELAS, I.: Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised) / IETF. August 2006 (4601). – RFC

[9] FENNER, W. ; CASNER, S.: A "traceroute"facility for IP Multicast / IETF. July 2000. – IETF Internet Draft

[10] HOLBROOK, H. ; CAIN, B.: Source-Specific Multicast for IP / IETF. August 2006 (4607). – RFC

[11] HOSSEINI, Mojtaba ; AHMED, Dewan T. ; SHIRMOHAMMADI, Shervin ; GEORGANAS, Nicolas D.: A Survey of Application-Layer Multicast Protocols. In: *IEEE Communications Surveys & Tutorials* 9 (2007), Nr. 3, S. 58–74

[12] HUFFAKER, Bradley ; NEMETH, Evi ; CLAFFY k: *Otter: A general-purpose network visualization tool*. 1999. – URL `http://www.caida.org/tools/visualization/otter/paper/`. – abgerufen 31. August 2010

[13] HUFFAKER, Bradley ; NEMETH, Evi ; CLAFFY k: *Tools to Visualize the Internet Multicast Backbone*. 1999. – URL `http://www.caida.org/publications/papers/1999/manta/manta.html`. – abgerufen 31. August 2010

[14] JACOBSON, Van: *Mrinfo*. – URL `http://svnet.u-strasbg.fr/mrinfo/index.html`. – abgerufen 2. September 2010

[15] JIN, Xing ; CHENG, Kan-Leung ; CHAN, S.-H. G.: Island multicast: combining IP multicast with overlay data distribution. In: *Trans. Multi.* 11 (2009), Nr. 5, S. 1024–1036. – ISSN 1520-9210

[16] LU, Shaofei ; WANG, Jianxin ; YANG, Guanzhong ; GUO, Chao: SHM: Scalable and Backbone Topology-Aware Hybrid Multicast. In: *16th Intern. Conf. on Computer Communications and Networks (ICCCN'07)*, August 2007, S. 699–703. – ISSN 1095-2055

[17] MAKOFSKE, David B. ; ALMEROTH, Kevin C.: *MHealth: A Real-Time Multicast Tree Visualization and Monitoring Tool*. – URL `http://imj.ucsb.edu/mhealth/mhealth_nossdav.pdf`. – abgerufen 31. August 2010

[18] MAKOFSKE, David B. ; ALMEROTH, Kevin C.: Real-Time Multicast Tree Visualization and Monitoring. In: *Software–Practice & Experience* 30 (2000), S. 1047–1065

[19] MALKIN, Gary S.: Traceroute Using an IP Option / IETF. January 1993 (1393). – RFC

[20] MEILING, Sebastian ; CHAROUSSET, Dominik ; SCHMIDT, Thomas C. ; WÄHLISCH, Matthias: System-assisted Service Evolution for a Future Internet – The HAMcast Approach to Pervasive Multicast. In: *Proc. of IEEE GLOBECOM 2010, Workshop MCS 2010*. Piscataway, NJ, USA : IEEE Press, December 2010

[21] OTT, J. ; CHESTERFIELD, J. ; SCHOOLER, E.: RTP Control Protocol (RTCP) Extensions for Single-Source Multicast Sessions with Unicast Feedback / IETF. February 2010 (5760). – RFC

[22] PROJECT, The V.: *VLC: open-source multimedia framework*. 2010. – URL `http://www.videolan.org/`. – abgerufen 17. August 2010

[23] SARAC, Kamil ; ALMEROTH, Kevin C.: Tracetree: A Scalable Mechanism to Discover Multicast Tree Topologies in the Internet. In: *IEEE/ACM TRANSACTIONS ON NETWORKING* 12 (2004), Nr. 5, S. 795 – 808. – ISSN 1063-6692

[24] TEAM, The G.: *The GTK+ Projekt*. 2008. – URL `http://www.gtk.org/`. – abgerufen 16. August 2010

[25] VIDA, R. ; COSTA, L.: Multicast Listener Discovery Version 2 (MLDv2) for IPv6 / IETF. June 2004 (3810). – RFC

[26] WÄHLISCH, Matthias ; SCHMIDT, Thomas C.: Between Underlay and Overlay: On Deployable, Efficient, Mobility-agnostic Group Communication Services. In: *Internet Research* 17 (2007), November, Nr. 5, S. 519–534. – URL `http://www.emeraldinsight.com/10.1108/10662240710830217`

[27] WÄHLISCH, Matthias ; SCHMIDT, Thomas C.: Multicast Routing in Structured Overlays and Hybrid Networks. In: SHEN, Xuemin (Hrsg.) ; YU, Heather (Hrsg.) ; BUFORD, John (Hrsg.) ; AKON, Mursalin (Hrsg.): *Handbook of Peer-to-Peer Networking*. New York Heidelberg : Springer, January 2010, S. 897–932. – URL `http://www.springerlink.com/content/x654q5507t418788/`

[28] WÄHLISCH, Matthias ; SCHMIDT, Thomas C. ; VENAAS, Stig: A Common API for Transparent Hybrid Multicast / individual. URL `http://tools.ietf.org/html/draft-waehlisch-sam-common-api`, July 2010 (04). – IRTF Internet Draft – work in progress

[29] WAITZMAN, D. ; PARTRIDGE, C. ; DEERING, S.: Distance Vector Multicast Routing Protocol / IETF. November 1988 (1075). – RFC

[30] ZHANG, Beichuan ; WANG, Wenjie ; JAMIN, Sugih ; MASSEY, Daniel ; ZHANG, Lixia: Universal IP multicast delivery. In: *Computer Networks* 50 (2006), Nr. 6, S. 781–806. – ISSN 1389-1286

Abbildungsverzeichnis

Quellcodeverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 7. September 2010	
Ort, Datum	Unterschrift