Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sebastian Wölke

Konzeption und Implementierung eines dynamisch konfigurierbaren und mehrfach instantiierbaren IGMP/MLD Proxy Daemons

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Wölke

Konzeption und Implementierung eines dynamisch konfigurierbaren und mehrfach instantiierbaren IGMP/MLD Proxy Daemons

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Abgegeben am 3. November 2011

Sebastian Wölke

Thema der Bachelorarbeit

Konzeption und Implementierung eines dynamisch konfigurierbaren und mehrfach instantiierbaren IGMP/MLD Proxy Daemons

Stichworte

Multicast-Proxy, IGMP, MLD, PMIPv6

Kurzzusammenfassung

IGMP/MLD Proxys bieten die Möglichkeit, Multicast-Netzwerke effizient mit größeren Multicast-Infrastrukturen zu verbinden. Im Gegensatz zu Multicast-Routern sind sie leichtgewichtiger und verzichten auf die Unterstützung von Routing-Protokollen wie PIM oder DVMRP. Ein Anwendungsbeispiel liegt im Bereich der PMIPv6-Domänen, um mobile Endgeräte transparent mit Multicast-Diensten zu versorgen. In dieser Bachelor Arbeit wird ein IGMP/MLD Proxy entwickelt. Hierfür wird der Aufbau, die Funktionsweise und Implementierung von IGMP/MLD Proxys sowie die Integration in PMIPv6 vorgestellt.

Title of the paper

Design and implementation of a dynamically configurable and multiple instantiable IGMP/MLD Proxy demon

Keywords

multicast proxy, IGMP, MLD, PMIPv6

Abstract

IGMP/MLD proxies offer the possibility to efficiently combine multicast networks with a larger multicast infrastructure. In contrast to multicast routers proxies are lightweight and do not need to support multicast routing protocols such as PIM or DVMRP. An usage example is in the PMIPv6 domains to provide transparent multicast services for mobile nodes. This bachelor thesis describes development and implementation of an IGMP/MLD proxy, as well as its integration into the PMIPv6 protocol.

Inhaltsverzeichnis

1 Einführung

Anwendungen wie Videokonferenzen, Internet Radio oder IPTV basieren auf Gruppenkommunikationen. Es werden die selben Daten zur mehreren Empfängern gesendet. Eine effiziente und skalierbare Gruppenkommunikation ist Mulitcast, welches darauf ausgelegt, ist redundanten Datenverkehr zu vermeiden. Sollen Daten für Multicast-Gruppen in einer größeren Netzwerk-Infrastrukur verteilt werden, können Multicast-Router eingesetzt werden. Multicast-Router versuchen hierbei zum Beispiel, die Gruppendaten über einen möglichst kurzen Weg zum Ziel zu bringen. Ist ein derartiges dynamisches Routing nicht erforderlich, weil es zum Beispiel nur einen Weg existiert, können auch Multicast-Proxies eingesetzt werden.

Multicast wird nicht nur von stationären Rechnern verwendet, sondern kann auch von mobilen Endgeräten genutzt werden. Ein hierfür notwendiges Protokoll ist PMIPv6 (Proxy Mobile IPv6 [GLD+08]). Es verbindet mobile Endgeräte mit dem Internet und sorgt beim Wechseln in ein anderes Netzwerk dafür, dass dies transparent für die Transprotokolle geschieht. Durch die Erweiterung PMIPv6 mit Multicast Listener Support [SWK11] wird PMIPv6 multicastfähig. So ist es dann zum Beispiel möglich, über ein moderneres Mobiletelefon per Multicast an einer Videokonferenz teilzunehmen. Auch hier werden Multicast-Proxies eingesetzt.

1.1 Zielsetzung

Im Rahmen dieser Bachelorarbeit soll ein Multicast-Proxy entworfen und implementiert werden. Der Proxy wird die Gruppenverwaltungprotokolle IGMP (IPv4) und MLD (IPv6) unterstützen. Außerdem wird der Proxy mehrfach instatiierbar und dynamisch konfigurierbar sein, was Anforderungen von PMIPv6 mit Multicast sind. Er soll unter Linux lauffähig sein und deren Kerneltabellen nutzen.

1.2 Gliederung dieser Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Im zweiten Kapitel werden die Multicast-Grundlagen beschrieben und das Prinzip erläutert. Hierfür wird die notwendige Gruppenverwaltung, der Aufbau des Multicast-Proxy und die zusätzlichen Anforderungen durch PMIPv6 mit Multicast im Detail dargestellt.

Anschließend werden in Kapitel 3 die Linux-spezifischen Multicast-Funktionalitäten dargelegt und als Basis für das Kapitel 4 Entwurf genutzt. Für den Entwurf wird zunächst die Zielsetzung dieser Arbeit erläutert und gegenwärtige Implementierungen auf Rückschlüsse für die eigenen Implementierung hin analysiert. Daraufhin werden die Problemfelder sensibilisiert und der eigene Entwurfsvorgang beschrieben. Kapitel 5 zeigt die wichtigen Aspekte der Implementierung, beschreibt die verwendete Testumgebung und den Testaufbau. Abschließend wird die Arbeit zusammenfassend bewertet und Möglichkeiten zur Weiterentwicklung aufgezeigt.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Eigenschaften von Multicast erläutert. Es wird das Nachrichtenformat und das IP-Adressformat der Multicast-Kommunikation dargelegt, die später im Detail implementiert werden. Anschließend wird auf die Gruppenverwaltungsprotokolle eingegangen, da sie einen wichtigen Teil dieser Arbeit einnehmen werden. Hierfür werden zusätzlich Szenarios der Kommunikation zwischen Router und Hosts beschrieben. Danach wird der Aufbau des Multicast-Proxys erläutert und die Grundlagen der Gruppenverwaltung zusammengeführt. Zum Schluss wird auf mobile Gruppenverwaltung (PMIPv6 mit Multicast) eingegangen, um die Erweiterungen am Multicast-Proxy zu erklären.

2.1 Multicast

Multicast ist eine Gruppenkommunikation, die in verschiedenen Technologien eingesetzt wird. So gibt es zum Beispiel Multicast für Ethernet und für das Internet Protokoll.

Weiterhin gibt es verschiedene Arten von Multicast zum Beispiel ASM (Any Source-Multicast) und SSM (Source Specific Multicast). Sendet ein Sender S Daten an eine Gruppe G, so wird bei SSM die Gruppe mit dem zugehörigen Sender abonniert (Sender S, Gruppe G). Bei ASM wird eine Gruppe in der Form (Sender *, Gruppe G) abonniert, so werden nur bestimmte Sender einer Gruppe akzeptiert.

2.2 IP-Multicast

IP-Multicast [DC90] ermöglicht eine skalierende Übertragung von IP-Datenpaketen an eine Gruppe. So untersteht diese Kommunikation einigen Anforderungen. Es muss gewährleistet sein, dass Gruppendaten nur in Netzwerke geroutet werden, in denen es auch Empfänger gibt, was sonst globalem Broadcast entsprechen würde und nicht skaliert. So wird ein Publish/Subscription Verfahren eingesetzt, wodurch der einzelne Empfänger entscheidet, welche Gruppen er abonniert. Und dadurch bestimmt, welche Daten in sein Netzwerk geroutet werden. Für ein skalierendes Abonnieren von Gruppe dürfen die Sender nicht durch Abonnierungen belastet werden. Aus diesem Grund werden die Gruppen beim nächsten Router abonniert. So wird die Subscription in einem Netzwerk als *Multicast-Zustand* bezeichnet. Dieser Multicast-Zustand wird nun über das Routing Protokoll von Router zu Router bis zum ersten Router des Senders weitergereicht, wodurch der Verteilbaum für die Gruppendaten invers aufgebaut wird. Weiterhin muss der Sender die Daten für die Gruppe nur einmal verschicken können, unabhängig von der Empfängeranzahl. Würde der Sender die Gruppendaten für jeden Empfänger separat verschicken, stiege der Bandbreitenbedarf und die Latenz der Daten durch das serielle Versenden mit jedem Empfänger. Stattdessen werden Gruppendaten an Verzweigungen des Multicast-Verteilbaums repliziert.

Bei der Übertragung wird nach dem Besteffort Verfahren gearbeitet, so gibt es keine Garantie, dass die Datenpakete bei allen Gruppenmitgliedern ankommen. Würde es wie

bei TCP eine Fehlerkorrektur geben, käme es zur einer ACK-Implusion [WZ01]. Gruppen können bei IP-Multicast zu einem beliebigen Zeitpunkt betreten und verlassen werden. Des Weiteren gibt es keine Begrenzung der Gruppengröße und Hosts können eine beliebige Anzahl von Gruppen abonnieren.

Als Transportprotokoll muss ein verbindungsloses Protokoll eingesetzt werden. Das ist notwendig, denn das Aufbauen von Verbindungen zu jedem Gruppenmitglied würde nicht skalieren. So wird UDP zum Übertragen der Gruppendaten eingesetzt. Multicast-Pakete unterscheiden sich von anderen UDP Paketen dadurch, dass sie an eine Multicast-Gruppen adressiert sind. Der Adressaufbau sieht wie folgt aus:

IPv4 Adressierung

Der Adressraum für die IPv4 Multicast-Pakete gehört zu der ehemaligen Netzklasse D, also zu dem IP-Adress-Bereich 224.0.0.0 bis 239.255.255.255, davon sind die Adressen 224.0.0.0 bis 224.255.255.255 unter anderen für Routingprotokolle belegt [IAN11a]. Einen Auszug der belegten Multicast-Adressen zeigt Tabelle 1.

Gruppenadresse	Beschreibung
224.0.0.0	Base Address (Reserved)
224.0.0.1	All Systems on this Subnet
224.0.0.2	All Routers on this Subnet
224.0.0.4	DVMRP Routers
224.0.0.13	All PIM Routers
224.0.0.22	IGMP (alle IGMPv3-Router)

Tabelle 1: Auszug von reservierten IPv4 Multicast-Adressen

IPv6 Adressierung

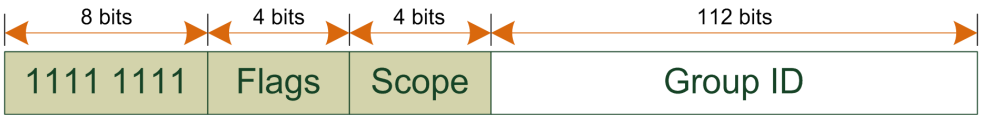


Abbildung 1: IPv6 Multicast Adressformat

Der Adressbereich für die IPv6 Multicast-Pakete ist FF00::/8 [HD06] (siehe Abbildung 1). Die acht höchsten Bits, belegt mit FF, definieren eine Multicast-Adresse. Zusätzlich ist die Multicast-Adressstruktur in die Blöcke Flags, Scope und Group ID unterteilt. Die Flags und der Scope (Reichweite) sind wie folgt definiert:

Flags:	000T	ist T gleich 0 so handelt es sich um eine permanete Gruppe, sonst nicht
	0RPT	für Details der Flags R und P siehe RFC 3306 [HT02] und RFC 3956 [SH04]
Scope:	1	node-local scope (Paket verlässt das Netzwerk-Interface nicht)
	2	link-local scope (werden nicht geroutet)
	5	site-local scope
	8	organization-local scope
	E	global scope

Die permanenten Gruppen sind unter anderem für Routingprotokolle belegt [IAN11b]. Ein Teil dieser Adressen werden in unterschiedlichen Scopes eingesetzt (siehe Tabelle 2).

Gruppenadresse	**Beschreibung**
FF0X::	Reserved Multicast Address
FF0{1, 2}::1	All Nodes Address
FF0{1, 2, 5}::2	All Routers Address
FF02::4	DVMRP Routers
FF02::D	All PIM Routers
FF02::16	All MLDv2-capable routers

Tabelle 2: Auszug von reservierten IPv6 Multicast Adressen

2.3 Gruppenverwaltung

Eine Gruppenverwaltungen beschreibt das Verhalten zwischen Host und Router. Wobei ein Router auch den Host-Teil der Gruppenverwaltung nutzt, um seinen Multicast-Zustand zum nächsten Router zu aggregieren, wodurch sich der Multicast-Verteilbaum aufbaut.

Als Gruppenverwaltungsprotokoll ist für IPv4 das IGMP (Internet Group Management Protocol) und für IPv6 das Protokoll MLD (Multicast Listener Discovery) standardisiert. Mit diesen Protokollen werden ausschließlich Gruppenmitgliedschaften verwaltet und keine Gruppendaten versendet. IGMP ist ein eigenständiges Protokoll mit der IP-Protokollnummer 2. MLD ist das IPv6 Äquivalent zu IGMP und ist in ICMPv6 [CDG06] eingebettet. Für IPv4 existieren die Versionen IGMPv1 [Dee89], IGMPv2 [Fen97] und IGMPv3 [CDK+02]. Für IPv6 existieren die Versionen MLDv1 [DFH99] und MLDv2. Tabelle 3 zeigt, welche IGMP/MLD Versionen funktional analog sind.

IGMP und MLD Protokolle definieren Nachrichten zum Betreten und Verlassen von Multicast-Gruppen, sowie Nachrichten für Router zum Aktualisieren der Multicast-Zustände im Netzwerk. Darauf aufbauend wird die Kommunikation zwischen Host und Router beschrieben. Zusätzlich erläutern diese Protokolle, wie abonnierte Multicast-Gruppen von Routern gehandhabt und verwaltet werden. Die Abläufe im Router und beim

IPv4 Version	Zugehörige RFC	IPv6 Version	Zugehöriger RFC
IGMPv1	RFC 1112	–	–
IGMPv2	RFC 2236	MLDv1	RFC 2710
IGMPv3	RFC 3376	MLDv2	RFC 3810

Tabelle 3: Gruppenverwaltungen und deren Äquivalenzen

Host sind neben den eventbasierten Ereignissen auch zeitgesteuert, so wird auch eine Anzahl von Zeitintervallen und Zählern definiert. Die folgende Beschreibung des Host- und Router-Teils ist ist für IGMPv2 und MLDv1 gültig.

Host-Teil

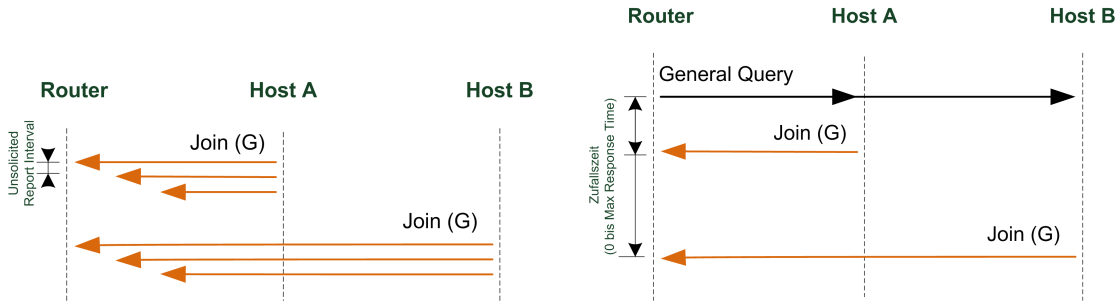
Zum Empfangen von Gruppendaten abonniert der Host die jeweilige Gruppe mit einer Join-Nachricht. Um die Gruppe zu verlassen, sendet der Host eine Leave-Nachricht. Zusätzlich reagiert der Host auf Anfragen vom Router. Sendet der Router einen General Query, antwortet der Host mit einer Join-Nachricht für jede abonnierte Gruppe. Sendet der Router einen Group Specific Query, antwortet der Host mit einer Join-Nachricht für die angefrage Gruppe, falls er sie abonniert hat.

Router-Teil

Wenn der Router eine Join-Nachricht empfängt, speichert er die Gruppenmitgliedschaft für die jeweilige Gruppe und Subnetzwerk für ein bestimmtes Zeitintervall, das von einem Gruppentimer verwaltet wird. Ist der Gruppentimer abgelaufen, wird die Gruppe wieder aus der Gruppenverwaltung gelöscht. Dies ist notwendig, denn Leave-Nachrichten von Hosts können beschädigt werden, verloren gehen oder vom Host gar nicht abgeschickt werden. Allerdings kann es dazu führt, dass Gruppendaten in Netzwerke geroutet werden, für die sich kein Host mehr interessiert.

Der Router versendet periodisch General Queries, woraufhin Hosts mit Join-Nachrichten für alle ihre abonnierten Gruppen antworten. Hierdurch wird der Gruppentimer aktualisiert und ein Löschen aus der Gruppenverwaltung verhindert. Router die General Queries versenden werden als Querier bezeichnet. Existieren in einem Netzwerk mehr als ein Querier, wird eine Querier Election durchgeführt. Der Querier mit der niedrigsten IP-Adresse übernimmt ausschließlich die Aufgabe der Gruppenverwaltung.

Empfängt der Router eine Leave-Nachricht, sendet er ein Group-Specific Query für die jeweilige Gruppe, um zu prüfen, ob andere Host interesse an der Gruppe haben. Antwortet kein Host mit einer Join-Nachricht, kann die Gruppe aus der Gruppenverwaltung gelöscht werden.



(a) Host A und B betreten Gruppe G

(b) Beide Hosts Antworten auf den General Query

Abbildung 2: Betreten und Bestätigen der Gruppe

Zeitintervalle und Zähler

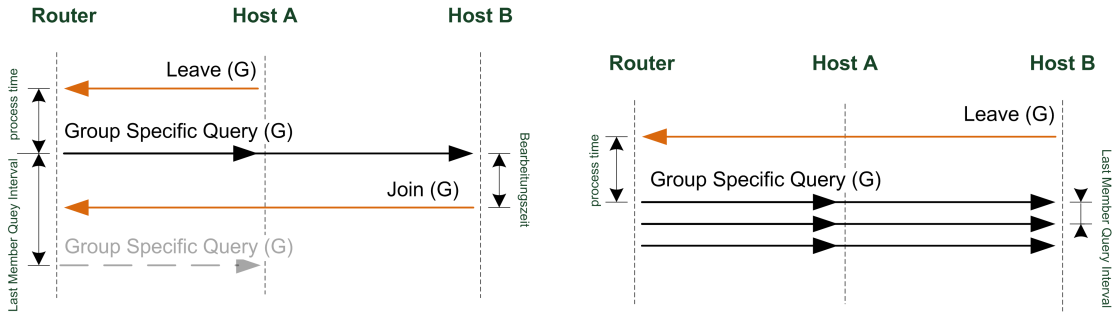
Die folgenden Zeitintervalle und Zähler sind frei wählbar und können an die jeweilige Netzwerkinfrastruktur angepasst werden.

- Sendet der Host eine Join-Nachricht zum Abonnieren einer Multicast-Gruppe, wird die Join-Nachricht mehrmals (Robustness Variable) in einem bestimmten Abstand (Unsolicited Report Interval) wiederholt, um einen möglichen Paketverlust zu kompensieren.
- Empfängt der Host einen General- oder einen Group-Specific Query, antwortet der Host erst nach einer zufälligen Wartezeit (zwischen 0 und Max Response Time), um den *Link-Stress Peak* abzuschwächen.
- Ein Group-Specific Query wird mehrmals (Last Member Query Count) in einem bestimmten Abstand (Last Member Query Interval) wiederholt.

Der Ablauf einer Kommunikation zwischen Host und Router, mit zugehören Zeitintervallen und Zählern, wird im folgenden Szenario veranschaulicht.

IGMPv2/MLDv1 Szenario

1. Abbildung 2a zeigt, wie Host A und Host B die Multicast Gruppe G betreten. Beide Hosts senden aufgrund ihrer Robustheitsvariable zwei zusätzliche Join-Nachrichten im Abstand vom *Unsolicited Report Interval*.
2. In Abbildung 2b empfangen beide Hosts einen General Query und senden daraufhin eine Join-Nachricht, nach einer zufällig bestimmten Zeit (0 bis *Max Response Time*), zurück.



(a) Host A verlässt die Gruppe G (b) Der letzte Host verlässt die Gruppe G

Abbildung 3: Verlassen der Gruppen und die Reaktion vom Router

3. In Abbildung 3a verlässt Host A die Gruppe G. Der Router sendet daraufhin einen Group-Specific Query, um zu prüfen, ob weitere Hosts in diesem Netzwerk die Gruppe G abonniert haben. Nachdem der Router die Join-Nachricht von Host B empfangen hat, ist das Senden von weiteren Group-Specific Queries überflüssig.

4. Nun verlässt auch Host B die Gruppe G (Abbildung 3b). Daraufhin sendet der Router in Höhe vom *Last Member Query Count* Group-Specific Queries, was per Default der Robustheitsvariable entspricht. Da kein Host antwortet, kann der Router die Gruppe aus der Gruppenverwaltung löschen.

Nachrichtenformat

Die IGMPv2 [MLDv1] Gruppenverwaltungs-Nachrichten sind 64 Bit [160 Bit] groß (siehe Abbildung 4).

Type legt den Typ der Nachricht fest (zum Beispiel Join-/ Leave-Nachricht).

Max Resp Time wird nur für den General Query genutzt und definiert die maximale Antwortzeit der Hosts.

Checksum ist die Internet Prüfsumme, die auch im IPv4 Header genutzt wird. Anhand der Prüfsumme kann geprüft werden, ob die Nachricht fehlerfrei ist. Fehlerbehaftete Nachrichten werden verworfen.

Group Address enthält die Multicast IP-Adresse.

Der Aufbau der Gruppenverwaltungs-Nachrichten wird in Tabelle 4 beschrieben. Alle Verwaltungsnachrichten werden unter IPv4 mit einer TTL (oder bei IPv6 Hop Limit) von eins versendet. Außerdem werden IPv6 Nachrichten nur an Link-local Adresse versendet. Auf diese Weise wird sichergestellt, dass diese Pakete nicht das Subnetzwerk verlassen und keine Gruppenverwaltungen andere Subnetzwerke manipulieren.

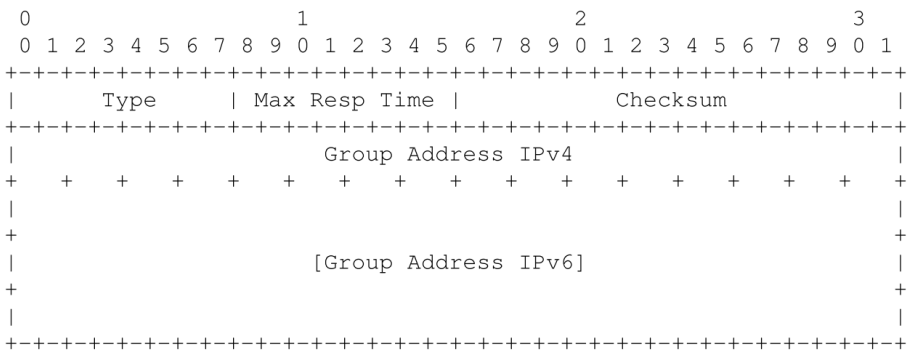


Abbildung 4: IGMPv2 Paketformat

Feld	Report	Leave [Done]	General Query	G. Specific Query
Type	0x16 [131]	0x17 [132]	0x11 [130]	0x11 [130]
Max Resp Time	0	0	Query Response Interval	0
Group Address	Gruppe G	Gruppe G	0	Gruppe G
IP-Destination	Gruppe G	224.0.0.2 [FF02::2]	224.0.0.1 [FF02::1]	Gruppe G

Tabelle 4: IGMPv2 Nachrichten [MLDv1 Anpassung]

IGMPv3/MLDv2

Bei der Multicast Gruppenverwaltung wird, wie schon in Abschnitt 2.3 erläutert, zwischen ASM (Any Source Address) und SSM (Source Specific Multicast) unterschieden. Bei ASM handelt es sich um die Gruppenverwaltung, wie sie für IGMPv2 beschrieben wurde. SSM bietet eine Erweiterung, bei der es möglich ist, auf Quelldressen zu filtern. So wird zum Beispiel beim Betreten einer Gruppe durch das Senden eines Reports eine Quellfilter-Liste mit übergeben. Hierfür stehen ein INCLUDE und ein EXCLUDE als Filtermodus zur Verfügung, was einer White- und einer Blacklist entspricht. Aufgrund dieser Filterlisten sind Router in der Lage Gruppendaten-Pakete zu filtern, die sonst erst beim Host gefiltert würden, was die Netzwerklast senken kann. IGMPv3 ist abwärts kompatibel.

2.4 Multicast-Proxy

Ein Multicast-Proxy ist ein Multicast-Forwarder. Er verwaltet Gruppenmitgliedschaften seiner Subnetzwerke, wie in Abschnitt 2.3 beschrieben, und leitet Gruppendaten nach bestimmten Regeln weiter, ohne ein Multicast-Routing Protokoll zu nutzen. Dies ist möglich, wenn die Netzwerktopologie in einer einfachen Baumstruktur angeordnet ist. So reicht es den Multicast-Status der angeschlossen Netzwerke zu kennen und auf Basis dieser Informationen die Gruppendaten weiter zu leiten. Da kein Multicast-Routing Protokoll eingesetzt wird, sinkt auch der Overhead für das Verteilen der Gruppendaten. So muss zum Beispiel

nicht nach dem kürzesten Pfad für die Gruppendaten gesucht oder Schleifen (Loops) entdeckt und vermieden werden.

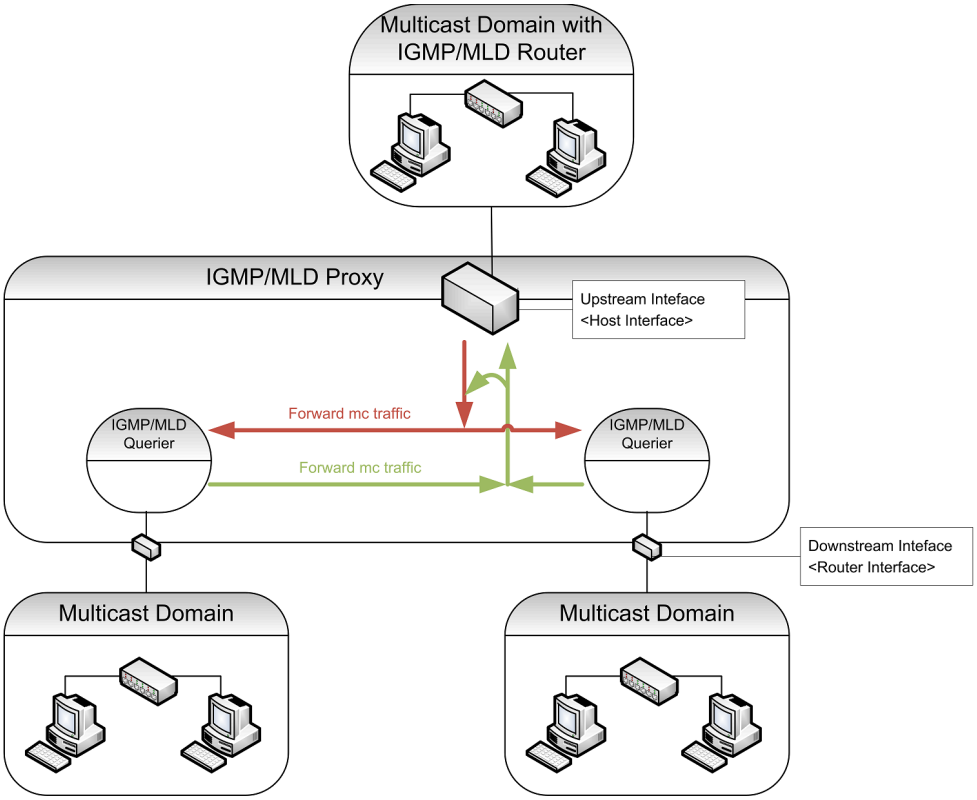


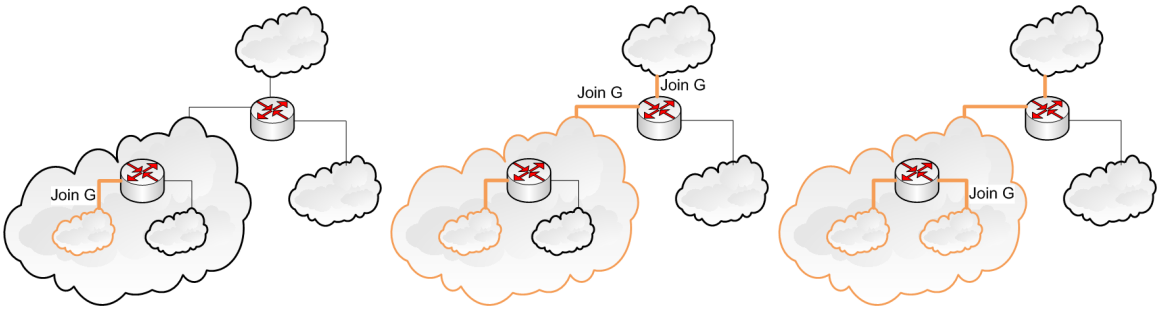
Abbildung 5: Multicast-Proxy

Proxy Aufbau

Der Multicast-Proxy ist im RFC 4605 [FHHS06] standardisiert. Er besitzt genau ein Upstream Interface und kann einen oder mehrere Downstream Interfaces haben (siehe Abbildung 5). Jeder Downstream (Router Interface) implementiert den Routerteil einer Gruppenverwaltung, den sogenannten Querier. Wird der Querier durch eine Querier Election zu einem NonQuerier, wird auch das Weiterleiten von Multicast-Paketen eingestellt. Der Upstream (Host Interface) hat ausschließlich die Funktionalität des Host Teils des IGMP/MLD Protokolls.

Die Multicast-Status Informationen werden in einer Proxy-Database zusammengefasst. Auf dieser Grundlage werden die Regeln zur Weiterleiten der Gruppendaten bestimmt. Die Proxy-Database enthält somit eine Menge von Abonnements, welche aus folgenden Tupeln besteht:

(Multicast Group Address, filtermode, source-list)



(a) (b) (c)

Abbildung 6: Aggregation der Multicastzustände

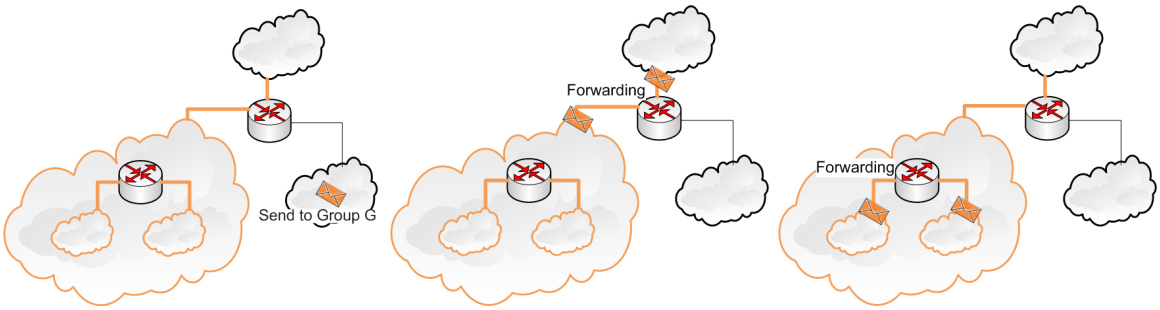
Ändert sich die Database, wird auch der Upstream-Zustand aktualisiert. Wenn zum Beispiel an einem Downstream von einem Proxy eine Gruppe G abonniert wird, abonniert auch das Upstream Interface diese Gruppe. Wenn zusätzlich an einem anderen Downstream dieselbe Gruppe abonniert wird, ändert dies nichts und nur wenn beide die Gruppe verlassen, verlässt auch der Upstream die Gruppe. Durch dieses Verfahren wirkt der Proxy transparent und es ist so gleichzeitig möglich, mehrere Subnetzwerke zu kapseln.

Abbildung 6a zeigt zwei Multicast-Proxies, die in einer Baumstruktur angeordnet sind, mit einer Verbindung an der Wurzel zu einer größeren Multicast Infrastruktur. Ein Multicast Subnetzwerk abonniert eine Gruppe. Der verantwortliche Proxy aggregiert daraufhin seinen neuen Multicast-Zustand zum Upstream. Wodurch sich auch der Multicast-Zustand vom anderen Proxy ändert (6b). Wird daraufhin aus einem anderen Subnetzwerk herraus die selbe Gruppe abonniert, bleibt der Upstream-Zustand erhalten (6c).

Weiterleiten von Gruppendaten

Gruppendaten, die am Upstream ankommen, werden in Abhängigkeit der Gruppenmitgliedschaften an die Downstreams weitergeleitet. Gruppendaten, egal welcher Gruppe, die am Downstream ankommen, werden zum Upstream und in Abhängigkeit der Gruppenmitgliedsschaften an andere Downstreams weitergeleitet. Außerdem werden Pakete nicht an Downstreams weitergeleitet, die im NonQuerier Zustand sind. In Abbildung 7a sendet ein Host aus einem beliebigen Subnetzwerk ein Gruppendaten-Paket. Dieses Paket wird zum Upstream weitergeleitet und an Downstreams, die dieselbe Gruppe abonniert haben, an die auch das Paket adressiert ist (7b, 7c).

Eine Vermischung von Gruppenverwaltungsversion der selben IP-Version ist möglich, da das Database Tupel auf alle Gruppenverwaltungs-Versionen angewendet werden kann.



(a) (b) (c)

Abbildung 7: Verteilung der Gruppendaten

2.5 Mobile Multicast

Mobile IP

Mobile IP (IPv4: RFC 3344 [Per02] und IPv6: RFC 6275 [PJA11]) ermöglicht mobilen Endgeräten, das Wechseln in ein anderes Subnetzwerk transparent für Transportprotokolle zu halten. So brechen zum Beispiel TCP-Verbindungen bei einem Netzwechsel nicht ab. Dafür hat das mobile Endgerät eine feste und eine dynamische IP-Adresse, die *Home-* und die *Care of Address*. Der Kommunikationspartner erreicht das mobile Endgerät immer über die Home-Adresse, welche stellvertretend von einem Home Agent gehalten wird. Der Home Agent leitet die Daten für das mobilen Endgerätetes zur dynamischen IP-Adresse weiter. So kann sich die dynamische IP-Adresse ändern, aber das mobile Endgerät ist trotzdem erreichbar. Dieses Verfahren hat den Nachteil, dass der TCP/IP Stack des Hosts angepasst werden muss. Um diese Anpassung zu vermeiden, wurde eine alternative Lösung entwickelt: PMIPv6.

PMIPv6

Proxy Mobile IPv6 (PMIPv6) ermöglicht wie Mobile IP eine Transparenz für Transportprotokolle beim Wechseln des Subnetzwerkes ohne Anpassung der mobilen Endgeräte (siehe Abbildung 8). Standardisiert ist dieses Protokoll im RFC 5213 [GLD+08].

Ein Mobile Node (MN), ein mobiles Endgerät, hat immer einen Local Mobility Anchor (LMA). Dieser LMA hat eine feste IP-Adresse und nimmt alle eingehenden Verbindungen für den MN stellvertretend entgegen. Betritt nun der MN eine PMIP-Domän, authentifiziert er sich beim Mobile Acces Gateway (MAG). Dieser MAG informiert den zuständigen LMA über die neue Position seines MNs und anschließend wird ein Tunnel zwischen LMA und dem MAG etabliert, über den der Informationsaustausch zwischen MN und LMA stattfindet. Wechselt der MN die PMIPv6 Domain, wiederholt sich der Vorgang. Da sich die IP-Adresse vom LMA hierbei nicht verändert, bleiben alle Verbindungen zum stellvertretenden MN erhalten.

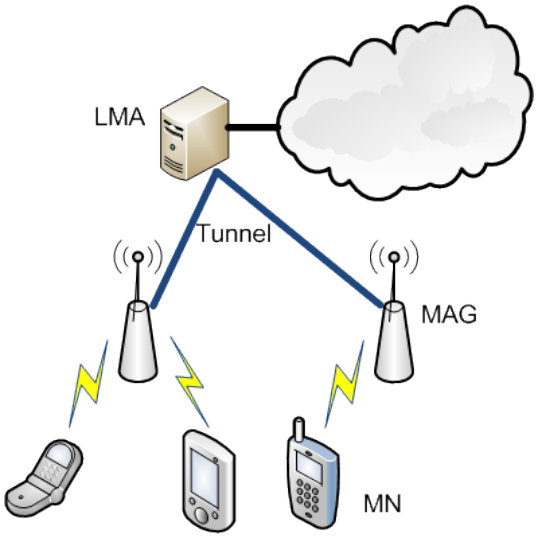


Abbildung 8: PMIPv6 Prinzip

An einem LMA können mehrere MNs angeschlossen werden und ein MAG kann auch mit mehreren LMAs verbunden sein.

Multicast in PMIPv6

Multicast Listener Support in PMIPv6 ist im RFC 6224 beschrieben. Das RFC beschreibt die Einbettung von Multicast Funktionalität in PMIPv6 Domains. Wie schon bei PMIPv6 selbst sind auch bei dieser Erweiterung weder Änderungen an den Multicast Protokollen noch an den MNs notwendig (siehe Abbildung 9). Die LMAs haben Anschluss an eine Multicast Infrastruktur. Um MNs nun mit dieser Multicast Infrastruktur verbinden zu können, benötigt jeder MAG einen MLD Proxy, der die Multicastzustände aggregiert und den Datenverkehr weiterleitet. Außerdem benötigen LMAs zur Tunnelseite einen Multicast Querier. Da ein Multicast-Proxy nur einen Upstream besitzen kann, ein MAG allerdings Verbindungen zu mehreren LMAs haben darf, wird jeder LMA mit den dazugehörigen MNs als eigene Proxy Domain gesehen, die jeweils auf eine isolierte Proxy-Instanz augelagert wird.

An diese MLD Proxy-Instanzen werden zusätzliche Anforderungen gestellt, da mit einer Fluktuation von MNs gerechnet werden muss. So wird ein General Query zu einem MN geschickt, nachdem er eine Verbindung zu einem neuen MAG aufgebaut hat, um zeitnah die Multicastzustände vom MN zu erfragen. Zum Beispiel kann so nach einem Handover die Unterbrechung eines Multicast Livestream minimal gehalten werden. Und wenn ein MN eine PIMPv6 Domain verlässt, aktualisiert die verantwortliche Proxy-Instanz ihre Multicastzustände zum Upstream, um ein unnötiges Versenden von Gruppendaten zu vermeiden.

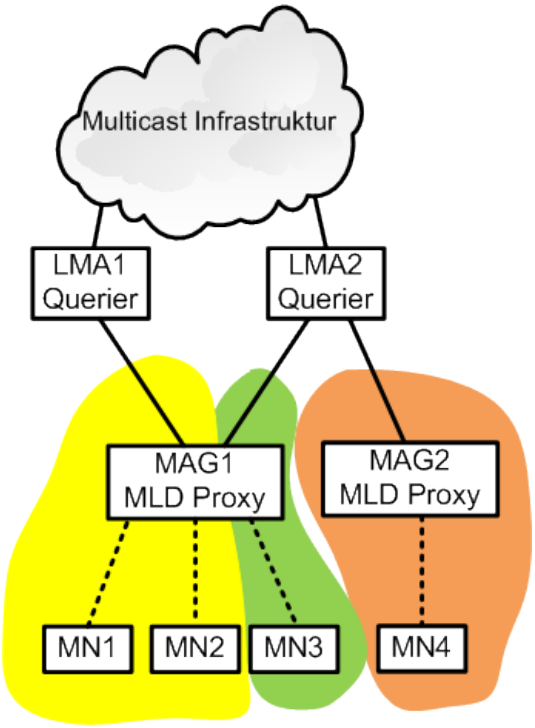


Abbildung 9: Multicast in PMIPv6

Fallbeispiel

In dem folgenden Fallbeispiel gehören MN1 und MN2 zum selben LMA und sind im ersten Abschnitt mit MAG1 verbunden. Im zweiten Abschnitt wechselt MN2 dann zu MAG2.

In Abbildung 10 wird der Ablauf gezeigt. Im 1. Schritt abonnieren beide MNs die Gruppe G. Der MAG1 aggregiert daraufhin seine Multicastzustände zum LMA (2. Schritt). Anschließend leitet der LMA den Gruppendaten-Strom für die Gruppe G über den Tunnel zum MAG1, welcher die Daten weiter an seine MNs leitet, die der Gruppe G beigetreten sind (3. Schritt).

Jetzt wechselt der MN2 die PMIPv6 Domain, verbindet sich mit MAG2 und es wird ein Tunnel zwischen LMA und MAG2 aufgebaut. Daraufhin sendet der MAG2 einen General Query an seinen neuen Downstream (4. Schritt) und MN2 antwortet mit einen Membership Report (Join-Nachricht). Der MAG2 aggregiert seine neuen Multicastzustände (5. Schritt) und der LMA sendet dann zu beiden MAGs den Gruppendaten-Strom. Die wiederum an die MNs weitergeleitet werden (6. Schritt).

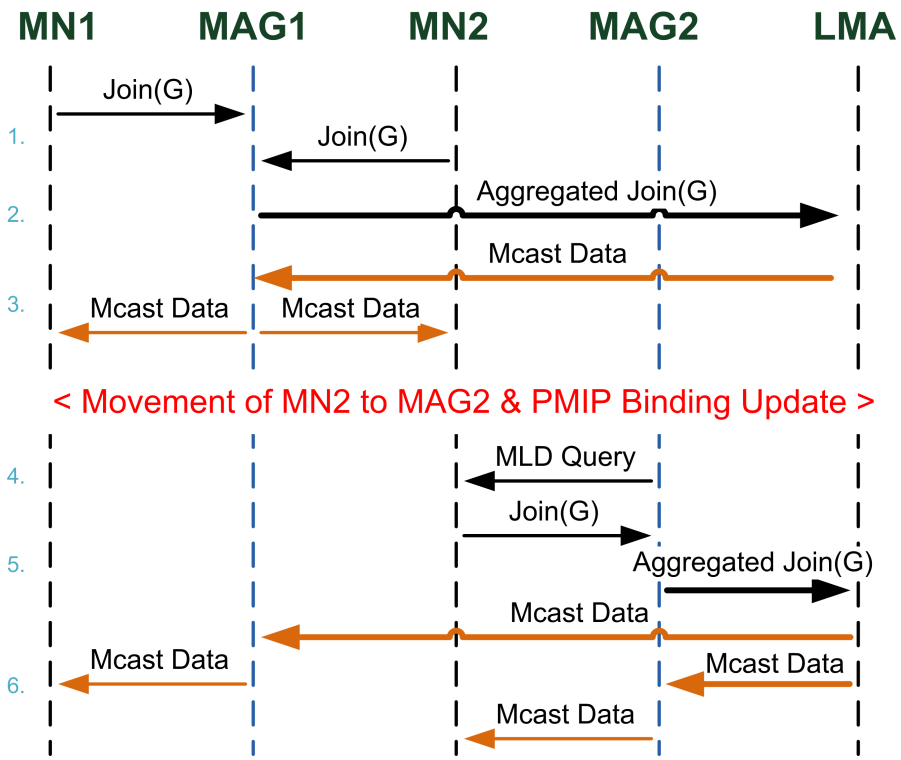


Abbildung 10: Ablauf eines Handovers

3 Multicast in Linux

Die Multicastfunktionalitäten vom Linux Kernel werden in diesem Kapitel erläutert. Auf Basis dieser Erkentnisse wird ein Entwurf für einen Multicast-Proxy erstellt.

Im ersten Schritt werden die Informationen, die der Kernel im Bereich Multicast zu Verfügung stellt, dargelegt, denn diese Informationen werden im Laufe der Entwicklungsphase zum Beispiel zu Debugzwecken, Fehleranalyse oder auch zur Evaluierung der Funktionstüchtigkeit verwendet. Dazu gehört das Auslesen von Multicast-Zuständen und Flags, mit denen das Verhalten des Kernels beeinflusst wird. Anschließend wird beschrieben wie Multicast-Gruppen mit Kernelfunktionen abonniert und verlassen werden. Nach den elementaren Funktionen wird dann das Routing von Gruppendaten analysiert. Daraufhin wird das Empfangen von Multicast-Nachrichten (Join-/Leave-Nachrichten) untersucht, welche als Informationen notwendig sind, um später die Multicast-Routen zu berechnen und eintragen zu lassen.

3.1 Programmcode und Header

Im Verlauf dieses Kapitels werden Programmcode-Ausschnitte zur Verdeutlichung vorgestellt. In diesen Ausschnitten werden die Fehlerbehandlungen zugunsten der Übersichtlichkeit in den meisten Beispielen weggelassen. Um in den nachfolgenden Unterkapiteln die Methoden und Definitionen nutzen zu können, müssen einige Header-Dateien inkludiert werden:

- <*sys/socket.h*>: enthält Methoden wie *socket(), set/getsockopt(), send(), recv()* usw.
- <*netinet/in.h*>: enthält Definitionen wie *SOCK_RAW, IPPROTO_IGMP, IPPROTO_IPv6*, Structs für IP-Adressen und Multicast-Routing Requests.
- <*net/if.h*>: enthält Methoden und Definitionen um die Eigenschaften von Netzwerk-Interfaces auszulesen.
- <*arpa/inet.h*>: enthält Funktionen um IP-Adressen zwischen verschiedenen Formaten zu konvertieren.
- <*linux/mroute.h*> und <*linux/mroute6.h*>: enthalten alle Definitionen und Strukturen, die für die Manipulation von Multicast-Routing-Tabellen benötigt werden

3.2 Multicast-Status

IPv4

Der Multicast-Status eines Rechners lässt sich für IPv4 unter /proc/net/igmp auslesen. In dieser Tabelle wird gezeigt, welche Multicast-Gruppe über welches Netzwerkinterface abonniert wurde (siehe Tabelle 5). Sie ist wie folgt definiert:

```
Idx       Device    : Count Querier     Group    Users Timer          Reporter
1         lo        :     1      V3
                                       010000E0     1 0:00000000            0
2         eth0      :     2      V2
                                       FB0000E0     1 0:00000000            1
                                       010000E0     1 0:00000000            0
```

Tabelle 5: Beispiel eines Multicast IPv4 Statuses

Idx: Interface Index

Device: Name des Interfaces

Count: Anzahl der abonnierten Gruppen an diesem Interface

Querier: Multicast-Version, die an diesem Interface genutzt wird. Lässt sich per Hand konfigurieren (siehe Kapitel 3.3) oder wird durch die Version der empfangenen General Queries bestimmt.

Group: Abonnierte Multicast-Gruppe in hexardezimal und Networkbyteorder.

Users: Anzahl der Benutzer an diesem PC, die diese Gruppe abonniert haben.

Timer und Reporter: Timer zeigt die aktuelle Verzögerungszeit an bis eine Join-Nachricht gesendet wird im Format [läuft der Timer : Wartezeit]. Wenn ein Report gesendet wurde, steht in der Spalte Reporter eine eins. Hinweis: Diese Felder werden bei IGMPv3 nicht genutzt, da es bei IGMPv3 keine Reportunterdrückung gibt.

So ist zum Beispiel in der Tabelle 5 an dem Netzwerkinterface eth0 mit den Interface Index zwei die Gruppe 224.0.0.1 (010000E0) und die Gruppe 224.0.0.251 (FB000E0) abonniert.

IPv6

Für IPv6 sind die Multicast-Zustände unter /proc/net/igmp6 zu finden (siehe Tabelle 6). Die Kerninformationen sind identisch zur IPv4 Tabelle. So ist zum Beispiel für das Netzwerkinterface eth0 die Gruppe ff02::fb und die Gruppe ff02:1 abonniert.

```
Idx  Device          Group                         Users  Flags Timer running
1    lo              ff020000000000000000000000000001     1 0000000C 0
2    eth0            ff0200000000000000000000000000fb     1 00000004 0
2    eth0            ff020000000000000000000000000001     1 0000000C 0
3    wlan0           ff020000000000000000000000000001     1 0000000C 0
```

Tabelle 6: Beispiel eines Multicast IPv6 Statuses

Die snmp6 Tabelle sollte noch erwähnt werden, welche auch unter /proc/net zu finden ist (siehe Tabelle 7). Anhand ihrer Informationen lässt sich erkennen, ob General Queries

```
Icmp6InGroupMembQueries          4
Icmp6InGroupMembResponses        0
Icmp6InMLDv2Reports              0

Icmp6OutGroupMembQueries         0
Icmp6OutGroupMembResponses       4
Icmp6OutMLDv2Reports             0
```

Tabelle 7: Ausschnitt aus der SNMP6 Tabelle

empfangen wurden. So wurden an diesem Computer vier Gerneral Queries empfangen und vier Join-Nachrichten verschickt. Für IPv4 gibt es derartige Informationen nicht, da keine MIB (Management Information Base) Gruppe für IGMP definiert wurde.

3.3 Flags

Die wichtigsten Flags, die das Multicast-Verhalten im Linux Kernel und an den Netzwerkadapter beeinflussen, sind in der folgenden Liste zusammengefasst.

multicast-flag: Unterstützt ein Netzwerkadapter Multicast, so ist dieses Flag gesetzt. Überprüft werden kann dieser Zustand mit ifconfig in der Linux Konsole [CKHR05].

allmulti-flag: Ist dieses Flag gesetzt, so filtert der Netzwerkadapter keine Multicast-Pakete. Es wird gesetzt, wenn das Multicast-Routing aktiviert wird. Überprüft werden kann auch dieser Zustand mit ifconfig in der Linux Konsole [CKHR05].

rp_filter: Der *Reverse Path Filter* entspricht einem Ingress Filter. Dieser Filter verwirft Pakete mit unbekannter Absender-Adresse. Dies hat mehrere negative Effekte. Wenn Multicast-Proxys kaskadierend aufgebaut sind, leiten sie keine Pakete von anderen Proxys weiter und Hosts verwerfen weitergeleitete Pakete. Deaktivieren kann dieser Filter unter /proc/sys/net/ipv<x>/conf/all/rp_filter. Außerdem muss zusätzlich für das jeweilige Interface der Filter deaktiviert werden. Dafür wird all durch zum Beispiel eth0 ersetzt.

force_igmp_version: Definiert die zu nutzenden IGMP Verison. Zur Auswahl stehen IGMPv1, IGMPv2 und IGMPv3. Per Default ist hier eine Null gesetzt. Das bedeutet, dass die IGMP Version in Abhängigkeit der empfangenen General Query Versionen gewählt wird. Wurde noch kein Query empfangen, ist die Version auf IGMPv3 gesetzt. Manuell ändern lässt sich die Version unter /proc/sys/net/ipv<x>/conf/<Interface>/force_[igmp | mld]_version.

mrt-flag: Ist für ein Socket das MRT-Flag gesetzt, so darf man hiermit die Multicast-Routing-Tabellen manipulieren. Ob das Flag gesetzt ist, kann unter /proc/sys/net/ipv<x>/conf/all/mc_forwarding eingesehen werden. Für mehr Details siehe Abschnitt 3.5.

mc_forwarding: Ist das Multicast Forwarding Flag bei einem Interface gesetzt, so darf dieses Interface Gruppendaten weiterleiten. Ob das Flag gesetzt ist, kann unter /proc/sys/net/ipv<x>/conf/<Interface>/mc_forwarding eingesehen werden. Für mehr Details siehe Kapitel 3.5.

3.4 Join- und Leaves-Nachrichten versenden

Das Senden von Join- und Leave-Nachrichten lässt sich über die Linux Kernelfunktionen setsockopt() realisieren. Als Parameter benötigt diese Funktion ein Betriebssystemsocket, das zu benutzende Protokoll, den Optionsnamen, die Optionswerte und die Länge der Optionswerte. Diese Methode wird für den einzelnen Anwendungsfall wie folgt parametrisiert.

Socket

Zum Versenden von Join- und Leave-Nachrichten reichen Datagrammsockets aus, welche ohne Root-Rechte benutzt werden können (siehe Listing 1).

Listing 1: Initialisierung von Datagramm Socket

```
0 //IPv4
  sk = socket(AF_INET, SOCK_DGRAM, IPPROTO_IP);
  //IPv6
  sk = socket(AF_INET6, SOCK_DGRAM, IPPROTO_IP);
```

Join-Nachricht

Zum Versenden einer Join-Nachricht wird für IPv4 die Struktur ip_mreq und für IPv6 ipv6_mreq benötigt. Die Strukturen enthalten zwei Membervariabeln. Die Erste zum Aufnehmen der Multicast-Adresse, die abonniert werden soll, und die Zweite zum Auswählen des Netzwerk-Interfaces. Für das Interface wird unter IPv4 die primäre IP-Adresse eingefordert, unter IPv6 der Interface Index (siehe Listing 2).

Leave-Nachricht

Das Verlassen einer Multicast-Gruppe unterscheidet sich vom Abonnieren nur durch den Optionsnamen. Siehe Listing 3.

3.5 Multicast-Routing

Das Multicast-Routing, also das Weiterleiten von Multicast-Paketen kann der Linux Kernel übernehmen. Die Abbildung 11 zeigt das Prinzip der Linuximplementierung.

Listing 2: Join-Nachricht

```
0 //IPv4
  struct ip_mreq imr4;
  imr4.imr_multiaddr = //group address
  imr4.imr_interface = //interface address
  rc = setsockopt(sk, IPPROTO_IP, IP_ADD_MEMBERSHIP, imt4, sizeof(
      imr4));
5 //IPv6
  struct ipv6_mreq imr6;
  imr6.ipv6mr_multiaddr = //group address
  imr6.ipv6mr_interface = //interface index
  rc = setsockopt(sk, IPPROTO_IP6, IPV6_JOIN_GROUP, imr6, sizeof(
      imr6));
```

Listing 3: Leave-Nachricht

```
0 //IPv4
  struct ip_mreq imr4; //identisch zum IPv4 join
  rc = setsockopt(sk, IPPROTO_IP, IP_DROP_MEMBERSHIP, imr4, sizeof(
      imr4));
  //IPv6
  struct ipv6_mreq imr6; //identisch zum IPv6 join
5 rc = setsockopt(sk, IPPROTO_IP6, IPV6_LEAVE_GROUP, imr6, sizeof(
      imr6));
```

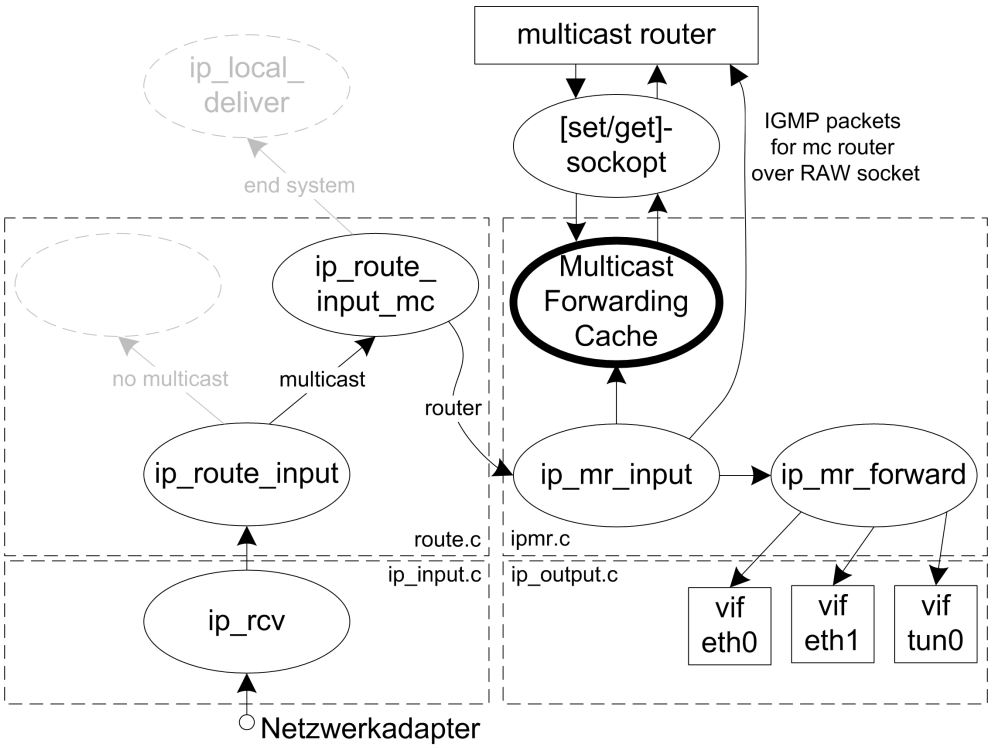


Abbildung 11: Multicast-Datenpfad im Linux Kernel

Viele Netzwerkadapter können empfangene Ethernetpakete vorfiltern. Dies wird gemacht, um die CPU von Paketen zu entlasten, die nicht für diesen Rechner bestimmt sind. Zum Filtern von Ethernetpakteten wird die MAC-Adresse des Netzwerkadapters mit dem Ziel des Paketes verglichen, stimmen die Adressen nicht überein oder ist es eine Broadcast Adresse, so kann es verworfen werden. Um Multicast-Pakete zu filtern, benötigt der Netzwerkadapter eine Liste von relevanten Multicast-Adressen, welche vom Betriebssystem bereitgestellt wird. Beim Start eines Routers wird der Multicast-Filter deaktiviert, da ein Router alle Multicast-Nachrichten empfangen und dann mögliche Regeln zum weiterleiten von Multicast-Daten anwenden muss.

Empfängt ein Netzwerkadapter ein Paket, so wird es über die Methode *ip_rcv* zur Methode *ip_route_input* gereicht. Diese leitet es, falls es ein Multicast-Paket ist, weiter zur Methode *ip_route_input_mc*. Handelt es sich bei diesem Rechner um ein Router, so wird das Paket an das Multicast-Routing Modul weitergereicht (Kerneldatei ipmr.c).

Ip_mr_input ist der Einstiegspunkt des Multicast-Routing Moduls. Hier werden alle Multicast-Pakete, die gegebenenfalls weitergeleitet werden sollen, bearbeitet. Über die Methode *ipmr_cache_find(source addr, group addr)* wird im ersten Schritt iterativ nach einem Eintrag in dem Multicast-Forwarding-Cache gesucht. Der Forwarding-Cache ist eine Tabelle, in der alle Multicast-Routen eingetragen sind. Whitecards für zum Beispiel die Quelladresse sind implementierungsbedingt nicht erlaubt. Wurde ein Eintrag gefunden, so kann

über die Methode *ip_mr_forward* das Multicast-Paket für die angeforderten Interfaces repliziert und verschickt werden. Konnte kein Eintrag für das Quell-, Gruppenadresse Tupel gefunden werden, so informiert der Kernel den Multicast-Router über einen Seitenkanal, dass es einen sogenannten Cache Miss gab und speichert gleichzeitig das Paket für einen kurzen Zeitraum zwischen. Nun kann der Router diesen Zeitraum nutzen und eine Route nachtragen.

Die Codebeispiele, Bilder und Inhalte wurden überwiegend aus dem "openbsd programmer's manual" [Wai07] und aus dem Buch "Design and Implementation of Network Protocols in the Linux Kernel" [WPR+04] erarbeitet.

In den folgenden Abschnitten wird beschrieben, wie die Kernelschnittstelle zum Multicast-Routing gehandhabt wird.

3.5.1 Sockets

Um alle weiteren Kernelfunktionen zu nutzen, wird ein Socket vom Betriebssystem angefordert. Die Aufrufe zwischen IPv4 und IPV6 unterscheiden sich, da unter IPv4 – IGMP noch ein eigenständiges Protokoll war und bei IPv6 nun als MLD-Protokoll in das ICMP-Protokoll eingebettet ist (siehe Listing 4).

Listing 4: Socket Initialisierung

```
0 //IPv4
  sk = socket(AF_INET, SOCK_RAW, IPPROTO_IGMP);
  //IPv6
  sk = socket(AF_INET6, SOCK_RAW, IPPROTO_ICMPV6);
```

Da hier Raw-Sockets benötigt werden, muss der Programmcode mit administrativen Rechten ausgeführt werden.

3.5.2 Multicast-Routing-Flag

Das Betriebssystem verhindert mithilfe des sogenannten MRT-Flags (Multicast-Routing-Flag), dass mehrere Programme gleichzeitig auf den Routing-Tabellen schreiben und so Race Conditions verursachen könnten. Das Socket, welches das MRT-Flag besitzt, hat das alleinige Schreibrecht. Die IPv4 und IPv6 Aufrufe zum Setzen des Flags sind im Listing 5 beschrieben.

Um das MRT-Flag zurück zu setzen wird entweder das Socket geschlossen oder die Anweisung aus Listing 6 ausgeführt.

Unter dem Betriebssystempfad */proc/sys/net/ipv<x>/conf/all/mc_forwarding* kann für jede IP-Version geprüft werden, ob das Weiterleiten von Gruppendaten aktiviert ist.

Listing 5: Setzen des MRT-Flag

```
0 //IPv4
  int val=1;
  rc = setsockopt(sk, IPPROTO_IP, MRT_INIT, (void*)&val, sizeof(val)
     );
  //IPv6
  int val=1;
5 rc = setsockopt(sk, IPPROTO_IPV6, MRT6_INIT, (void*)&val, sizeof(
     val));
```

Listing 6: Freigeben des MRT-Flag

```
0 //IPv4
  rc = setsockopt(sk, IPPROTO_IP, MRT_DONE, NULL, 0);
  //IPv6
  rc = setsockopt(sk, IPPROTO_IPV6, MRT6_DONE, NULL, 0);
```

3.5.3 Virtuelles Interface

> "As routed multicast datagrams can be received/sent across either physical interfaces or tunnels, a common abstraction for both was devised" [dG98].

Das heißt für ein Interface, ob physikalisch oder Tunnel, das für Multicast-Routing genutzt werden soll, muss ein sogenannter vif (virtueller Interface Index) definiert, konfiguriert und an den Kernel weiter gereicht werden. Für jedes Netzwerkinterface, das Multicast-Pakete weiterleiten soll, muss stellvertretend ein vif definiert werden. Die maximale Anzahl der vifs ist durch die Definition, unter IPv4, MAXVIFS auf 32 und unter IPv6 MAXMIFS auch auf 32 gesetzt. Diese Definitionen können aufgrund von Implementierungsdetails bei einer 32-Bit-Systemarchitektur nicht erhöht werden.

IPv4

Um einen vif zu definieren wird ein freier Unique Identifier ($0 \leq \text{vifNum} < \text{MAXVIFS}$) gewählt, welcher ein abstraktes Interface repräsentiert. Für dieses virtuelle Interface werden mehrere Eigenschaften konfiguriert. Zum Setzten und Löschen von vifs siehe Listing 7.

- flags:
  - `VIFF_USE_IFINDEX`: falls der Interface Index anstatt der IP-Adresse angeben werden soll
  - `VIFF_TUNNEL`: wenn es sich bei diesem Interface um einen Tunnel handelt.

Listing 7: Hinzufügen von vifs unter IPv4

```
0  //add vif
   struct vifctl vc;
   memset(&vc, 0, sizeof(vc));
   vc.vifc_vifi = vifNum;
   vc.vifc_flags = flags; //VIFF_USE_IFINDEX
5  vc.vifc_rate_limit = MROUTE_RATE_LIMIT_ENDLESS; //0
   vc.vifc_threshold = MROUTE_TTL_THRESHOLD; //1
   vc.vifc_lcl_ifindex =if_nametoindex(if_name); //e.g. "eth0"
   if(ipTunnel){
       inet_pton(AF_INET6, ipTunnelRemoteAddr, (void*)&vc.
           vifc_rmt_addr);
10 }
   rc = setsockopt(sk,IPPROTO_IP,MRT_ADD_VIF,(void *)&vc,sizeof(vc));

   //del vif
   struct vifctl vifc;
15 memset(&vifc, 0, sizeof(vifc));
   vifc.vifc_vifi= vifNum;
   rc = setsockopt(sk, IPPROTO_IP, MRT_DEL_VIF, (char *)&vifc,sizeof(
       vifc));
```

- *threshold*: kleinster TTL-Wert von Multicast-Paketen, die weitergeleitet werden.
- *rate_limit*: maximale Weiterleitgeschwindigkeit von Gruppendaten in Bits/s (max speed = -1).
- *lcl_ifindex*: Interface-Index zum gewählten Interface.
- *rmt_addr*: die Remote-Adresse wird nur benötigt, falls das Tunnel Flag gesetzt wurde.

IPv6

Der Aufbau ist äquivalent zu IPv4 bis auf, dass keine zusätzlichen Flags mehr gesetzt werden (siehe Listing 8).

Listing 8: Hinzufügen von vifs unter IPv6

```
0  //add vif
   struct mif6ctl mc;
   memset(&mc, 0, sizeof(mc));
   mc.mif6c_mifi = vifNum;
   mc.mif6c_flags = flags; //0
5  mc.vifc_rate_limit = MROUTE_RATE_LIMIT_ENDLESS6; //0
   mc.vifc_threshold = MROUTE_TTL_THRESHOLD; //-1
   mc.mif6c_pifi = if_nametoindex(if_name); //e.g. "eth0"
   rc = setsockopt(sk, IPPROTO_IPV6, MRT6_ADD_MIF, (void *)&mc, sizeof
       (mc));

10 //del vif
   struct mif6ctl mc;
   memset(&mc, 0, sizeof(mc));
   mc.mif6c_mifi = vifNum;
   rc = setsockopt(sk, IPPROTO_IPV6, MRT6_DEL_MIF, (char *)&mc,
       sizeof(mc));
```

Unter dem Betriebssystempfad /proc/sys/net/ipv<x>/conf/<Interface Name>/mc_forwarding kann von jedem Interface und jeder IP-Version der Multicast-Forwarding-Status angezeigen werden. Alternativ kann unter /proc/net/[ip_mr_vif | ip6_mr_vif] die Multicast-Routing-Interface-Tabelle betrachtet werden[Pel02].

3.5.4 Multicast-Forwarding-Cache

Der Multicast-Forwarding-Cache enthält Regeln wie eingehende Multicast-Pakete behandelt werden. Unter dem Betriebssystempfad /proc/net/[ip_mr_cache | ip6_mr_cache] wer-

den alle Regeln zur Weiterleitung von Gruppendaten aufgelistet. Ein Beispiel soll demonstrieren, wie diese Regeln zu verstehen sind.

In Listing 9 sind mehrere Interfaces zum Weiterleiten von Gruppendaten deklariert. Tun0 hat hier zum Beispiel einen virtuellen Interface Index (vif) von 2 und ist an den Flags als Tunnel zu erkennen. Nun wird eine Regel zum Weiterleiten von Gruppendaten eingerichtet. Pakete der Gruppe 224.225.0.1 vom Sender (Origin) 192.168.1.118 die am Interface eth0 empfangen werden, werden zum Interface eth1 und tun0 weitergeleitet. In Listing 10 ist genau diese Regel eingetragen. Hier sind die IP-Adressen, wie auch schon bei den Multcastzuständen, in hexadezimal und Networkbyteorder angegeben. Das Input Interface (Iif) hat einen vif von 0 und steht somit für eth0. Die Output Interfaces (Oifs) stehen für eth1 und tun0. Der nachfolgende TTL Wert von 5 ist hier irrelevant. Dieses Beispiel wurde im "Linuxjournal" [Pel02] beschrieben.

Listing 9: Beispiel: proc/net/ip_mr_vif

```
0 // vifNum4    ifName                                                  lcl_addr
1 Interface          BytesIn  PktsIn  BytesOut  PktsOut  Flags  Local
2 0          eth0    129090   1655    1872      24       00000  C801A8C0
3 1          eth1    1872     24      129090    1655     00000  C802A8C0
4 2          tun0    0        0       0         0        00001  B10008B0
```

Listing 10: Beispiel: prox/net/ip_mr_cache

```
0 Group    Origin   Iif     Pkts    Bytes    Wrong  Oifs
1 0100E1E0 7601A8C0 0       755     58890    0      1:5  2:5
```

IPv4

Die Anweisungen um eine Regel zum Weiterleiten von Gruppendaten zu erstellen und zu entfernen sind in Listing 11 beschrieben. Zum Erstellen wird die Origin- und die Group-Adresse übergeben. Außerdem wird ein vif als Input (Input_vifNum) und eine Menge von vifs als Oif angegeben. Das unsigned char Array *mfcc_ttls* ist MAXVIFS lang und jede Position im Array deren Eintrag größer Null ist repräsentiert ein Oif (nach Listing 9). Das heißt z.B. wenn an Position 1 und 2 ein Eintrag größer Null steht, werden die Interfaces eth0 und tun0 als Oif festgesetzt. Der Eintrag selber gibt an, welche TTL das Multicast-Paket mindestens haben muss, damit es weitergeleitet wird. Zum Entfernen einer Regel zum Weiterleiten von Gruppendaten wird das Triple Origin, Group und Iif benötigt.

Listing 11: Erstellen und entfernen einer Regel zum Weiterleiten von Gruppendaten unter IPv4

```
0  //add
   struct mfcctl mc;
   memset(&mc, 0, sizeof(mc));
   mc.mfcc_parent = input_vifNum; //Iif
   mc.mfcc_origin = source_addr; //origin
5  mc.mfcc_mcastgrp = group_addr; //group
   for (unsigned int i = 0; i < MAXVIFS; i++){
      mc.mfcc_ttls[output_vifTTL[i]] = MROUTE_DEFAULT_TTL //Oif
   }
   rc = setsockopt(sk, IPPROTO_IP, MRT_ADD_MFC,(void *)&mc, sizeof(mc
      ));
10
   //del
   struct mfcctl mc;
   memset(&mc, 0, sizeof(mc));
   mc.mfcc_parent = input_vifNum;
15 mc.mfcc_origin = source_addr;
   mc.mfcc_mcastgrp = group_addr;
   rc = setsockopt(sk, IPPROTO_IP, MRT_DEL_MFC,(void *)&mc, sizeof(mc
      ));
```

IPv6

Die Anweisungen um eine Multicast-Forwading-Regel unter IPv6 zu erstellen, sind bis auf den Oif Anteil identisch. Für die Oifs wird nun das Struct if_set verwendet, wobei die Position jedes Bits, welches true ist, als Oif genutzt wird. Die Bits lassen sich mit dem Macro IF_SET() setzen (siehe Listing 12).

Listing 12: Erstellen und entfernen einer Regel unter IPv6

```
0  //add
   struct mf6cctl mc;
   memset(&mc, 0, sizeof(mc));
   mc.mfcc_parent = input_vifNum; //Iif
   mc.mf6cc_origin = source_addr; //origin
5  mc.mf6cc_mcastgrp = group_addr; //group
   for (unsigned int i = 0; i < MAXMIFS; i++){
       if(output_vifTTL[i] > 0){
           IF_SET(i,&mc.mf6cc_ifset); //Oif
       }
10 }
   rc = setsockopt(sk, IPPROTO_IPV6, MRT6_ADD_MFC,(void *)&mc, sizeof
       (mc));

   //del
   struct mf6cctl mc;
15 memset(&mc, 0, sizeof(mc));
   mc.mf6cc_parent = input_vifNum;
   mc.mf6cc_origin = source_addr;
   mc.mf6cc_mcastgrp = group_addr;
   rc = setsockopt(sk, IPPROTO_IPV6, MRT6_DEL_MFC,(void *)&mc, sizeof
       (mc));
```

3.6 Kernel- und Verwaltungs-Nachrichten empfangen

Um Gruppenverwaltungs-Nachrichten zu empfangen, bedarf es spezifischer Bedingungen und Voreinstellungen im Kernel (siehe Tabelle 8).

Zum Empfangen aller Nachrichten werden Raw Sockets benötigt (siehe hierzu Listing 4). Anschließend können General Queries empfangen werden, da sie an die All-Nodes Adresse adressiert werden (siehe Tabelle 4) und jeder Rechner dieser Gruppe angehört, wird diese Nachricht vom Kernel- in den Userspace gereicht. Um eine Leave-Nachricht zu empfangen wird zusätzlich für All-Routers Gruppe (IPv4: 224.0.0.2, IPv6: FF01::2) abon-

Nachrichten	Raw Socket	All-Node Addr	All-Routers Addr	Mrt-Flag
General Query	x	x		
Leave	x		x	
Join	x			x
G. Specific Query	x			x
Cache Miss	x			x

Tabelle 8: Vorraussetzungen zum Empfangen von Gruppenverwaltungs-Nachrichten

niert. Die Join-Nachricht und der Group-Specific Query werden an individuelle Gruppen gesendet, so ist es notwendig, dass Mrt-Flag zu setzen um diese Nachrichten zu empfangen.

Cache Miss

Findet der Kernel im Router Modus zu einem empfangenen Gruppendaten-Paket keine passende Regel zum Weiterleiten von Gruppendaten so, schickt er eine Cache Miss Nachricht zu dem Raw Socket, welches das Multicast-Routing-Flag gesetzt hat.

Unter IPv4 versendet der Kernel das *struct igmpsmp* als Cache Miss Nachricht. Es enthält einen Member *msgtype* der auf IGMPMSG_NOCACHE gesetzt sein muss, das vif (virtueller Interface Index)von dem das Paket empfangen wurde, sowie die Quell- und Zieladresse. Um zu überprüfen, ob es sich bei einer empfangenen Nachricht um ein Cache Miss handelt, wird der Protokolltyp des empfangenen IP-Paketes auf den Wert Null getestet.

Unter IPv6 heißt das *struct mrt6msg*. Es handelt sich um diese Struktur wenn der ICMPv6 Type den Wert Null hat.

IP Interface Mapping

Um Gruppenverwaltungs-Nachrichten zu empfangen, wird ein Raw Socket mit einem gesetzten Mrt-Flag benötigt. Außerdem muss die All-Routers Group Adresse an den relevanten Netzwerk-Interfaces abonniert werden. Sollen nun von mehreren Interfaces die Gruppenverwaltungs-Nachrichten empfangen werden, muss es von diesem Socket geschehen, da nur ein Mrt-Flag gleichzeitig gesetzt sein darf. Es besteht also nicht die Möglichkeit das Socket an ein Interface zu binden. Um nun das Interface herauszufinden, von welchem die Nachricht (zum Beispiel Join-Nachricht) empfangen wurde, muss ein Mapping zwischen der Quelladresse des empfangen IP-Paketes und den IP-Adressen der Interfaces durchgeführt werden.

Unter IPv6 ist dies nicht notwendig, denn durch die Kernel Methode *recvmsg()* wird neben dem IP-Paketet zusätzlich eine Struktur mit Zusatzinformation, wie dem genutzten Netzwerk Inteface, übergeben.

4 Entwurf des Multicast-Proxys

Dieses Kapitel beschreibt den Entwurf des Multicast-Proxys. Im ersten Schritt werden die Anforderungen zusammengefasst und Rahmenbedingungen erläutert. Anschließend werden existierende Implementierungen aus diesem Bereich vorgestellt und die Erstellung eines neuen Konzeptes motiviert. Daraufhin wird das eigene Multicast-Proxy Konzept vorgestellt.

4.1 Anforderungen

Der Multicast-Proxy wird sowohl IGMP (IPv4) als auch MLD (IPv6) unterstützen und somit versionsneutral entworfen werden. Zusätzlich werden die Erweiterungen durch PMIPv6 implementiert, das bedeutet der Proxy wird mehrfach instanziierbar und die zugehörigen Downstreams dynamisch konfigurierbar sein. Außerdem gibt es das Ziel, dass das Programm unter den gängigen Linux Distributionen lauffähig sein wird und die vorhanden Multicast-Kernel-Funktionen, im Wesentlichen das Weiterleiten von Multicast-Daten, für ein performantes Design und für die Vermeidung neuen Codes verwendet werden. Um spezifische Kernelfunktionen nutzen zu können, die C-Intefaces zur Verfügung stellen, und um gleichzeitig ein objektorientiertes Design zu ermöglichen, wird zur Implementierung die Programmiersprache C++ eingesetzt.

4.2 Existierende Implementierungen

Es gibt drei freie Implementierungen, deren Ansätze, Vor- und Nachteile in der folgenden Auflistung beschrieben und in Tabelle 9 zusammengefasst sind.

ecmh unterstützt die Protokolle MLDv1 und MLDv2 [Mas05]. Ecmh verarbeitet alle Multicast-Funktionalitäten im Userspace. Das heißt, es werden weder Multicast-Routing-Tabellen noch die Report/Leave Funktionen, die für den Upstream benötigt werden, vom Kernel genutzt. Durch die Unabhängigkeit von den Kerneltabellen, lassen sich mehrere Instanzen starten. Da das Weiterleiten von Paket im Userspace gehandhabt wird, ist ecmh im allgemeinen weniger performant. Das Programm wurde in C geschrieben und ist singlethreaded.

igmpproxy unterstützt die Protokolle IGMPv1 und IGMPv2 [CB09]. Der Proxy benutzt wie im Abschnitt 3.5 beschrieben die Multicast-Routing-Tabellen, weshalb er nur einmal startbar ist. Das Programm wurde in C geschrieben und ist singlethreaded.

gproxy unterstützt die Protokolle IGMPv1, IGMPv2 und IGMPv3 [Lah02]. Das Programm wurde in C geschrieben und ist singlethreaded. Der Proxy benutzt wie im Abschnitt 3.5 beschrieben die Multicast-Routing-Tabellen, weshalb er nur einmal startbar ist. Die offizielle Webseite von gproxy ist am Anfang des Jahres 2011 offline gegangen und so ist der Quellcode nicht mehr verfügbar.

	ecmh	**igmpproxy**	**gproxy**
Protokolle	MLDv1/2, m6trace,	IGMPv1/2,	IGMPv1/2/3,
Mehrfach instanziierbar	ja	nein	nein
Sprache	C	C	C
Last Release Date	09.02.2005	05.10.2009	10.06.2002

Tabelle 9: Multicast-Proxy Implementierungen

Die bisherigen Implementierungen sind zum Teil nicht mehr aktuell und werden nicht mehr weiterentwickelt. Kein Proxy unterstützt IGMP (IPv4) und MLD (IPv6) und die einzige MLD Implementierung nutzt die Multicast-Funktionalität des Linux Kernels nicht. Von den IGMP Implementierungen lassen sich nur einzelne Instanzen starten. Somit erfüllt keines der Programme die Anforderungen und so wurde entschieden einen neuen Multicast-Proxy zu entwickeln.

4.3 Problemstellung und Lösungskonzepte

Der Multicast-Proxy wird IGMP (IPv4) und MLD (IPv6) unterstützen. Deshalb ist es notwendig, das Programm zum größten Teil IP-versionsneutral zu halten, um doppelten Programmcode zu vermeiden. Es werden die versionsabhängigen Programmteile in eigenständige Klassen ausgelagert. Dazu gehören überwiegend die Kernelfunktionen, wie zum Beispiel Abonnieren/Verlassen von Multicast-Gruppen und das Multicast-Routing. Zum Speichern von IP-Adressen wird das *struct sockaddr_storage* in einer Klasse gekapselt, um es transparent mit Adressfunktionen erweitern zu können. Dazu gehören zum Beispiel Vergleichsoperationen, die für Maps notwendig sind, oder das Setzen von Netzwerkmasken, um das IP-Mapping zwischen Interfaceadresse und Quelladresse von Gruppenverwaltungsnachrichten durchzuführen (siehe Abschnitt 3.6).

Die Netzwerkmasken, die Interfaceadressen und weitere Zustände eines Interfaces, wie zum Beispiel, ob ein Interface aktiv ist und ob ein Netzwerkkabel angeschlossen ist, müssen für jedes Interface ausgelesen werden.

Weiterhin ist aufgrund der Erweiterungen für PMIPv6 gefordert, dass mehrere isolierte Proxy-Instanzen startbar sind. Es besteht der Konflikt, dass genau ein Socket das MRT-Flag (Multicast Routing Flag) halten darf (siehe Abschnitt 3.5.2), aber jede Proxy-Instanz solch ein Socket mit dem MRT-Flag benötigt. Zur Lösung werden die Aufgabenbereiche, für die das Flag benötigt wird, voneinander entkoppelt und in eigenständige Module ausgelagert. Dazu gehört das Empfangen von Multicast-Nachrichten, das in ein Modul Receiver ausgelagert wird. Ein weiteres Modul wird für das Routing erstellt. Beiden Modulen werden das MRT-Socket nutzen. Da die Module mit unterschiedlichen Fuktionalitäten vom Kernel arbeiten, beeinträchtigen sie sich nicht gegenseitig.

Das Verschicken von Multicast-Nachrichten wird vom Linux Kernel nur teilweise unterstützt. Das Senden von Join-/Leave-Nachrichten (siehe Abschnitt 3.4) übernimmt der Kernel, was dem Host-Teil der Gruppenverwaltungen entspricht (siehe Abschnitt 2.3). Das

Erstellen und Senden von Membership Queries (General Query oder Group-Specific Query) wird stattdessen unabhängig vom Kernel mit Raw-Sockets gelöst.

Der Multicast-Proxy benötigt eine größere Anzahl von Timern, die voneinander unabhängig sind. So wird das *Query Interval* für jede Proxy-Instance benötigt, das *Unsolicited Report Interval* und das *Last Member Query Interval* temporär für bestimmte Gruppenmitgliedschaften. Da der Linux Kernel keine passenende Lösung zur Verwaltung der Zeiten zur Verfügung stellt, wird das Verwalten von Zeiten in einem weiteres Modul Timer ausgelagert.

Die Verbindung von einem Mobile Node (MN) zu seinem Mobile Acces Gateway (MAG) kann zu jedem Zeitpunkt unterbrochen werden, was dem ziehen eines Netzwerksteckers an einem Downstream entspricht. Dies muss der Multicast-Proxy mitbekommen, um zum Beispiel die zum Downstream gehörenden Regeln für das Weiterleiten von Gruppendaten aus dem Linux Kernel zu löschen.

4.4 Konzept

Der Multicast-Proxy wurde in kleinere Aufgabenbereiche unterteilt und in Modulen angeordnet (siehe Abbildung 12). Ein übergreifendes Modul Proxy, welches das MRT-Socket hält, startet und beendet eine Menge von Proxy-Instanzen. Jede dieser Instanzen hat ein Modul Sender, über das alle notwendigen Multicast-Nachrichten (Join, Leave, General Query und Group-Specific Query) versendet werden, und alle Proxy-Instanzen haben Zugriff auf die Module Receiver, Routing und Timer. Die Kommunikation zwischen Proxy-Instanzen und den Modulen wird ähnlich dem Actor Pattern gehandhabt. Es werden über synchronisierte Queues Nachrichten ausgetauscht. Durch dieses Konzept werden die einzelnen Ressourcen und Datenstrukturen der Proxy-Instanzen zentral synchronisiert.

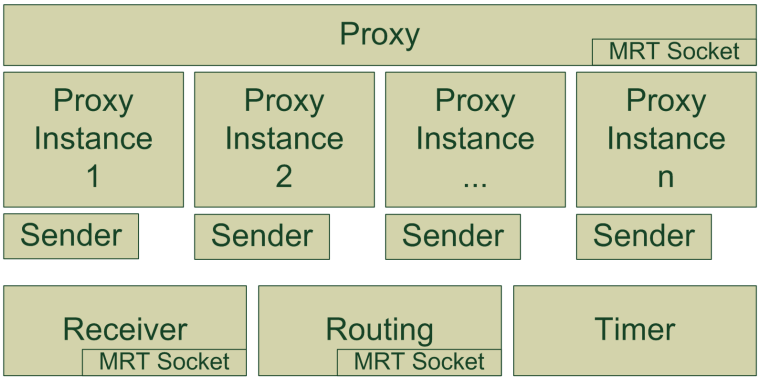


Abbildung 12: Übersicht des Aufbaus

Die Aufgabenbereiche der Module werden in der folgenden Auflistung zusammengefasst.

Proxy: Beim Start des Multicast-Proxys ließt das Modul Proxy eine Konfigurationsdatei ein. Die Datei enthält die zu nutzende IP Version und welche Up- und Downstream

Interfaces zu welcher Proxy-Instanz gehören. Die Konfiguration wird geprüft und so verhindert, dass ungültige Interfaces geladen oder mehrfach verwendet werden. Daraufhin werden die Module Receiver, Routing und Timer initialisiert, gestartet und das MRT-Socket übergeben. Im nächsten Schritt werden die Proxy-Instanzen mit den vorkonfigurierten Up- und Downstreams instanziiert. Zum Beenden des Multicast-Proxys schickt das Modul Proxy eine Kill Nachricht an alle anderen Module und Proxy-Instanzen.

Proxy-Instanz: Eine Proxy-Instanz repräsentiert einen vollständigen Multicast-Proxy. Jede Instanz hat ein Modul Sender und Zugriff auf die Module Receiver, Routing und Timer. Eine Instanz arbeitet mit einer von den anderen Instanzen unabhängigen Datenstruktur. Diese dient zur Verwaltung der Gruppenmitgliedschaften und den Regeln zum Weiterleiten von Gruppendaten. Zu einer Instanz gehört genau ein Upstream und mindestens ein Downstream, weitere Interfaces können zu beliebigen Zeitpunkten hinzugefügt oder entfernt werden. Der interne Ablauf wird in Abschnitt 4.4.2 und die genutzte Datenstruktur in Abschnitt 4.4.3 erläutert.

Sender: Ein Modul Sender generiert und versendet alle notwendigen Multicast-Nachrichten zur jeweiligen Proxy-Instanz. Zum Upstream werden Join- und Leave-Nachrichten und zu den Downstreams Membership Queries verschickt.

Receiver: Der Receiver nutzt das MRT-Socket, um Multicast-Nachrichten und Cache Miss Nachrichten, welche über das Fehlen von Routing-Einträgen informieren (siehe Abschnitt 3.6), zu empfangen. Proxy-Instanzen registrieren ihre Interfaces beim Receiver, welcher beim Empfangen einer Multicast-Nachricht von einem Interface eine Mitteilung an die zugehörigen Proxy-Instanzen schickt. Wobei über das Source Address Mapping bestimmt wird von welchem Interface die Nachricht emfpangen wurde (siehe Abschnitt 3.6). Unbekannte Nachrichten und Nachrichten, die keinen Treffer beim Source Address Mapping erzielen, werden verworfen.

Routing: Das Modul Routing verarbeitet Anfragen von Proxy-Instanzen. So setzt oder löscht es mithilfe des MRT-Sockets virtuelle Interfaces und Regeln zum Weiterleiten von Gruppendaten.

Timer: Proxy-Instanzen setzen Reminder (Erinnerungen) im Modul Timer. Ein Reminder enthält eine Zeit und eine Nachricht. Ist die für den Reminder vordefinierte Zeit abgelaufen, sendet das Modul Timer die im Remind enthaltene Nachricht zur jeweiligen Proxy-Instanz, wodurch sich zeitabhänges Verhalten definieren lässt. Für Details der Benutzung des Moduls Timer siehe Abschnitt 4.4.2.

4.4.1 Interaktion mit den Modulen

Die Interaktionen zwischen Proxy-Instanzen und den Modulen Timer, Receiver, Routing und Sender finden auf zwei Wegen statt (siehe Abbildung 13). Alle Nachrichten für eine

Proxy-Instanz werden in einer synchronisierten Message-Queue abgelegt und sequenziell verarbeitet (symbolisch durch gestrichelte Pfeile gekennzeichnet). Nachrichten von einer Proxy-Instanz an die Module werden direkt über Methodenaufrufe weitergegeben (durchgezogene Pfeile), eine Ausnahme bildet das Routing, denn auch das nutzt eine Message-Queue. Diese Nachrichten enthalten Events, auf die Proxy-Instanzen reagieren, und Anweisungen, die Module ausführen.

Proxy-Instanzen werden vom Modul Proxy erstellt. Events, die ein Modul Proxy bei einer Proxy-Instanz auslösen kann, sind `Add Interface`, `Del Interface`, `Debug` und `Exit`. `Add Interface` und `Del Interface` erweitern beziehungsweise löschen ein Downstream dynamisch von einer Proxy-Instanz. Mithilfe dem Event `Debug` werden aus allen Proxy-Instanzen die internen Datenstrukturen zusammengefasst und synchronisiert ausgegeben. Das Event `Exit` beendet eine Proxy-Instanz.

Das Modul Receiver wird vom Modul Proxy erstellt und je nach geforderter IP-Version als IGMP oder MLD Modul instanziiert. Die Proxy-Instanzen registieren ihre Interfaces beim Receiver. Daraufhin kann an der jeweiligen Proxy-Instanz für das Interface die Events `Join`, `Leave` und `New Source` auftregen. Tritt das Event `Join` oder `Leave` auf, wurde eine Join-Nachricht beziehungsweise eine Leave Nachricht an einem registrierten Interface empfangen und das `New Source` Event tritt beim Empfang einer Cache Miss Nachricht vom Linux Kernel auf.

Jede Proxy-Instanz erstellt ein eigenes Modul Sender mit einer IGMP oder einer MLD Instanz. Der Sender verschickt nach Anweisung seiner Proxy-Instanz Joins, Leaves, General Queries und Group-Specific Queries. Die Kommunikation mit dem Sender ist einseitig und erfordert keinen Rückkanal, wodurch das Modul Sender keine Events bei seiner Proxy-Instanz auslöst.

Das Modul Timer nimmt von den Proxy-Instanzen Events als Reminder auf, die nach einer Zeitverzögerung zur Message-Queue der jeweiligen Proxy-Instanz zurück geschickt werden. Reminder, die ein Timer aufnimmt, enthalten folgende Events: `Send GQ to all`, `Send GSQ` und `Del Group`. Diese Events sind im Abschnitt 4.4.2 beschrieben.

Das Modul Routing verfügt über eine eigene Message-Queue zum Aufnehmen der Anweisungen add/del vif und add/del route. Es löscht oder erstellt virtuelle Interfaces beziehungsweise Regeln zum Weiterleiten von Gruppendaten. Die Kommunikation mit dem Modul Routing ist einseitig und erfordert keinen Rückkanal, wodurch ebenfalls keine Events bei einer Proxy-Instanz ausgelöst werden.

4.4.2 Interner Ablauf einer Proxy-Instanz

Eine Proxy-Instanz wird mit einem Upstream und einer beliebigen Anzahl von Downstreams gestartet. Jedes Interface wird anschließend registriert. Dazu wird beim Modul Routing ein virtuelles Interface erstellt (siehe 3.5.3) und das Interface beim Receiver angemeldet. Bei den Downstream Interfaces wird zusätzlich die All-Routers Gruppe abonniert (siehe Abschnitt 3.6) und General Queries im *Startup Query Intervall* verschickt.

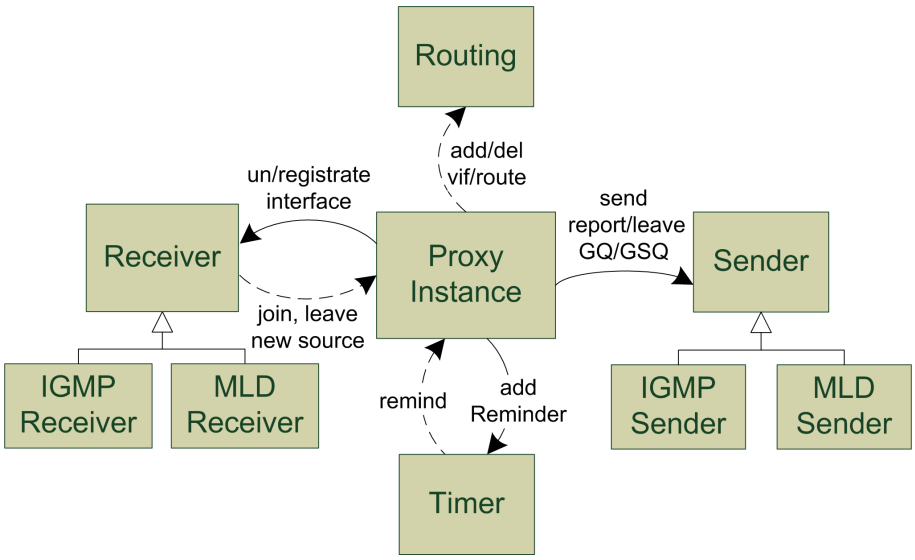


Abbildung 13: Übersicht der Interaktionen einer Proxy-Instanz mit den Modulen

Anschließend wartet die Proxy-Instanz auf das Eintreffen von Events in seiner sychnronisierten Message-Queue. Es können Events von den Modulen Receiver, Timer und Proxy eintreffen.

Receiver Events: Bei Events, die das Modul Receiver an eine Proxy-Instanz schickt, handelt es sich um ein `Join`, ein `Leave` oder eine `New Source` Event. Beim Empfangen dieser Events wird die interne Datenstruktur und das Routing aktualisiert. Bei den Events `Join` und `Leave` wird wird die Aggregation der Multicast-Zustände zum Upstream durchgeführt (siehe Abschnitt 2.4).

Timer Events: Bei Events die das Modul Timer an eine Proxy-Instanz schickt, handelt es sich um `Send GQ to all`, `Send GSQ` und `Del Group`.

`Send GQ to all` wird periodisch im [Query Intervall] verschickt und bewirkt, dass ein General Query von jedem Downstream Interface versendet und die interne Datenstruktur inklusive Routing aktualisiert wird.

`Del Group` löscht Gruppen aus der internen Datenstruktur, beeinflusst dadurch den Upstreamstatus und das Routing. Aufgerufen wird `Del Group` von der Auswertung der Events `Send GQ to all` und `Send GSQ`, um eine Gruppe endgültig aus der internen Datenstruktur zu entfernen.

`Send GSQ` wird durch das Event `Leave` vom Modul Receiver initiiert und beeinflusst ausschließlich die interne Datenstruktur.

Proxy Events: Bei den Events, die das Modul Proxy an eine Proxy-Instanz schickt, handelt es sich um `Add Interface`, `Del Interface`, `Debug` und `Exit`. Für Details siehe Abschnitt 4.4.1.

4.4.3 Datenstruktur einer Proxy-Instanz

Die interne Datenstruktur einer Proxy-Instanz bildet das Querier-Verhalten der Gruppenverwaltung (siehe Abschnitt 2.3) in einer Tabellendarstellung ab. Dieser Automat wird für jedes Downstream Interface und für jede Multicast-Gruppe erstellt und als Eintrag in der Tabelle gespeichert (siehe Tabelle 10). Die Tabelle enhält als Key das Tupel (Netzwerk-Interface, Multicast-Gruppe), welche jeweils in einem bestimmten Zustand sind. Der Zustand besteht aus dem Tupel (Flag, Counter). Für die Gruppenverwaltung stehen die Flags RUNNING, RESPONSE STATE, WAIT FOR DEL zur Verfügung. Die Verwendung der Flags wird durch die Abbildungen 14, 15 und 16 beschrieben.

Interface	Gruppe	Counter	Flag
eth0	239.99.99.99	2	RUNNING
eth0	239.0.0.15	1	RESPONSE STATE
eth1	239.0.0.15	0	WAIT FOR DEL

Tabelle 10: Beispiel einer Proxy-Instanz Datenstruktur

In Abbildung 14 sendet im ersten Schritt die Proxy-Instanz ein General Query zu seinen Downstream Interfaces. Ein Host antwortet auf dem Interface eth0 mit einem Join auf Gruppe G1. Dies wird mit einem Eintrag in der Tabelle vermerkt, dafür wird das Flag RUNNING mit dem Counter in Größe der Robustheitsvariable gesetzt (per default 2). Beim Versenden des nächsten General Querys wird der Counter bei allen Einträgen mit diesem Flag um eins dekrementiert. Sendet der Host anschließend eine neue Join-Nachricht, wird der Counter wieder auf Größe der Robustheitsvariable gesetzt. Empfängt die Proxy-Instanz keine Join-Nachricht, so wird nach dem Versenden des Nächsten General Querys der Counter wieder dekrementiert. Ist der Wert des Counters Null, wird das Flag auf WAIT FOR DEL gesetzt. Vergeht anschließend ein Zeitraum von [QUERY RESPONSE INTERVAL], ohne das eine Join-Nachricht empfangen wurde, wird der Eintrag aus der Tabelle gelöscht (siehe Abbildung 15).

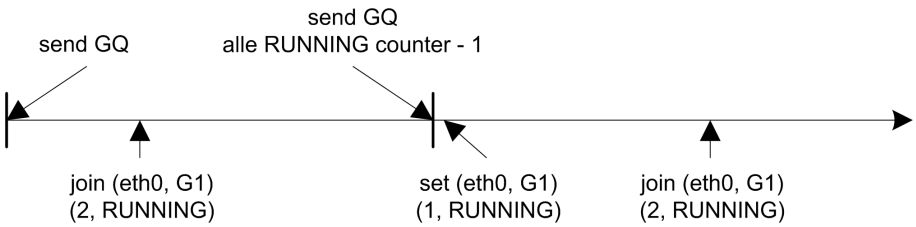


Abbildung 14: Anwendung des Flags RUNNING

In Abbildung 16 verlässt der Host die Gruppe G1 und das Flag wird auf RESPONSE STATE und der Counter auf [Last Member Query Count], welches per default der Robustheitsvariable entspricht, gesetzt. Daraufhin sendet die Proxy-Instanz einen Group-Specific Query, um zu prüfen, ob andere Host in diesem Subnetzwerk Interesse an Gruppe G1 haben.

Abbildung 15: Anwendung des Flags WAIT FOR DEL

Dieser Vorgang wird zwei mal wiederholt, was dem [Last Member Query Count] entspricht. Wird in dieser Zeit keine Join-Nachricht für Gruppe G1 empfangen, wird der Eintrag aus der Tabelle gelöscht.

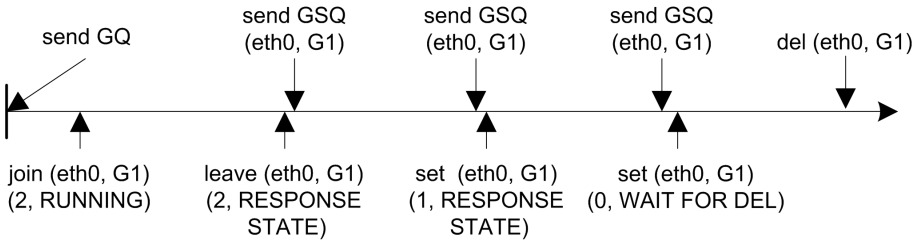


Abbildung 16: Anwendung des Flags RESPONSE STATE

Durch dieses Verfahren werden die Multicast-Zustände aller Downstream Interfaces abgebildet.

4.4.4 Weiterleiten von Gruppendaten

In diesem Abschnitt wird das Weiterleiten von Gruppendaten algorithmisch erläutert wobei die Erkenntnisse aus Abschnitt 2.4 und 3.5.4 genutzt werden. Da keine Wildcards für die Quelle im Linux Routing Cache eingerichtet werden können, wird für jede Quelle eine eigene Regel zum Weiterleiten von Gruppendaten gesetzt. Neue Quellen macht der Kernel über Cache Miss Nachrichten bekannt. Diese Nachrichten enthalten die Quelladresse, die Multicast-Gruppen Adresse und das Netzwerk-Interface. Mit der Cache Miss Nachricht und der Datenstruktur der Proxy-Instanz stehen alle Informationen zur Verfügung, um Regeln zum Weiterleiten von Gruppendaten berechnen und eintragen zu können.

Das Auftreten der Events `Join`, `Del Group`, `Send GQ to all` und `New Source` erfordern eine Anpassung der Regeln zum Weiterleiten von Gruppendaten. `Leave` und `Send GSQ` beeinflussen das Weiterleiten nicht direkt, denn diese Events ändern zwar den Gruppenzustand, aber erst beim Löschen der Gruppe durch `Del Group` werden die Regeln zum Weiterleiten von Gruppendaten angepasst.

Für die folgenden Erläuterungen der einzelnen Events wird ein Multicast-Proxy mit einem Upstream u und den Downstreams d1, d2 und d3 genutzt.

new source G1 (Upstream): Von einer für die Gruppe G1 neuen Quelle wird am Upstream eine Datenpaket empfangen. Der Linux Kernel informiert den Multicast-Proxy mit einer Cache Miss Nachricht über diese Neuerung. Der Proxy durchsucht die Downstreams, in diesem Fall d1, d2 und d3, nach der abonnierten Gruppe G1. Alle Downstreams, an denen Gruppe G1 abonniert wurde, werden als Output Interfaces an den Linux Kernel übergeben. So sieht die Regel zum Weiterleiten von Gruppendaten wie folgt aus:
(Upstream u, Source Address, Gruppe G1) ==> (Downstream d2, Downstream d3).

new source G1 (Downstream): Am Downstream d1 wird eine neue Quelle, der Gruppe G1, vom Kernel gemeldet. Daraufhin wird eine neue Regel zum Weiterleiten von Gruppendaten erstellt. Alle Downstreams, außer d1, an denen die Gruppe G1 abonniert wurde, werden als Output Interface für diese Regel zum Weiterleiten von Gruppendaten angegeben und zusätzlich das Upstream Interface u.

join G1: Wird eine Join-Nachricht für die Gruppe G1 am Downstream d1 empfangen und ist diese Gruppe nicht in der internen Proxy Datenstruktur für das Interface d1 eingetragen, werden alle Regeln zum Weiterleiten von Gruppendaten, die die Gruppe G1 abhandeln, um das Output Interface d1 erweitert.

del G1: Durch das Event `Del Group` für G1 wird aus der Proxy Datenstruktur die Gruppe G1 vom Interface d1 gelöscht, welches durch ein Leave Group in Verbindung mit mehreren Group-Specific Queries oder durch ein General Query ausgelöst wird (siehe Abschnitt 4.4.2). Es werden alle Regeln zum Weiterleiten von Gruppendaten, die die Gruppe G1 abhandeln, um das Output Interface d1 gemindert. Besitzt eine Regel keine Output Interfaces mehr, wird sie gelöscht.

send GQ: Der General Query wird in regelmäßigen Abständen verschickt (per default alle 125 Sekunden). Dieses Event wird genutzt, um die eingetragenen Regeln zum Weiterleiten von Gruppendaten im Kernel zu aktualisieren. Dies ist notwendig, denn der Kernel schickt zwar Cache Miss Nachrichten zum Multicast-Proxy, falls ein neue Multicast-Quelle entdeckt wurde, aber nicht wenn der Multicast-Proxy das Senden von Gruppendaten einstellt. Aufgrund dieses Problems werden alle Regeln zum Weiterleiten von Gruppendaten periodisch gelöscht, um zu verhindern, dass ungenutzte Regeln nicht dauerhaft den Multicast Forwarding Cache blockieren. Aktive Regeln werden durch Cache Miss Nachrichten neu erstellt. In dem Zeitraum vom Löschen bis zum Neuerstellen der Regeln zum Weiterleiten von Gruppendaten, werden eingehende Multicast-Pakete vom Linux Kernel zwischengespeichert. So entsteht zwar eine Verzögerung beim Weiterleiten, aber kein Datenverlust.

5 Implementierung des Multicast-Proxys

In diesem Kapitel wird die Implementierung des Multicast-Proxys beschrieben. Zunächst werden dafür Klassen für die Kernelabstraktion und IP-Adressverwaltung beschrieben. Anschließend wird auf die Details der Kommunikation zwischen den Proxy-Instanzen und den Modulen mithilfe der Message-Queue eingegangen und darauf aufbauend die Verarbeitung der Nachrichten in der Proxy-Instanz. Anschließend wird die interne Proxy-Instanz Datenstruktur und die einzelnen Module beschrieben. Das vereinfachte Klassendiagramm aus Abbildung 17 soll hierbei als Überblick dienen.

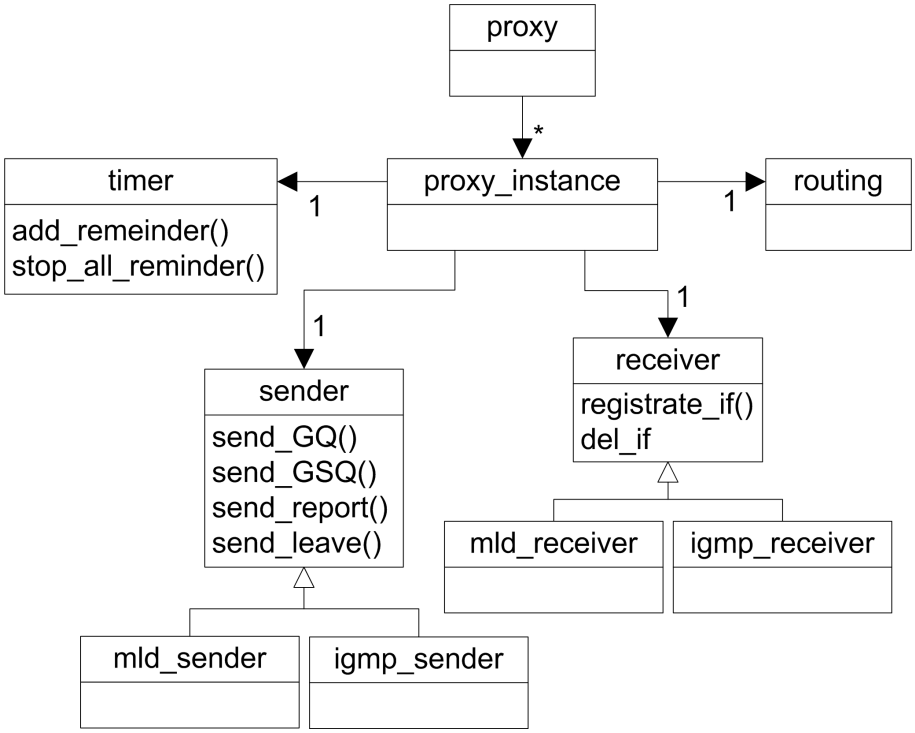


Abbildung 17: Aufbau des Multicast-Proxys als Klassendiagramm

Der Multicast-Proxy ist als multithreaded Anwendung implementiert worden, so arbeitet jede Proxy-Instanz autonom. Die Proxy-Instanzen nutzen eigene Threads, ebenso die Klassen Timer, Routing und Receiver.

5.1 Kernelabstraktion

In diesem Abschnitt werden die Klassen beschrieben, welche die Kernelfunktionen abstrahieren. Ein Teil der Methoden, die abstrahiert werden, wurden im Kapitel 3 beschrieben. Dazu gehöhren Join-, Leave-Nachrichten und jene zur Manipulation der Tabellen zum Weiterleiten von Gruppendaten. Weitere Methoden, die benutzt werden, sind zum Beispiel das

Setzen und Berechnen von Checksummen für den ICMPv6 und IGMP Header, das Hinzufügen von Extension Header an zu versendende Pakete, das Senden beziehungsweise Empfangen von UDP Paketen und das Setzten von Timeouts beim Empfangen. Diese und weitere Methoden sind in den Klassen mc_socket und mroute_socket, welche von einander abgeleitet sind, gekapselt. Aufgeteilt sind die Methoden anhand des benötigten Socket Typs. Methoden, die ein Datagramm Socket verwenden, sind in der Klasse mc_socket und Methoden welche Raw Sockets benötigen, in der Klasse mroute_socket zusammengefasst. Von einander abgeleitet werden können sie, weil alle Methoden, die ein Datagramm Socket benötigen, auch ein Raw Socket aktzeptieren. Gleichzeitig wurden die meisten Methoden IPv4 und IPv6 kompatibel entwickelt. Hierfür wird die zu nutzende IP Version wird dem Konstruktor mit übergeben. Der Klassenaufbau ist in Abbildung 18 gezeigt.

Bei der Socket-Abstraktion wurde der Copy Konstruktor in den private Scope gesetzt um zu verhindern, dass die Klasse unbeabsichtigt kopiert wird. Denn hier besteht die Gefahr, dass das Socket über den Destruktor von mc_socket geschlossen wird und beim Benutzen der Kopie daraufhin zu Fehlern führen. Referenzen und Pointer auf diese Klassen können weiterhin genutzt werden, da hier die Eigentümerschaft nicht weitergereicht wird.

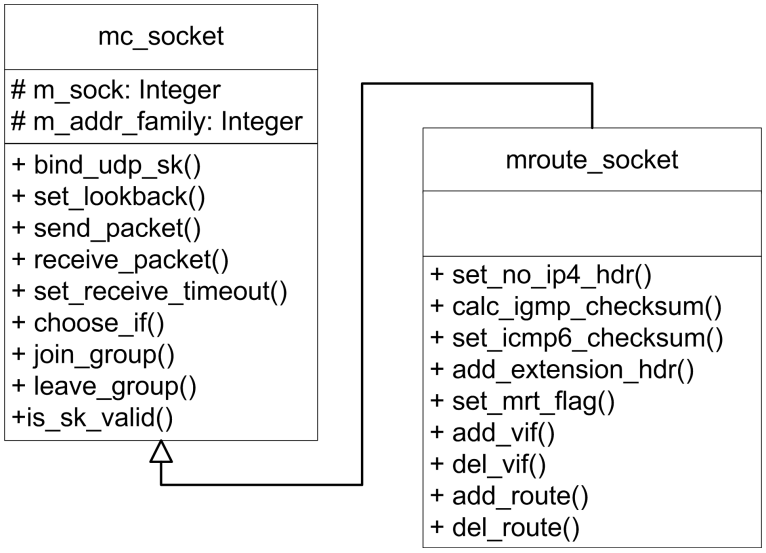


Abbildung 18: Abstraktion der Linux Kernel Funtkionen

Durch die Klasse if_prop werden Netzwerkinterface-Eigenschaften ausgelesen. Das Auslesen der Eigenschaften wurde in Listining 13 zusammengefasst. Die Kernelmethode getifaddrs() gibt eine verkettete Liste von IP Adressen und weiteren Eigenschaften zu allen Interfaces zurück. Ein Item der Liste enthält den Interface Namen, eine IPv4 oder Ipv6 Adresse, eine Anzahl Flags, und weitere Eigenschaften. Diese Liste wird als Map umkonfiguriert, um den Zugriff auf die IPv4 und IPv6 Adressen der einzelnen Interfaces zu beschleunigen. Eine nicht mehr benötige Interface Liste wird über freeifaddrs() freigegeben.

Es wird bewusst auf ioctl() Aufrufe verzichtet, um die Portierung auf andere Betriebsysteme zu vereinfachen.

Die Klasse sender benötigt eine Instanz der Klasse mroute_socket zum Versenden von Raw Socket Paketen. Die Klassen routing und receiver teilen eine Instanz der Klasse mroute_socket mit dem gesetzten MRT-Flag. Zusätzlich benötigen die Klasse receiver und die Klasse proxy Interface spezifische Eigenschaften wie IP-Adressen, Flags und Netzmaske welche jeweils eine Klasse if_prop zur Verfügung stellt.

Listing 13: Netzwerk Interface Eigenschaften auslesen

```
0  #include <ifaddrs.h>
   struct ifaddrs* m_ifa;
   if(getifaddrs(&m_ifa) < 0){ /*error*/ } //FILL

   struct ifaddrs* ifEntry=NULL;
5  for(ifEntry=m_ifa; ifEntry!=NULL; ifEntry=ifEntry->ifa_next) { //
     ITERATE
     //m_ifa->ifa_name
     //m_ifa->ifa_addr {ipv4, ipv6}
     //m_ifa->ifa_flags e.g. IFF_UP, IFF_RUNNING, IFF_POINTOPOINT,
        ...
           //flags defined in <net/if.h>
10
     //...
   }

   freeifaddrs(m_ifa) //FREE
```

5.2 IP-Adresstransparenz

Für den IP Versions neutralen Programmaufbau werden die verwendeten IP Adressen (IPv4/IPv6) in der Klasse addr_storage gespeichert. Die Klasse addr_storage kapselt das struct sockaddr_storage, welches IPv4 oder IPv6 Adressen aufnimmt. Es wurden Standardoperatioren überschrieben, so liest die Klasse verschiedene Adressstrukturen aus oder füllt diese. Weiterhin verarbeitet die Klasse addr_storage auch IP-Adressen im Klartext und gibt diese über den Streamingoperator aus. Zum Vergleichen von Adressen stehen die Operatoren gleich und kleiner als zur Verfügung, außerdem existiert eine Methode mask() zum Maskieren von Adressen. Zur Benutzung der Klasse siehe Listing 14.

Addr_storage wird von Proxy, Receiver und Sender genutzt.

Listing 14: Verwendungsmögichkeiten der Klasse addr_storage

```
0  addr_storage addr, addr1;
   //strings
   addr = "251.0.0.224"; //"ff02:231:abc::1";
   cout << "addr: " << addr << endl; // addr: 251.0.0.224
   //structs
5  struct sockaddr_storage s1 = //...
   struct in_addr s2 = //...
   struct in6_addr s3 = //...
   struct sockaddr s4 = //...
   addr = s1; //s2, s3, s4
10 s1 <<= addr; s2 <<= addr; //...
   //comparisons
   if(addr < addr1) { /*...*/}
   if(addr == addr1) { /*...*/}
   //other function
15 addr.mask(addr1);
```

5.3 Start des Multicast-Proxys

In diesem Abschnitt wird der Ablauf in der Klasse Proxy beschrieben. Eine Übersicht bietet die Methode init(). Sie beschreibt die Aufgaben, welche vor dem Start der Proxy-Instanzen ausgeführt werden. Der Programm Code wird in Listing 15 vorgestellt. Im ersten Schritt wird die Konfigurationsdatei geladen. Sie enthält die IP-Adressversion und die Interfaces für die jeweiligen Proxy-Instanzen. Bei einer fehlerhaften Konfiguration wird der Multicast-Proxy beendet, so dürfen zum Beispiel keine Interfaces doppelt genutzt werden. Die Methode init_vif_map() in Zeile 6 definiert für jedes Interface einen virtuellen Interface Index, der in den Multicast-Kernel-Tabellen genutzt wird. Die Abbildung von einem Netzwerk-Interface auf einen vif wird für einen schnellen Zugriff in einer Map gespeichert. Anschließend werden die Eigenschaften der Netzwerk-Interfaces geladen und vorkonfiguriert, hierbei wird das multicast und das allmulti Flag gesetzt (siehe Abschnitt 3.3). In Zeile 13 wird die Klasse mroute_socket initialisiert, für die aus der Konfigurationsdatei ausgelesene IP Version eingestellt und das MRT-Flag gesetzt. Daraufhin wird der Receiver, das Routing und der Timer erstellt und gestartet. Das Routing und der Timer können durch das implementierte Singletonpattern [EG94] nur einmal instanziiert werden, da alle Proxy-Instanzen mit denselben Instanzen arbeiten. Beim Receiver war dieses Vorgehen aufgrund der Vererbungshierarchie nicht möglich und diese werden deshalb als Parameter an die Proxy-Instanzen übergeben. Das Beenden des Multicast-Proxys erfolgt in umgekehrter Reihenfolge.

Listing 15: Start des Multicast-Proxys wird durch die Methode init() vorgegeben

```
bool proxy::init(std::string config_path){
    //load and evaluate config file
    if(!load_config(config_path) return false;
    if(!check_double_used_if() return false;

    //define a vif for every interface
    if(!init_vif_map()) return false;

    //configurate Interfaces
    if(!init_if_prop()) return false;
    if(!check_and_set_flags()) return false;

    //create socket with mrt-flag
    if(!init_mrt_socket()) return false;

    //create and start receiver
    if(m_addr_family == AF_INET){
        m_receiver = new igmp_receiver();
    }else if(m_addr_family == AF_INET6){
        m_receiver = new mld_receiver();
    }endif
    m_receiver->init(&m_mrt_sock);
    m_receiver->start();

    //start routing
    routing* r=routing::get_instance();
    r->init(&m_mrt_sock);
    r->start();

    //start timer
    timer* tim = timer::get_instance();
    tim->start();

    //start proxy instances
    if(!start_proxy_instances()) return false;

    return true;
}
```

5.4 Aufbau der Kommunikation

Für die Kommunikation zwischen den Modulen und der Proxy-Instanz wurde eine synchronisierte Message-Queue implementiert und ein Nachrichtenformat definiert. Darauf aufbauend wurde eine abstrakte Klasse worker entwickelt, welche einen Thread und die Message-Queue mit dem Nachrichtenformat enthält und so als Kommunikationseinheit genutzt wird. Damit anschließend die Proxy-Instanzen und Routing Nachrichten entgegen nehmen, wurden sie von der Klasse worker abgeleitet.

Synchronisierte Message-Queue

Die synchronisierte Message-Queue wurde selbst implementiert, da weder die Standard Template Library (STL) noch die Boost Library eine derartige Klasse zur Verfügung stellen. Weitere Libraries wurden aufgrund von zusätzlichen Abhängigkeiten vermieden. Für die synchronisierte Message-Queue wurde eine STL Message-Queue verwendet, die durch einen Mutex und eine Condition Variable aus der Boost Library threadsicher gemacht wurde (siehe hierzu Listing 16).

Nachrichten Format

Nachrichten, die zur Kommunikation im Multicast-Proxy genutzt werden, gehören dem Typ proxy_msg an. Proxy_msg hat zwei Member, einen Enum type, durch den der Inhalt der Nachricht definiert wird und einen Intrusive Shared Pointer aus der Boost Library. Der Shared Pointer speichert die Klasse intrusive_msg, welche als Basisklasse für die eigentlichen Nachrichten genutzt wird (siehe Abbildung 19).

Ein Shared Pointer hält die Eigentümerschaft für eine beliebige Klasse. Wird der Shared Pointer kopiert, inkrementiert er einen Referenzzähler anstatt die Klasse tatsächlich zu kopieren und Aufrufe des Destruktors vom Shared Pointer dekrementieren den Referenzzäler. Hat der Referenzzähler den Wert Null, wird die Klasse, welche vom Shared Pointer gehalten wird, vom Heap gelöscht. Intrusive bedeutet, dass die Klasse die vom Shared Pointer gehalten wird, den Referenzzähler selbstständig verwalten kann, was durch die Methoden intrusive_ptr_release() und intrusive_ptr_add_ref() übernommen wird (siehe Listing 17).

Auf diese Weise werden zwar noch Nachrichten zwischen den Modulen kopiert, aber da die Nachricht ein Pointer und ein Enum der den Nachrichteninhalt beschreibt, umfasst, bleibt der Kopieraufwand trotz beliebiger Nachrichtengröße konstant.

Nachrichtenverarbeitung in der Proxy-Instanz

Die Proxy-Instanz ist von der Klasse worker abgeleitet, welche einen Workerthread und die synchronisierte Message Queue zur Verfügung stellt. In dem Thread der Klasse worker werden Nachrichten sequentiell entgegengenommen und verarbeitet (siehe Listing 18).

Listing 16: Synchronisierte Message-Queue

```
0  #include <boost/thread/pthread/mutex.hpp>
   #include <boost/thread/pthread/condition_variable.hpp>
   #include <queue>

   template< typename T>
5  class message_queue{
   private:
        message_queue();
        queue<T> m_q;
        boost::mutex m_global_lock;
10      boost::condition_variable cond_empty;
   public:
        ...
   };

15 template< typename T>
   void message_queue<T>::enqueue(T t){
        boost::unique_lock<boost::mutex> lock(m_global_lock);
        m_q.push(t);
        cond_empty.notify_one();
20 }

   template< typename T>
   T message_queue<T>::dequeue(void){
        boost::unique_lock<boost::mutex> lock(m_global_lock);
25      while(m_q.size() == 0){
             cond_empty.wait(lock);
        }

        T t= m_q.front();
30      m_q.pop();
        return t;
   }
```

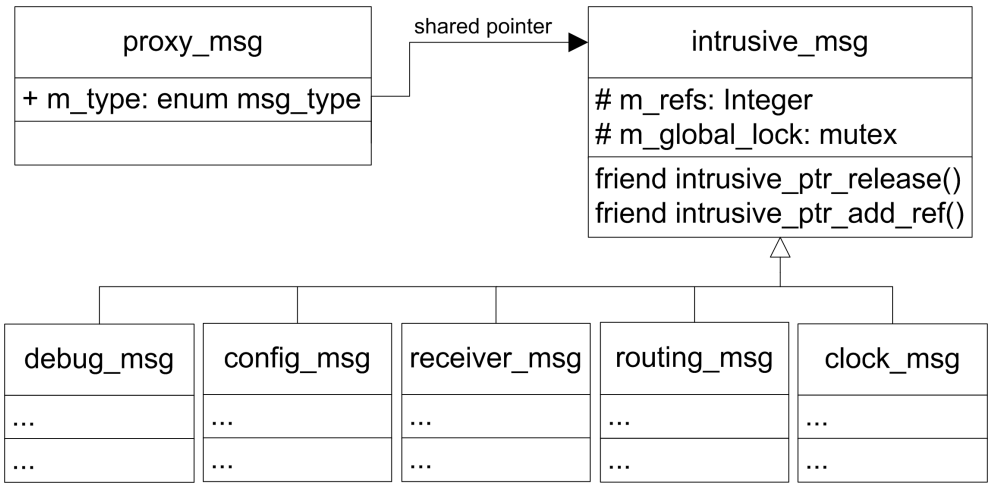


Abbildung 19: Aufbau der Nachrichten

5.5 Datenstruktur einer Proxy-Instanz

Das Konzept der Datenstruktur von den Proxy-Instanzen wurde in Abschnitt 4.4.3 erläutert und die darür erforderliche Tabelle 10 beschrieben. Die Umsetzung dieser Tabelle in C++ Code wird in diesem Abschnitt vorgestellt.

Aufgrund der Art, wie die Tabelle genutzt wird, lässt sie sich in Key und Value Spalten aufteilen und somit in einer Map speichern. Der Key besteht aus dem Tupel (Netzwerk Interface, Multicast-Gruppe) und der Value enthält das Tupel (Flag, Counter). Da es notwendig ist, über die einzelnen Netzwerk-Interfaces zu iterieren, aber der schnellen Map Zugriff über das Key Tupel auf das Value Tupel erhalten bleiben soll, wurde das Key Tupel in eine verschachtelte Map umgewandelt:

Map< Interface, **Map**< Gruppe, (Flag, Counter) > >

Die verwendete Map aus der STL hat beim Einfügen sowie beim Suchen eine Komplexität von O(log n) bezogen auf die Anzahl der Elemente in der Map.

Listing 17: Implementierung von intrusive_message

```
0  #include <boost/intrusive_ptr.hpp>
   #include <boost/thread.hpp>

   struct intrusive_message{
        intrusive_message(): refs(0) {}
5
   private:
        int refs;
        boost::mutex m_global_lock;

10      friend inline void intrusive_ptr_release(struct
           intrusive_message* p){
               p->m_global_lock.lock();
               if(--p->refs == 0 ) {
                    p->m_global_lock.unlock();
                    delete p;
15                  return;
               }
               p->m_global_lock.unlock();
        }

20      friend inline void intrusive_ptr_add_ref(struct
           intrusive_message* p){
               boost::lock_guard<boost::mutex> lock(p->m_global_lock);
               p->refs++;
        }
   };
```

Listing 18: Verarbeitung von Nachrichten in einer Proxy-Instanz

```
0  //...
   proxy_msg m;
   while(m_running){
        m = m_job_queue.dequeue();
        switch(m.type){
5       case RECEIVER_MSG: {
              struct receiver_msg* t= (struct receiver_msg*) m.msg.get
                  ();
              handle_igmp(t);
              break;
        }
10      case CLOCK_MSG: //... handle_clock(t);
        case CONFIG_MSG: //... handle_config(t);
        case DEBUG_MSG: //... handle_debug_msg(t);
        case EXIT_CMD: //... m_running = false;
        }
15 }
   //...
```

6 Validierung und Testen

In diesem Kapitel wird beschrieben, wie und unter welchen Bedingungen der entwickelte Multicast-Proxy geprüft und getestet wurde.

6.1 Testumgebung

Für den Testaufbau standen zwei Testrechner mit jeweils zwei physikalisch vorhandenen Netzwerk-Interfaces zur Verfügung sowie ein Notebook mit einem Ethernetanschluss. Das genutzte Betriebssystem und der Testaufbau wird in den folgenden Abschnitten beschrieben.

6.2 Betriebssystem

Der Multicast-Proxy wurde auf dem Betriebssystem *Ubuntu 10.10 - Maverick Meerkat*, mit einem modifizierten Linux Kernel-Version *2.6.35* entwickelt. Da zum Entwicklungszeitpunkt die IPv6 Multicast-Routing-Funktionalität nur experimental zur Verfügung stand wurde der Kernel für diesen Zweck selbst compiliert. Ob der eigene Kernel IPv6 Multicast-Routing unterstützt, lässt sich unter dem Betriebsystempfad /proc/net/ einsehen. Befinden sich dort die Dateien ip6_mr_cache (Auflistung aller IPv6 Multicast-Routen) und

ip6_mr_vif (Auflistung aller virtuellen Multicast-Routing-Interfaces) nicht, so muss ein modifizierter Kernel[1] in das System eingebunden werden.

6.3 Debugging

Zum Debuggen wurde ein Tool genutzt, welches für das Projekt Hamcast [gro11] entwickelt wurde. Es erstellt für multithreaded Anwendungen Logdateien. So werden Informationen wie Debug-, Error- und Trace Nachrichten gesammelt, um den Programmablauf nachzuvollziehen.

6.4 Test Tools

Zum Testen des Multicast-Proxys wurde ein Testtool erstellt, welches Gruppenoperationen wie Abonnieren und Verlassen ausführt, Kernel- und Interfaceflags setzt und periodisch Gruppendaten verschicken und empfangen kann. Das Testtool unterstützt IPv4 und IPv6.

Außerdem wurde das Programm Wireshark zum Aufzeichnen des Netzwerkverkehrs verwendet.

6.5 Testaufbau

Zum Testen des Multicast-Proxys wurden mehrere Testszenarien erstellt.

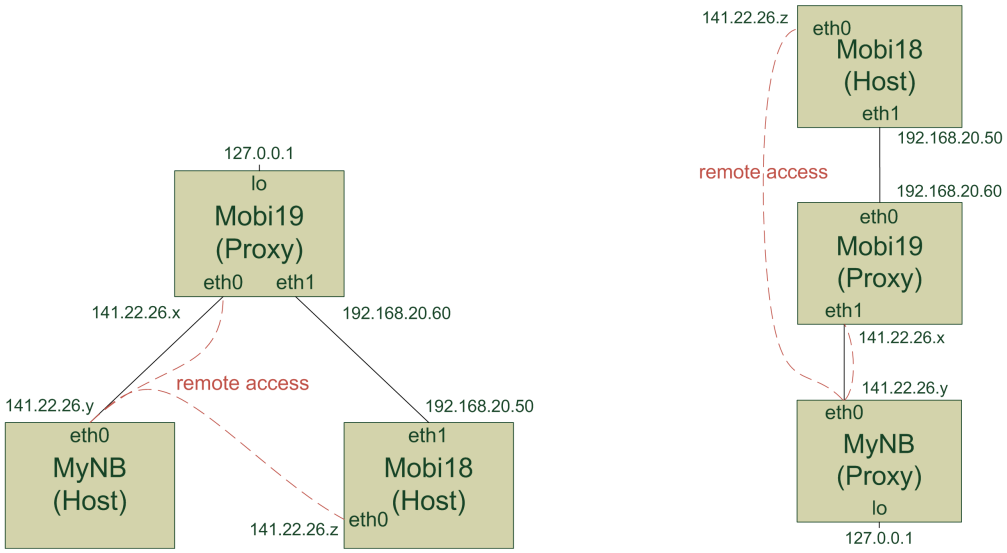
Zuerst wurden einzelne Proxy-Instanzen getestet. Dafür wurde der Testaufbau, wie in Abbildung 20a beschrieben, aufgebaut und die einzelnen Funktionsteile soweit wie möglich seperat getestet. Hierzu gehört der Querier an jedem Downstream, die Gruppenverwaltung, die Aggregation der Multicast-Zustände zum Upstream, das Berechnen und Eintragen der Regeln zum Weiterleiten von Gruppendaten, das Weiterleiten selbst sowie die Unterstützung der Multiinstanziierbarkeit.

Danach wurde das Hierarchisieren von Proxies getestet (siehe Abbildung 20b).

6.6 Testszenario

Für dieses Testszenario wird der Multicast-Proxy auf einem Testrechner mit Root-Rechten gestartet. Der Testrechner ließt eine Konfigurationsdatei ein, welche bestimmt, dass der Proxy die IP-Version vier nutzt und eine Instanz mit dem Interface *lo* als Upstream und *eth0* und *eth1* als Downstreams lädt. Der Multicast-Proxy versendet anschließend zu jedem Downstream in regelmäßigen Abständen General Queries. Abbildung 21 zeigt den Ausdruck eines General Querys des Analysetools Wireshark. Hier ist zu sehen, dass die Nachricht korrekt aufgebaut wurde. So entspricht zum Beispiel die Zieladresse (224.0.0.1) der All-Host Adresse, das TTL-Feld (Time-to-Live) ist auf eins gesetzt, beide Checksummen

[1] Tutorial: `http://wiki.ubuntuusers.de/Kernel/Kompilierung`



(a) Test von mehreren Instanzen und Downstreams

(b) Test von hierarchisierten Proxies

Abbildung 20: Testaufbauten zur Evaluierung des Multicast-Proxys

sind korrekt bestimmt worden und Wireshark identifiziert das Paket als Membership Query mit einem leeren Multicast-Adressfeld, was einem General Query entspricht.

Nun wird an beiden Downstreams die Testgruppe 239.99.99.99 abonniert. Die Tabelle 11 zeigt einen Debug-Ausdruck des Multicast-Proxys, welche belegt dass die Proxy-Instanz die Join-Nachrichten erkannt und in seine interne Datenstruktur eingetragen hat. Der Ausdruck zeigt die Datenstruktur der Proxy-Instanz mit dem Upstream *lo*. Als vif (virtueller Interface Index) wurde hierfür die null gewählt. Die Downstreams haben die vifs eins und zwei. Betreten wurde an beiden Downstreams die Testgruppe und eine Gruppe die für Multicast-DNS (224.0.0.251) benötigt, aber hier nicht weiter verwendet wird. Jede Multicast-Gruppe hat wie in Abschnitt 4.4.3 beschrieben ein Flag und einen Zähler.

Neben der Auflistung der Multicast-Gruppen zeigt die Tabelle noch einen zusätzlichen Eintrag. Am Interface *eth1* wurde für die Mutlicast-Testgruppe 239.99.99.99 eine Cache Miss Nachricht vom Linux Kernel empfangen. Das bedeutet, es existiert eine Multicast-Quelle mit der IP-Adresse 141.22.27.155 in dem zugehörigen Subnetzwerk.

Der Ausdruck der Kerneltabelle 12, für die Registrierung der Interfaces, zeigt dass drei Pakete empfangen und zu den Interfaces *lo* und *eth0* weitergeleitet wurden. Die zugehörige Regel zum Weiterleiten von Gruppendaten ist in Tabelle 13 zu sehen. Die Pakete der Testgruppe 239.99.99.99 (636363EE) mit der Quelladresse 141.22.27.155 (9B1B168D), empfangen am vif zwei (*eth1*), werden zu den vifs null und eins (*lo* und *eth0*) weitergeleitet. Dies entspricht den Grundlagen aus Abschnitt 2.4 .

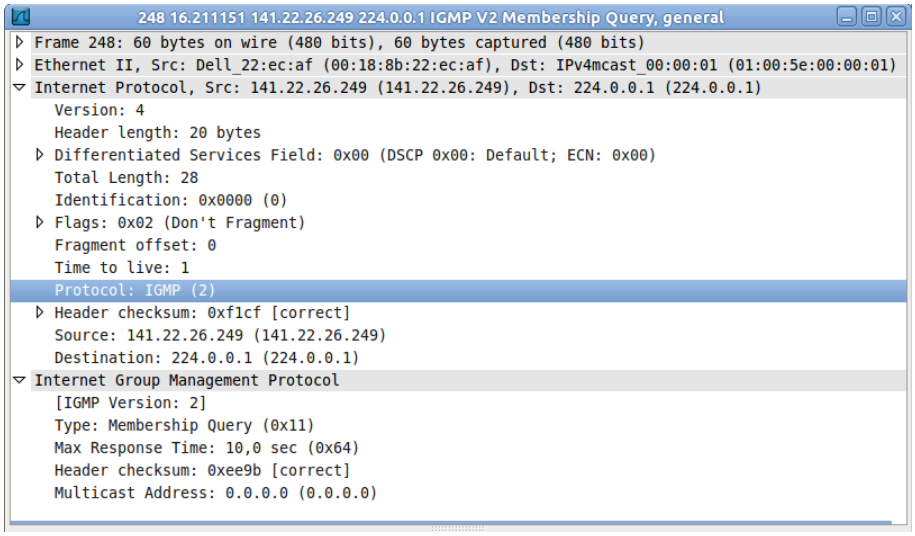


Abbildung 21: Wireshark Ausdruck: eines IPv4 General Querys

```
##-- instance upstream lo [vif=0] --##
     group addr

     -- downstream eth0 [vif=1] --
          group addr | robustness_counter | flag
          [0] 224.0.0.251     2    RUNNING
          [1] 238.99.99.99    2    RUNNING

     -- downstream eth1 [vif=2] --
          group addr | robustness_counter | flag
          [0] 224.0.0.251     2    RUNNING
          [1] 238.99.99.99    2    RUNNING
               src addr | robustness_counter | flag
               [0] 141.22.27.155   2    CACHED_SRC
```

Tabelle 11: Debugging Ausdruck des Multicast-Proxy

```
cat /proc/net/ip_mr_vif
Interface      BytesIn  PktsIn  BytesOut PktsOut Flags Local    Remote
 0 lo               0       0       204       3 00008 00000001 00000000
 1 eth0             0       0       204       3 00008 00000002 00000000
 2 eth1           204       3         0       0 00008 00000003 00000000
```

Tabelle 12: ip_mr_vif

```
cat /proc/net/ip_mr_cache
Group    Origin   Iif     Pkts    Bytes    Wrong Oifs
636363EE 9B1B168D 2          9      612        0  0:1    1:1
```

Tabelle 13: ip_mr_cache

6.7 Testergebnisse

Die Module Sender, Receiver und Routing wurden jeder für sich getestet. Der Sender versendet alle Multicast-Nachrichten korrekt an ein beliebiges Netzwerk-Interface, welches mit Wirheshark geprüft wurde. Das Modul Receiver empfängt Join- und Leave-Nachrichten und ordnet es dem korrektem Netzwerk-Interface zu. Das Modul Routing trägt die Regeln zum Weiterleiten von Gruppendaten korrekt in die Kerneltabellen ein und löscht sie auch wieder.

Das Testszenario (Abschnitt 6.6) zeigt, dass auch das Zusammenspiel funktioniert. So werden die Multicast-Zustände korrekt aggregiert sowie die Regeln zum Weiterleiten von Gruppendaten korrekt berechnet und eingetragen.

Weiterhin erkennt der Multicast-Proxy, wenn ein Netzwerkinterface seinen RUNNING-Zustand ändert. So wird über das jeweilige Interface, wenn es in den Zustand RUNNING übergeht eine General Query verschicken.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Design für ein Multicast-Proxy entwickelt, welches anschließend implementiert wurde. Der Multicast-Proxy erfüllt die klassischen Anforderungen aus RFC 4605 (IGMP/MLD Proxying) für IGMPv2 und MLDv1. So wurde im ersten Schritt ein Querier für jeden Downstream und die zugehörige Datenstruktur zur Verwaltung der Multicast-Gruppen entwickelt. Anschließend wurde mit Hilfe des Inhaltes der Datenstsruktur die Multicast-Zustände der Downstreams zum Upstream aggregiert. Daraufhin wurde das Weiterleiten von Gruppendaten über die performaten Multicast-Routing Tabellen des Linux Kernels realisiert. Hierfür wurde der Inhalt der Datenstruktur zur Verwaltung der Multicast-Gruppen genutzt, um die Regeln zur Weiterleitung von Gruppendaten zu berechnen und eintragen zu lassen. Neben den Basisanfunktionaltiäten unterstützt der Multicast-Proxy auch die Anforderungen aus dem Proxy Mobile IP Bereich. So unterstützt er mehrere unabhängige Proxy Domänen und es lassen sich die einzelnen Proxy Domänen dynamisch konfigurieren.

Der Multicast-Proxy wurde IP-versionsneutral entworfen. So unterstützt er neben dem Gruppenverwaltungsprotokoll IGMPv2 (IPv4) und auch MLDv1 (IPv6).

Allerdings konnten nicht alle Anforderungen aus dem Multicast-Proxy Standard aufgrund der begrenzten Zeit erfüllt werden. So unterstützt die Implementierung keine Querier Election und keine Abwärtskompatibilität zu IGMPv1. Des Weiteren konnte diese Arbeit nicht vollständig getesten werden. So wurden bisher nur auf Funktionstüchtigkeit getestet, aber keine Performanz-Tests durchgeführt. Es wurde in Abschnitt 4.4.4 die Problematik erläutert, dass alle eingetragene Regeln zum Weiterleiten von Gruppendaten periodisch gelöscht werden müssen und dadurch eine Zeitverzögerung beim Weiterleiten der Gruppendaten entsteht. Dies sollte unter anderem quantifiziert werden.

Der nächste Entwicklungsschritt des Multicast-Proxys ist die Integrierung von SSM (Source Spezific Multicast), sowie den Protokollen IGMPv3 und MLDv2 in die Implementierung. Dafür muss die Datenstruktur der Proxy-Instanz um eine Liste von Quelladressen für jede Gruppe erweitert werden und es muss das Senden und Empfangen von Multicast-Nachrichten für die neuen Nachrichtenformate angepasst werden.

Um die Zielgruppe für diese Anwendung zu erweitern, ist eine betriebssystemübergreifende Implementierung erforderlich. Dafür ist es notwendig, den Quellcode für MacOS und FreeBSD anzupassen. Zusätzlich ist die Veröffentlichung eines Releases und die Bekanntmachung in der einschlägigen Community (zum Beispiel SAMRG - IRTF und MULTIMOB - IETF) angebracht.

Zur Integration in komplexere Netzwerkarchitekturen wird die Unterstützung von Analyse- und Debugprotokollen unerlässlich. Dies erfordert die Erweiterung durch mtrace2 [AJFC11] und mrinfo, um beispielsweise Multicast-Pfade zu verfolgen und die Konfiguration und Eigenschaften von Multicast-Proxies auszulesen.

Literatur

[AJFC11] Hitoshi Asaeda, Tatuya Jinmei, William Fenner, and Stephen Casner. Mtrace Version 2: Traceroute Facility for IP Multicast. Internet-Draft – work in progress 08, IETF, January 2011.

[CB09] Johnny Egeland Constantin Baranov. igmpproxy. Website, 2009. Available online at `http://sourceforge.net/projects/igmpproxy/`; visited on August 10th 2011.

[CDG06] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443, IETF, March 2006.

[CDK+02] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, IETF, October 2002.

[CKHR05] Jonathan Corbet, Greg Kroah-Hartman, and Alessandro Rubini. In *Linux Device Drivers, 3rd Edition*, 2005.

[DC90] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Trans. Comput. Syst.*, 8(2):85–110, 1990.

[Dee89] Steve Deering. Host extensions for IP multicasting. RFC 1112, IETF, August 1989.

[DFH99] Stephen E. Deering, William C. Fenner, and Brian Haberman. Multicast Listener Discovery (MLD) for IPv6. RFC 2710, IETF, October 1999.

[dG98] Juan-Mariano de Goyeneche. Multicast over TCP/IP HOWTO. Website, 1998. Available online at `http://tldp.org/HOWTO/Multicast-HOWTO.html`; visited on April 11th 2011.

[EG94] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. In *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*, 1994.

[Fen97] William C. Fenner. Internet Group Management Protocol, Version 2. RFC 2236, IETF, November 1997.

[FHHS06] B. Fenner, H. He, B. Haberman, and H. Sandick. Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ('IGMP/MLD Proxying'). RFC 4605, IETF, August 2006.

[GLD+08] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil. Proxy Mobile IPv6. RFC 5213, IETF, August 2008.

[gro11] INET group. HAMcast - Hybrid Adaptive Mobile Multicast. Website, 2011. Available online at `http://www.realmv6.org/hamcast.html`; visited on August 10th 2011.

[HD06] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291, IETF, February 2006.

[HT02] B. Haberman and D. Thaler. Unicast-Prefix-based IPv6 Multicast Addresses. RFC 3306, IETF, August 2002.

[IAN11a] IANA. IPv4 Multicast Address Space Registry. Website, 2011. Available online at `http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xml`; visited on August 09th 2011.

[IAN11b] IANA. IPv6 Multicast Address Space Registry. Website, 2011. Available online at `http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml`; visited on August 10th 2011.

[Lah02] Abdelkader Lahmadi. gproxy. Website, 2002.

[Mas05] Jeroen Massar. ecmh - Easy Cast du Multi Hub. Website, 2005. Available online at `http://sourceforge.net/projects/ecmh/`; visited on August 10th 2011.

[Pel02] Matteo Pelati. Multicast Routing Code in the Linux Kernel. Website, 2002. Available online at `http://www.linuxjournal.com/article/6070`; visited on April 4th 2011.

[Per02] C. Perkins. IP Mobility Support for IPv4. RFC 3344, IETF, August 2002.

[PJA11] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. RFC 6275, IETF, July 2011.

[SH04] P. Savola and B. Haberman. Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address. RFC 3956, IETF, November 2004.

[SWK11] T. Schmidt, M. Waehlisch, and S. Krishnan. Base Deployment for Multicast Listener Support in Proxy Mobile IPv6 (PMIPv6) Domains. RFC 6224, IETF, April 2011.

[VC04] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810, IETF, June 2004.

[Wai07] David Waitzman. Multicast Routing, OpenBSD Programmer's Manual. Website, 2007. Available online at `http://resin.csoft.net/cgi-bin/man.cgi?section=4&topic=multicast`; visited on April 4th 2011.

[WPR+04] Klaus Wehrle, Frank Paehlke, Hartmut Ritter, Daniel Mueller, and Marc Bechler. In *The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel*, 2004.

[WZ01] R. Wittmann and M. Zitterbart. Multicast Communication, 2001.

8 Versicherung

*Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 3. November 2011	
Ort, Datum	Unterschrift