JAVA POLICY SERVER

Diplomarbeit

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von
Dragan Milic

2004

Leiter der Arbeit:
Prof. Dr. Torsten Braun, Institut für Informatik und angewandte Mathematik/Rechnernetze und Verteilte Systeme

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Durch die immer grössere Bandbreite, die Internetnutzern zur Verfügung steht, gewinnt das Internet auch als Medium für den Transport von Audio- und Videodaten an Bedeutung. Schon heute besteht eine grosse und stetig wachsende Zahl von Radio- und Fernsehprogrammen, die per Internet empfangen werden können. Die Verwendung des Internets für die Übertragung von Audio- und Videoinhalten unterscheidet sich von den klassischen Übertragungswegen (Übertragung durch die Luft, Kabelfernsehen, Satellitenübertragung) in folgenden Punkten:

- An der Datenübertragung sind mehrere voneinander unabhängige Internet-Service-Provider (im Folgenden: ISP) beteiligt.
- Das IP-Protokoll legt weder die verfügbare Bandbreite noch Grenzen für die bei der Zustellung der Daten entstehende Verzögerung oder die Wahrscheinlichkeit für ihre effektive Zustellung fest.

Die Bedeutung dieser Unterschiede wächst mit der Zunahme der Nutzung des Internets für die Übertragung kostenpflichtiger Inhalte. Obgleich die Nutzer (Empfänger) für die zahlreichen frei verfügbaren Inhalte nach wie vor keine Gebühr bezahlen müssen, wird mittlerweile bereits für die Nutzung einer ganzen Reihe von Inhalten eine Gebühr verlangt. Das Erheben einer Gebühr verpflichtet aber auch zur qualitativ einwandfreien Zustellung des Inhalts. Um eine solche qualitativ hochwertige Zustellung von Daten durch das Internet garantieren zu können, wurde das Konzept der so genannten Dienstgüte (Englisch: Quality-of-Service, im Folgenden: QoS) entwickelt. QoS ermöglicht die Sonderbehandlung der ausgewählten IP-Pakete durch Router, so dass für bestimmte Datenflüsse (zum Beispiel eine Audio- und Videoübertragung) bezüglich der verfügbaren Bandbreite, der Verzögerung bei der Datenübertragung und der Wahrscheinlichkeit der Datenzustellung genau festgelegte Garantien gegeben werden können.

Um QoS für alle Router, die an der Datenübertragung beteiligt sind, einstellen zu können, wurde das RSVP-Protokoll [1] definiert. Das RSVP-Protokoll ermöglicht eine benutzerspezifische Definition von QoS bezüglich der Datenflüsse, die von einem Benutzer empfangen werden. Da bei QoS bestimmte limitierte Ressourcen wie zum Beispiel die Bandbreite von vornherein reserviert werden, ist es notwendig sicherzustellen, dass mit RSVP wirklich nur diejenigen Ressourcen reserviert werden, für die auch bezahlt wird. Die Entscheidung, ob ein Benutzer das Recht zur Betätigung einer Reservierung hat, muss dann von jedem Router, der am RSVP-Protokoll beteiligt ist, getroffen werden. Da die Überprüfung der Zugangsberechtigung unter Umständen sehr aufwendig ist, wird diese Entscheidung von den Routern selbst an eine zentrale Instanz innerhalb eines ISPs delegiert: den Policy-Server. Die Kommunikation zwischen den Routern und dem Policy-Server ist durch das COPS-Protokoll [2] definiert.

1.2 Ziele der Diplomarbeit

Die Ziele dieser Diplomarbeit sind die folgenden:

- Die Erweiterung des RSVP-Protokolls durch ein Authentifizierungsschema, das sicherstellt, dass nur diejenigen Benutzer, die dazu berechtigt sind, die entsprechenden RSVP-Reservierungen vornehmen können.
- Die Erstellung eines COPS-Servers, der dieses Authentifizierungsschema implementiert.

1.3 Struktur der Diplomarbeit

Das nachfolgende zweite Kapitel der vorliegenden Arbeit beschreibt die verwendeten Konzepte und Protokolle (Multicast, RSVP, COPS und COPS-RSVP). Daran schliesst sich das dritte Kapitel an, in dem das angenommene Szenario der Datenübertragung erläutert wird. Im Anschluss daran wird eine neues, vom Autor der vorliegenden Arbeit entwickeltes Authentifizierungsschema definiert, dessen Vor- und Nachteile beschrieben und ein Vergleich zu anderen Authentifizierungsschemata gezogen. Die Architektur und das Design des im Zusammenhang hiermit ebenfalls neu entwickelten Java-Policy-Servers (der Implementierung des Policy-Servers in JAVA) und der Komponenten, die bei der Implementierung entstanden sind, werden im vierten Kapitel dargestellt. Auch die Ergebnisse der Leistungsmessungen des Policy-Servers und der COPS-Implementierung sowie deren Analyse werden in diesem Kapitel vorgestellt. Ein Weg, das Authentifizierungsschema anzuwenden, ohne die RSVP-Implementierungen auf den Routern und Empfängern verändern zu müssen, sowie eine universelle Implementierung des Authentifizierungsschemas für die RSVP-Router und Empfänger unter Linux werden in Kapitel 6 beschrieben. Im siebten Kapitel finden sich schliesslich die Zusammenfassung der Ergebnisse und ein Ausblick.

Für die sprachliche Überarbeitung der vorliegenden Arbeit musste der Autor als Nicht-Muttersprachler des Deutschen Hilfe in Anspruch nehmen. Die linguistische Beratung und Unterstützung – inbesondere im Hinblick auf grammatische und Übersetzungsprobleme –, erfolgte durch Prof. E. Hentschel (Bern) und Dr. K. Carstensen (Zürich), denen an dieser Stelle herzlich gedankt sei.

2 Verwendete Konzepte und Protokolle

Um das angenommene Szenario und die Entscheidungen besser verständlich zu machen, die beim Entwurf des Authentifizierungsschemas und der Implementierung des Policy-Servers getroffen wurden, ist es notwendig, zunächst einen Überblick über die Konzepte und Protokolle zu geben, die bei der Übertragung von Audio- und Videodaten über das Internet eine Rolle spielen. Deswegen werden die wichtigsten Konzepte und Protokolle im vorliegenden Kapitel kurz erläutert. Vorausgesetzt wird dabei nur die Kenntnis des IP-Protokolls (IPV4 [3] und IPV6[4]).

2.1 IP-Multicast

Im Gegensatz zu „Punkt-zu-Punkt“ Datenübertragungen (Englisch: Unicast), bei denen die Daten, die von einem Sender versendet werden, nur von einem einzelnen Empfänger empfangen werden können, ermöglicht das Multicast-Konzept eine Datenübertragung, bei der Daten, die von einem Sender versandt werden, von mehreren Empfängern empfangen werden können. Das Multicast-Konzept wurde eingeführt, um die Effizienz der Datenübertragung derselben Daten an mehrere Empfänger (zum Beispiel bei einer Videokonferenz) zu erhöhen. Der Vorteil von Multicast besteht in der Skalierbarkeit, denn der Sender muss die Daten unbhängig von der Anzahl der Empfänger nur eimal versenden. Bei IP-Multicast müssen sich alle Teilnehmer (Sender wie Empfänger) in einer Multicast-Gruppe anmelden. Dafür wird eine Version des IGMP-Protokolls verwendet (vgl. [5],[6],[7]). Eine Multicast-Gruppe hat eine eindeutige Adresse und diese wird als Zieladresse für den Versand von IP-Paketen verwendet. Dabei entsteht ein so genannter Multicast-Baum, der alle Pfade für den Empfang der Daten eines Senders beinhaltet. Ein Beispiel für einen solchen Multicast-Baum findet sich in Abbildung 1 (Seite 4). Die Daten, die von einem Sender an eine Multicast-Gruppe versandt werden, werden von den Routern nach Bedarf in mehrere Richtungen weitergeleitet (repliziert).

Ein Sender oder Empfänger kann einer Multicast-Gruppe zu jedem beliebigen Zeitpunkt beitreten oder sich aus ihr abmelden. Wenn man ihn im Zeitverlauf betrachtet, ist der Multicast-Baum für die Auslieferung der Daten also nicht konstant. Eine ausführliche Beschreibung von IP-Multicast findet sich in [8].

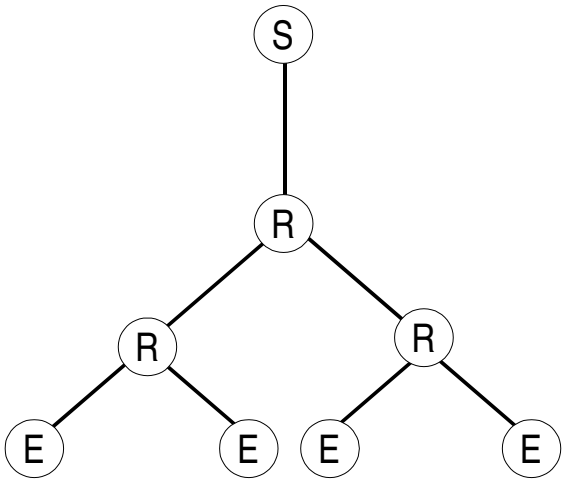


Abbildung 1: Ein Beispiel für einen einfachen Multicast-Baum. Die Bezeichnung der Komponenten: S = Sender, R = Router, E = Empfänger. Die Linien zwischen den Komponenten markieren den Pfad für die Auslieferung der Daten.

2.2 QoS

Das IP-Protokoll beinhaltet keine Garantien für die Auslieferung der IP-Pakete (Stichwort: „best effort“). Grundsätzlich gilt, dass IP-Pakete, die von einem Sender versandt werden, den Empfänger keineswegs notwendigerweise auch erreichen, und wenn sie ihr Ziel erreichen, kann dies mit einer Verzögerung beliebiger Länge erfolgen. Darüber hinaus müssen die versandten IP-Pakete den Empfänger auch keineswegs in der Reihenfolge erreichen, in der sie versandt wurden. Diese „Unsicherheit“ des IP-Protokolls wird durch darüber liegende Protokolle wie zum Beispiel TCP [9], durch den Einsatz von Mechanismen für die Detektion von Paketverlusten und durch die abermalige Übertragung verlorengegangener Pakete ausgeglichen. Der Einsatz solcher Protokolle reicht aber nicht aus, um den hohen Anforderungen an die Übertragung von Daten für verzögerungsempfindliche Applikationen gerecht zu werden, da die Detektion von Paketverlusten und die erneute Übertragung ihrerseits zu hohe Verzögerungen verursachen.

So kann beispielsweise ein Telefongespräch, das durch das Internet übertragen wird, durch die Verzögerung, die durch das Wiederversenden der verlorenen IP-Pakete entsteht, völlig unverständlich werden. Andererseits können, wenn man die Daten einfach ohne Fehlerkorrektur überträgt, wesentliche Teile des Gesprächs verloren gehen: die Gesprächspartner werden dann nur ein Teil von dem hören, was gesagt wurde. Um die Anforderungen an die Datenübertragung zu erfüllen, die nicht vom IP-Protokoll abgedeckt werden, wurde das im Vorigen bereits erwähnte Konzept „QoS“ (Abkürzung für „Quality of Service“, auf Deutsch auch: „Dienstgüte“) eingeführt. QoS ermöglicht die Definition von Parametern wie Bandbreite, maximale Verzögerung der Pakete, maximaler Datenverlust, die für den Datenfluss wichtig sind. Beim IP-Protokoll wird QoS

in Routern so implementiert, dass die IP-Pakete, die zu einem Datenfluss gehören, für den QoS definiert wurde, gesondert behandelt werden – das heisst mit höherer Priorität gegenüber den restlichen IP-Paketen – , so dass der Datenfluss innerhalb der vorgegebenen Parameter bleibt.

2.3 RSVP

Allein durch die Existenz von QoS ist die Frage, wie QoS in den Routern konkret eingestellt wird, natürlich noch nicht gelöst. In Fällen, in denen Datenübertragungen im Hinblick auf den Umfang der Daten sowie Anzahl und Adressen der Empfänger schon im Voraus genau fest stehen, können die QoS-Parameter für die entsprechenden Datenflüsse fest definiert werden. Bei Radio- und Fernsehübertragungen ist dies aber nicht der Fall – die Adressen der Empfänger sind im Allgemeinen nicht im Voraus bekannt. Um QoS in solchen Fällen einzustellen, wird ein Signalisierungsprotokoll benötigt. Ein solches Protokoll ist RSVP [1].

RSVP ist ein Internetkontrollprotokoll (ähnlich wie ICMP [10] oder IGMP), das die Definition von QoS (die Reservierung der benötigten Ressourcen) für die gewünschten Datenströme ermöglicht. Die Kommunikation erfolgt dabei durch Austausch von RSVP-Nachrichten. Eine RSVP-Nachricht wird per IP-Protokoll transportiert und hat die Protokollnummer 46. Das RSVP-Protokoll selbst dient nur der Signalisierung und transportiert keine Daten. Die Reservierungen werden von den Empfängern initiiert, was auf den ersten Blick kontraintuitiv erscheint, aber durchaus sinnvoll ist, weil die Empfänger diejenigen sind, die entscheiden, wie viele Ressourcen tatsächlich reserviert werden müssen. Die Eigenschaften des RSVP-Protokolls sind die folgenden:

- **Datenunabhängigkeit**: Das RSVP-Protokoll ist unabhängig vom Format der Daten, für die die Reservierung vorgenommen wird, denn das RSVP-Protokoll überträgt selbst keine Daten, sondern sorgt nur für die Signalisierung.
- **Soft-State**[1]: Der Zustand (State) der RSVP-Reservierung muss immer wieder erneuert werden. Falls ein Zustand über eine definierte Zeitspanne (Verfallszeit) hinweg nicht erneuert wird, werden die Ressourcen, die durch den Zustand beansprucht wurden, wieder zur anderweitigen Verwendung freigegeben.
- **Reservierung von Simplex-Flows**: RSVP-Reservierungen verlaufen grundsätzlich nur in einer Richtung (vom Sender zum Empfänger). Falls eine Duplex-Reservierung, also eine Reservierung in beide Richtungen, gewünscht wird, muss für beide Richtungen jeweils eine eigene Reservierung vorgenommen werden.
- **Uni- und Multicastunterstützung**: Das RSVP-Protokoll unterstützt sowohl Uni- als auch Multicast, und zwar unabhängig vom Routing-Protokoll.
- **Unterstützung für Router, die das RSVP-Protokoll nicht unterstützen**: Das RSVP-Protokoll wurde so konzipiert, dass es möglich ist, auch dann RSVP-Reservierungen zu generieren, wenn nicht jeder Router, der an der Datenübertragung teilnimmt, das RSVP-Protokoll unterstützt. Ein Mechanismus für die Entdeckung solcher „Lücken“ ist in RSVP vorhanden.

Ein Datenstrom ist im RSVP-Protokoll als eine RSVP-Session definiert. Eine RSVP-Session besteht aus der Empfängeradresse, der Protokollnummer und – falls das Protokoll das Konzept

1 Im Folgenden wird mehrheitlich der deutsche Begriff „Zustand“ verwendet, im Interesse terminologischer Eindeutigkeit jedoch gelegentlich auch der englischer Begriff „State“ hinzugefügt. Auf die Benutzung des deutschen Fremdwortes „Status“ wird verzichtet. Zu generellen Problemen bei der Wiedergabe englischer Informatik-Fachterminologie im Deutschen vgl. auch die Fussnote auf Seite 38 der vorliegenden Arbeit.

von Ports unterstützt – der Empfängerportnummer. Die Empfängeradresse ist im Fall von Unicast die IP-Adresse des Empfängers, im Fall von Multicast die Adresse der Multicast-Gruppe. Alle RSVP-Nachrichten in der Version 1 des RSVP-Protokolls beziehen sich immer auf eine RSVP-Session.

Innerhalb einer RSVP-Session werden die Datenströme nach Sendern unterschieden. Entsprechend können die Netzwerkressourcen je nach Reservierungsstil für jeden Sender unterschiedlich reserviert werden. Das RSVP-Protokoll teilt die Reservierungsstile nach zwei Kriterien ein: dem Reservierungsmodus und der Senderwahl.

- **Reservierungsmodus**

In der Version 1 von RSVP existieren zwei Typen von Reservierungsmodi: Einzelnutzung (im Folgenden: „Distinct“) und „gemeinsame Nutzung“ (im Folgenden: „Shared“) . Distinct-Reservierungen definieren getrennte Netzwerk-Ressourcen (Bandbreite, Verzögerung etc.) für jeden einzelnen Sender. Shared-Reservierungen definieren geteilte Netzwerkressourcen für alle Sender.

- **Senderauswahl**

Bei der Senderauswahl gibt es zwei Optionen: „Explicit“ und „Wildcard“. Bei der Explicit-Senderauswahl muss jeder Sender einzeln angegeben werden; mit der Wildcard-Senderauswahl werden dagegen alle Sender aus einer Session ausgewählt.

Aus der Kombination von Reservierungsmodus und Senderauswahl ergeben sich somit folgende Reservierungsstile:

- **Fixed-Filter (FF)**: Die FF-Reservierung definiert für jeden einzelnen Sender eigene Ressourcen.
- **Shared-Explicit (SE)**: Die SE-Reservierung definiert gemeinsame Ressourcen für mehrere Sender. Die so festgelegten Ressourcen werden dann von den Datenströmen geteilt.
- **Wildcard-Filter (WF)**: Die WF-Reservierung definiert gemeinsame Ressourcen für alle Sender, die innerhalb einer RSVP-Session senden. Der Unterschied zwischen SE- und WF-Reservierung ist, dass bei WF-Reservierung die Sender nicht explizit genannt werden müssen. Das bedeutet, dass jeder neue Sender, der zu senden beginnt, durch die WF-Reservierung bereits automatisch selektiert ist.

Schematisch lassen sich die Reservierungsstile folgendermassen darstellen:

	Reservierungsmodus		
Senderauswahl		*Einzeln*	*Gemeinsam*
	Explizit	Fixed-Filter (FF)	Shared-Explicit (SE)
	Wildcard	--	Wildcard-Filter (WF)

Jede RSVP-Nachricht hat folgendes Datenformat:

Version	Flags	Message Type	RSVP Checksum
Send_TTL		Reserved	RSVP Length
RSVP Objects			

Die Bedeutung der Felder ist dabei die folgende:

- **Version (4 Bit)**: Die Version des RSVP-Protokolls. In [1] wird die Version 1 definiert.
- **Flags (4 Bit)**: Die Flags der Nachricht. In der Version 1 des Protokolls wurden keine Flags definiert.
- **Message Type (8 Bit)**: Der Typ der Nachricht. Die Version 1 von RSVP definiert folgende Nachrichtentypen: PATH, RESV, RESVERR, PATHTEAR, RESVTEAR, RESVCONF.
- **RSVP Checksum (16 Bit)**: Die Internet-Prüfsumme der gesamten RSVP-Nachricht (inklusive des RSVP-Headers, wobei das RSVP-Checksum-Feld, also die RSVP -Prüfsumme, auf 0 gesetzt wird).
- **Send_TTL (8 Bit)**: Der „time to live“-Wert des IP-Pakets beim Versand der RSVP-Nachricht. Dieser Wert wird verwendet, um festzustellen, ob die RSVP-Nachricht durch einen Router, der das RSVP-Protokoll nicht unterstützt, transportiert wurde.
- **RSVP Length (16 Bit)**: Die Länge der RSVP-Nachricht (in Bytes) inklusive des RSVP-Headers).
- **RSVP Objects (RSVP Length – 8 Bytes)**: Eine Liste der RSVP-Objekte, die den Inhalt der RSVP-Nachricht darstellen.

Der Inhalt einer RSVP-Nachricht wird in Form von RSVP-Objekten dargestellt. Die Objekte können in beliebiger Reihenfolge innerhalb der RSVP-Nachricht auftreten.

Jedes RSVP-Objekt hat folgendes Format:

Length	Class Number	Class Type
Contents		

Die Bedeutung der Felder ist dabei:

- **Length (16 Bit)**: Die Länge des RSVP-Objekts (in Bytes).
- **Class Number (8 Bit)**: Die Klassennummer des Objekts (der „Typ“ des Objekts).
- **Class Type (8 Bit)**: Die Subklassennummer des Objekts.
- **Contents (Length – 4 Bytes)**: Die Daten des Objekts.

In der ersten Version des RSVP-Protokolls wurden folgende Hauptklassen von RSVP-Objekten definiert:

- **SESSION**

Das SESSION-Objekt definiert, zur welcher RSVP-Session (3-Tupel: (Zieladresse, ProtokollID, Zielportnummer)) eine RSVP-Nachricht gehört. Weiter beinhaltet das SESSION-Objekt in der PATH-Nachricht das so genannte E_Police-Flag, das vom Sender gesetzt wird, falls er das so genannte Traffic-Policing nicht unterstützt, und das vom ersten RSVP-Router, der Traffic-Policing unterstützt, zurückgesetzt wird.

- **RSVP_HOP**

Das RSVP_HOP-Objekt trägt die Information, durch welche Netzwerkschnittstelle (IP-Adresse und logische Schnittstellen-Nummer) die RSVP-Nachricht versandt wurde. Dieses Objekt wird benutzt, um den Pfad von RSVP-Nachrichten in umgekehrter Richtung verfolgen zu können.

- **INTEGRITY**

Wenn es in einer RSVP-Nachricht enthalten ist, ermöglicht das INTEGRITY-Objekt die Authentifizierung des RSVP-Routers, der die Nachricht versandt hat. Das Integrity-Objekt beinhaltet die ID (Identifikationsnummer) eines geheimen Schlüssels, die Sequenznummer der RSVP-Nachricht[2] und einen Hash-MAC[3] der Nachricht mit dem geheimen Schlüssel. Damit wird so genannten Replay- und Man-in-the-Middle-Attacken[4] vorgebeugt. Das Format des INTEGRITY-Objekts ist in [12] definiert.

- **TIME_VALUES**

Das TIME_VALUES Objekt definiert die Auffrischungszeit einer RSVP-Nachricht. So kann jeder RSVP-Router seinen Nachbarn den Zeitraum angeben, innerhalb dessen eine Auffrischung des Soft-States erfolgen sollte. Falls diese Zeitspanne überschritten wird,[5] darf der Soft-State entfernt werden.

- **SCOPE**

Das SCOPE-Objekt beinhaltet eine Liste der IP-Adressen, die für das Routing solcher RSVP-Nachrichten benutzt wird, die sich auf Reservierungen mit Wildcard-Filter beziehen. Auf diese Weise kann die Selbsterneuerung des Zustands durch das Kreisen der RSVP-Nachrichten (Endlosschleife) verhindert werden.

- **STYLE**

Das Style-Objekt definiert den Reservierungsstil. Der Reservierungsstil besteht aus zwei Komponenten: Der Zusammenfassungskontrolle (mit einer eigenen Reservierung für jeden Sender oder aber einer gemeinsamen Reservierung für alle Sender zusammen) und der Auswahl der Sender (bei der jeder Sender entweder einzeln benannt wird oder aber alle Sender aus einer Session zusammengefasst werden)[6].

---

2 Die Sequenznummern der RSVP-Nachrichten müssen nicht lückenlos gedeckt werden, aber sie müssen fortlaufend sein.

3 Ein Hash-MAC (Hashed Message Authentication Code) ist ein Verfahren zur Authentifizierung von Nachrichten auf der Basis einer kryptografischen Prüfsumme und eines geheimen Schlüssels. Für die genaue Definition vgl. [11].

4 Bei einer so genannter Replay-Attacke benutzt der Angreifer bei einer früheren Gelegenheit gewonnene Daten (typischerweise den Authentifizierungscode), um sich illegal Zugriff zu verschaffen. Bei einer so genannten Man-in-the-Middle-Attacke versucht eine dazu nicht berechtigte Person, Zugriff auf den Kommunikationskanal zu erlangen, ohne dass die eigentlichen Kommunikationspartner feststellen können, mit wem sie in Wirklichkeit Daten austauschen.

5 Um zu verhindern, dass die Zeitspanne infolge eines verlorengegangenen Paketes überschritten wird, wird dabei zusätzlicher Spielraum zugelassen, indem die eigentlich vorgesehene Zeitspanne mit einem Faktor – üblicherweise 3 – multipliziert wird.

6 Vgl. hierzu Seite 6 der vorliegenden Arbeit.

- **FLOWSPEC**

Das FLOWSPEC-Objekt beinhaltet die Beschreibung der Reservierung, die jeweils gewünscht wird. Der Inhalt dieses Objekts ist für das RSVP-Protokoll transparent und wird unverändert an das so genannte Traffic-Control-Modul[7] des Routers weitergeleitet.

- **FILTER_SPEC**

Das FILTER_SPEC-Objekt dient der Auswahl der Sender, auf die sich die Netzwerkressourcenreservierung bezieht. In der Version 1 von RSVP kann ein Sender anhand der IP-Adresse und der Source-Portnummer (in IPV4 und IPV6) oder anhand der IP-Adresse und des Flow-Labels (nur in IPV6) ausgewählt werden.

- **SENDER_TEMPLATE**

Das SENDER_TEMPLATE-Objekt beinhaltet die Senderadresse und weitere Informationen für die Identifikation des Senders (Source-Portnummer oder Flow Label).

- **SENDER_TSPEC**

Das SENDER_TSPEC definiert die Traffic-Spezifikation, also die Eigenschaften des Datenstroms der Daten, die durch den Sender erzeugt werden, und stellt die oberere Grenze für die Reservierung von Netzwerkressourcen für den Datenstrom dieses Senders dar.

- **ADSPEC**

Das ADSPEC-Objekt kann vom Sender in die PATH-Nachricht eingefügt werden und wird von jedem RSVP-Router auf dem Pfad zum Empfänger bearbeitet. Es kann Informationen über die maximale verfügbare Bandbreite auf dem Pfad enthalten (jeder RSVP-Router kann diesen Wert nach unten korrigieren), ferner über die MTU[8] sowie darüber, ob alle Router auf dem Pfad das RSVP-Protokoll unterstützen, und anderes mehr.

- **RESV_CONFIRM**

Das RESV_CONFIRM-Objekt beinhaltet die IP-Adresse des RSVP-Empfängers, der bei Erfolg der Reservierung benachrichtigt werden soll.

- **POLICY_DATA**

Die Policy-Data-Objekte beinhalten alle Informationen, die sich auf die Authentifizierung von RSVP-Nachrichten beziehen. Da sie für die vorliegende Diplomarbeit von zentraler Bedeutung sind, sollen sie an dieser Stelle etwas eingehender beschrieben werden.

Der Inhalt von POLICY_DATA-Objekten wird in RSVP nicht definiert. Die eigentliche Definition von POLICY-DATA-Objekten findet sich in [13] und hat die folgende Form:

7 Ein Traffic-Control-Modul ist eine Komponente des Routers, deren Aufgabe die Einstellung und Einhaltungvon QoS ist.

8 MTU, eine Abkürzung für „Maximal Transmit Unit“, bezeichnet die maximale Grösse eines Datenpakets, das ohne Segmentierung durch eine Netzwerkverbindung übertragen werden kann.

Data-Offset	Reserved
Option-List	
Policy-Element-List	

Die Bedeutung der Felder ist:

- **Data-Offset (16 Bit)**: Der relative Abstand (in Bytes) des ersten Eintrags aus der Policy-Element-Liste zum Anfang des Objektheaders (also nicht des Objektinhalts).
- **Option-List (Variabel)**: Eine Liste von RSVP-Objekten, deren Semantik gegenüber ihrer jeweiligen Semantik innerhalb der RSVP-Nachricht verändert ist.[9]
- **Policy-Element-List (Variabel)**: Eine Liste von Elementen des Typs POLICY_ELEMENT.

Jedes POLICY_ELEMENT hat folgendes Format:

Length	P-Type
Data	

Die Bedeutung der Felder ist:

- **Length (16 Bit)**: Die Länge des POLICY_ELEMENTs in Bytes, inklusive Header.
- **P-Type (16 Bit)**: Der Typ des POLICY_ELEMENTs. Die Zahlen von 0 bis 49151 sind für Standard-Bestandteile reserviert. Die Zahlen von 49152 bis 53247 sind für Firmen registriert (eine pro Firma) und werden von IANA [14] vergeben. Die Zahlen von 53248 bis 65535 sind für den privaten Gebrauch bestimmt und werden nicht von IANA verwaltet.
- **Data (Length-8 Bytes)**: Die Daten des POLICY_ELEMENTs. Der Inhalt ist für das RSVP-Protokoll transparent und wird von den PEPs und PDPs interpretiert.

Version 1 des RSVP-Protokolls definiert die folgenden sieben Typen von Nachrichten:

- **PATH**

PATH-Nachrichten werden periodisch vom Sender in Richtung Empfänger versandt. Eine PATH-Nachricht wird vom RSVP-Router empfangen und benutzt, um den Path-Zustand für ein RSVP-Session aufzubauen. Die RSVP-Router selbst generieren dann aus den Path-Zuständen die PATH-Nachrichten, die in Richtung Empfänger versandt werden.

Eine PATH-Nachricht beinhaltet ein SESSION-Objekt (IP-Adresse, Protokollnummer und die Portnummer des Empfängers), die Information über den Sender (IP-Adresse und entweder Portnummer oder Flow Label), die Informationen über den Datenverkehr, der generiert wird (maximale benötigte Bandbreite etc.), die Adresse des Versenders der PATH-Nachricht (sie wird

9 Zur Semantik von RSVP-Objekten als Optionen innerhalb von POLICY_DATA-Objekten vgl. [13], Seite 4.

benötigt, um den Pfad der Nachricht zurückverfolgen zu können), Policy-Daten für die Authentifizierung und optional ferner auch Informationen über die verfügbaren Ressourcen auf dem Pfad zum Sender.

- **RESV**

RESV-Nachrichten werden periodisch von den Empfängern versandt, um die Reservierung der Ressourcen aufrecht zu erhalten. Der Grund hierfür ist der Soft-State von RSVP, der ohne Auffrischung nach Ablauf einer bestimmten Frist automatisch verfällt. Eine RESV-Nachricht beinhaltet die Information über die gewünschten Netzwerkressourcen wie Bandbreite, Verzögerung, etc.; die Auswahl der Server, für die Reservierungen vorgenommen werden; den Stil der Reservierung; ein SCOPE-Objekt, falls eine Wildcard-Auswahl der Server benutzt wird; die Policy-Daten für die Authentifizierung sowie optional die Adresse des Empfängers, der die Bestätigung der Reservierung erhalten möchte.

Im Fall von Multicast wird in denjenigen RSVP-Routern, wo sich der Multicast-Baum verzweigt, das so genannte Merging, also die Zusammenfassung der RESV-Nachrichten, vorgenommen. Mehrere RESV-Nachrichten können nur dann zusammengefasst werden, wenn sie denselben Reservierungsstil benutzen. Die dabei entstehende Nachricht beinhaltet zugleich die kleinste obere Schranke aller Reservierungen aus den RESV-Nachrichten, die zusammengefasst wurden. Die so erzeugte RESV-Nachricht wird dann in Richtung Sender versandt.

- **PATHTEAR**

Die PATHTEAR-Nachricht wird von einem RSVP-Router zum Empfänger versandt, um den PATH-Zustand für einen Sender zu löschen. Die PATHTEAR-Nachricht kann entweder vom Sender selbst (beim Herunterfahren des Sender-Prozesses) oder von einem beliebigen RSVP-Router generiert werden. Letzteres ist der Fall, wenn ein PATH-Zustand abgelaufen ist. Beim Empfang einer PATHTEAR-Nachricht muss ein RSVP-Router den entsprechenden PATH-Zustand löschen.

- **RESVTEAR**

Die RESVTEAR-Nachricht wird von einem RSVP-Router in Richtung Sender versandt, um die bestehende Reservierung von Netzwerkressourcen zu löschen. Die RESVTEAR-Nachricht wird entweder von einem Empfänger generiert, der die Reservierung nicht mehr braucht, oder aber von einem RSVP-Router erzeugt; letzteres erfolgt, wenn die Frist für die Auffrischung des Reservierungszustandes überschritten wird. Beim Empfang einer RESVTEAR-Nachricht muss jeder RSVP-Router den Reservierungszustand neu berechnen und in Abhängigkeit vom Ergebnis entweder durch eine neue RESV-Nachricht in Richtung Sender die Änderung der Reservierung oder aber per RESVTEAR deren Löschung signalisieren.

- **PATHERR**

Falls bei der Verarbeitung einer PATH-Nachricht ein Fehler aufgetreten ist, wird vom RSVP-Router eine PATHERR-Nachricht versandt, die den Grund für das Scheitern der Verarbeitung enthält.

- **RESVERR**

Falls bei der Verarbeitung einer RESV-Nachricht oder bei der Reservierung von Netzwerk-Ressourcen in einem RSVP-Router ein Fehler aufgetreten ist, wird eine RESVERR-Nachricht an den RSVP-Router, der die RESV-Nachricht versandt hat, zurückgeschickt. Der Inhalt der Nachricht ist der Grund für das Scheitern der Verarbeitung beziehungsweise der Reservierung der Netzwerkressourcen.

- **CONFIRM**

Falls in einer RESV-Nachricht mit dem Objekt RESV_CONFIRM eine IP-Adresse übertragen wurde, wird bei erfolgreicher Reservierung der Netzwerk-Ressourcen eine CONFIRM-Nachricht versandt. Diese Nachricht beinhaltet die Informationen über die reservierten Ressourcen.

Der Ablauf der RSVP-Kommunikation ist folgender:

- **Erstellen des PATH-Zustands**

Zu Beginn versenden die Sender PATH-Nachrichten. Eine PATH-Nachricht beinhaltet das SESSION-Objekt, die IP-Adresse und die logische Netzwerkschnittstellennummer des Absenders (RSVP_HOP-Objekt), die Erneuerungsperiode für den PATH-Zustand (TIME_VALUES-Objekt), die Informationen über den Sender (SENDER_TEMPLATE- und SENDER_TSPEC-Objekt) sowie eventuell das ADSPEC-Objekt. Letzteres ist ein Objekt, das benutzt wird, um das „One Pass with Advertisement"-Protokoll verwenden zu können: das Objekt akkumuliert die Informationen über den Pfad und ermöglicht es dem Empfänger, die Eigenschaften des Datenpfades zu erfahren. Die Zieladresse der PATH-Nachrichten ist die Adresse der RSVP-Session. Jeder Router, der das RSVP-Protokoll unterstützt, empfängt die PATH-Nachrichten und generiert aus ihnen die PATH-Zustände. Die Hauptaufgabe eines PATH-Zustands ist es, die IP-Adresse des Routers zu speichern, an den die Reservierungsnachrichten für die entsprechenden Server versandt werden sollen. Auf diese Weise speichert jeder Router den nächsten „Sprung" für die Reservierungsnachrichten. Aus den PATH-Zuständen werden dann weitere PATH-Nachrichten generiert, um auf den jeweils nächsten Routern auf dem Pfad zum Empfänger den PATH-Zustand aufzubauen. Am Ende erreichen die PATH-Nachrichten die Empfänger.

- **Reservierung**

Nach dem Empfang von RESV-Nachrichten kann der Empfänger Netzwerkressourcen reservieren. Dabei kann der Empfänger den Reservierungsstil wählen. Um eine Reservierung vorzunehmen, versendet der Empfänger eine RESV-Nachricht an den Router, von dem die entsprechende PATH-Nachricht gekommen ist. Die RESV-Nachricht beinhaltet Folgendes:

- das SESSION-Objekt, die Adresse und die logische Nummer der Netzwerkschnittstelle, durch das die PATH-Nachricht versandt wurde (RSVP_HOP-Objekt);
- die Erneuerungsperiode für den RESV-Zustand (TIME_VALUES-Objekt);
- ein RESV_CONFIRM-Objekt (falls der Empfänger benachrichtigt werden möchte, ob die Reservierung gelungen ist)
- ein SCOPE-Objekt mit den Adressen der Sender, in deren Richtung noch keine RESV-Nachricht versandt wurde. Dieses Objekt wird nur beim WF-Reservierungsstil benötigt, um das Entstehen von Schleifen beim Versand von RESV-Nachrichten zu vermeiden;
- eine Deklaration des Reservierungsstils (STYLE-Objekt) und
- eine Liste von Reservierungen, deren Inhalt vom Reservierungsstil abhängig ist.

Beim Empfang der ersten RESV-Nachricht muss der Router die entsprechenden Ressourcen reservieren (das heisst, QoS für die Datenströme einstellen), und falls die Reservierung erfolgreich war, wird ein RESV-Zustand (State) erstellt. Aus dem RESV-Zustand werden dann RESV-Nachrichten erzeugt, die auf dem umgekehrten Pfad der PATH-Nachrichten an die Router versandt werden. Bei Multicast-Datenübertragung werden die Reservierungen in den Routern, wo sich der Multicast-Baum verzweigt, zusammengefasst, so dass die resultierende RESV-Nachricht die kleinste obere Schranke aller Reservierungen aus den Teilbäumen darstellt. Dabei dürfen nur Reservierungen des gleichen Stils zusammengefasst werden.

- **Freigeben der Reservierung**

Wenn ein Empfänger eine Reservierung nicht mehr benötigt oder wenn der RESV-Zustand in einem Router wegen Überschreitens der maximal zulässigen Erneuerungszeit des Reservierungszustands abgelaufen ist, wird eine RESVTEAR-Nachricht in Richtung Sender versandt, das heisst an den nächsten Router auf dem Pfad der PATH-Nachrichten in umgekehrter Richtung. Diese Nachricht bewirkt, dass der RESV-Zustand gelöscht wird. Wenn ein RESV-Zustand gelöscht wird, kann das eine der folgenden beiden Konsequenzen haben:

1. Falls kein RESV-Zustand für die RSVP-Session mehr vorhanden ist, wird eine RESVTEAR-Nachricht in Richtung Sender versandt.
2. Falls für die RSVP-Session weitere RESV-Zustände existieren, wird eine neue Zusammenfassung der Reservierungen berechnet (die kleinste obere Schranke aller übriggebliebenen Reservierungen), die QoS im Router wird angepasst und die neue Reservierung wird als RESV-Nachricht in Richtung Sender versandt.

- **Abmelden des Senders**

Falls ein Sender aufhört zu senden oder ein PATH-Zustand (State) ausläuft, wird eine PATHTEAR-Nachricht in Richtung Empfänger versandt. Beim Empfang einer PATHTEAR-Nachricht wird der PATH-Zustand gelöscht, was in Abhängigkeit vom Reservierungsstil Auswirkungen auf die bestehenden RESV-Zustände haben kann.

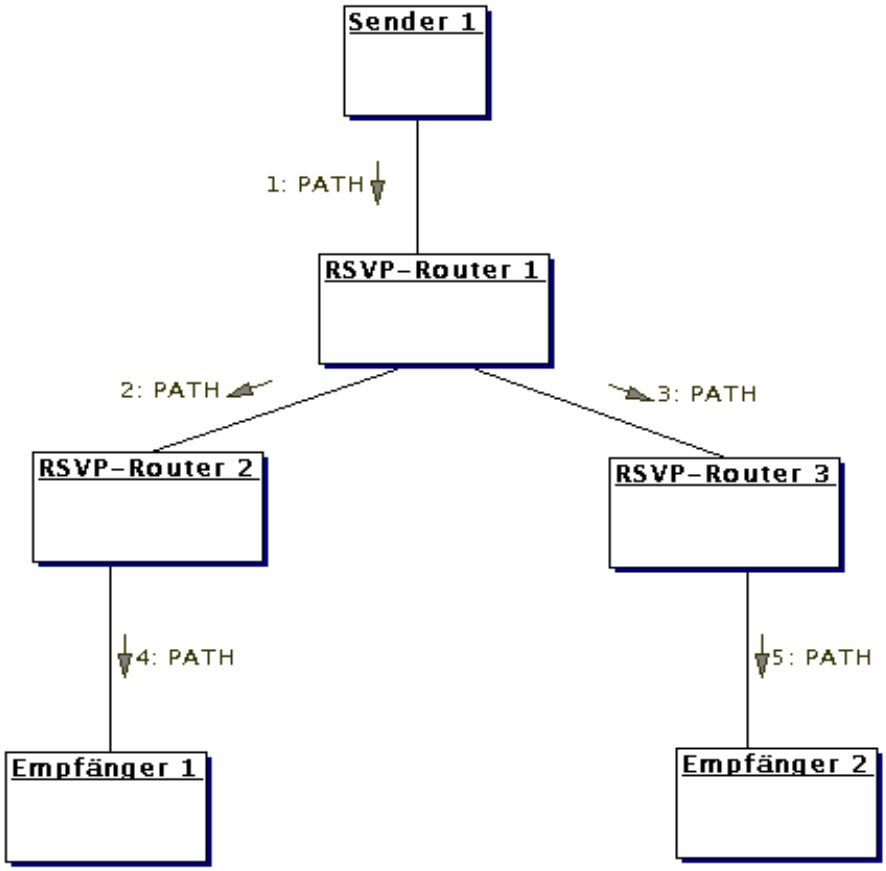
- **Signalisierung von Fehlern**

Falls bei der Verarbeitung von PATH- oder RESV-Nachrichten Fehler auftreten, werden entsprechende PATHERR- beziehungsweise RESVERR-Nachrichten an die Fehlerquelle, also den Absender der PATH- oder RESV-Nachricht, versandt. Die RESVERR- und PATHERR-Nachrichten beinhalten die Beschreibung des Fehlers sowie ausreichend Informationen, um den Fehler einer PATH- beziehungsweise RESV-Nachricht zuordnen zu können.

Im folgenden Beispiel werden Daten von einem Sender (Sender 1) durch drei Router (RSVP-Router 1, RSVP-Router 2, RSVP-Router 3) per Multicast an zwei Empfänger (Empfänger 1, Empfänger 2) versandt. Dabei wird das RSVP-Protokoll benutzt, um die Bandbreite für die Übertragung zu reservieren. Der Empfänger 1 reserviert 1 Mbit und der Empfänger 2 reserviert 2 Mbit für die Übertragung. Beide benutzen den FF-Reservierungsstil.

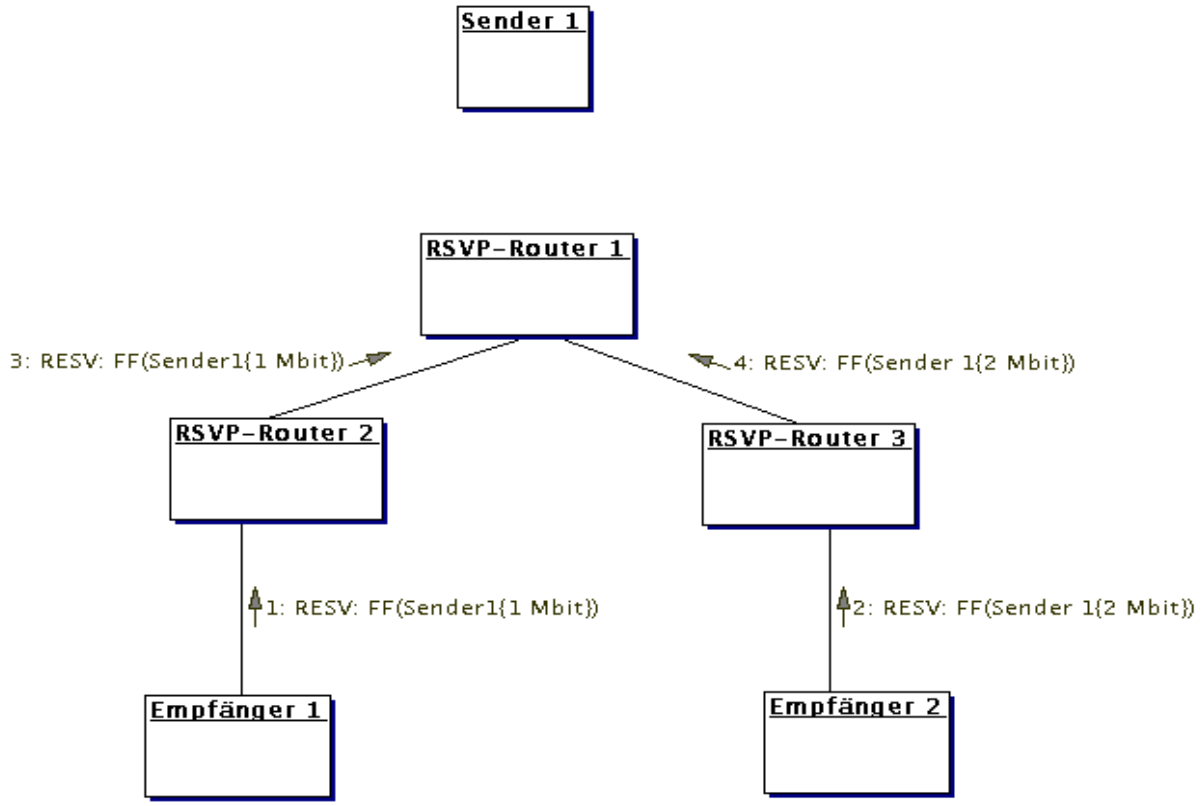
- **Versenden der PATH-Nachricht**

Zu Beginn werden die PATH-Nachrichten vom Sender in Richtung Empfänger versandt.



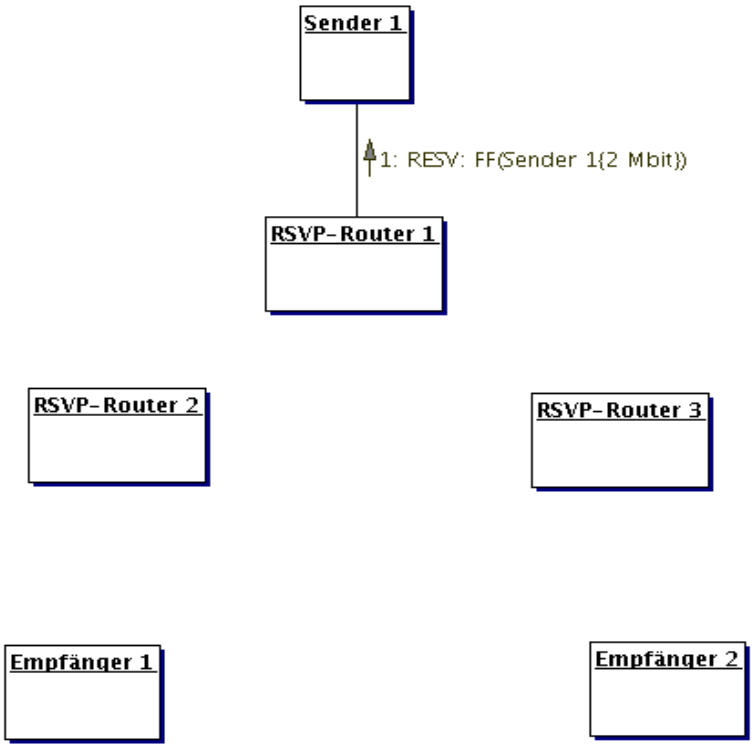
- **Versenden von RESV-Nachrichten**

Die Empfänger senden die RESV-Nachrichten mit dem entsprechenden Reservierungsstil (FF) und Flowspec (1 beziehungsweise 2 Mbit):



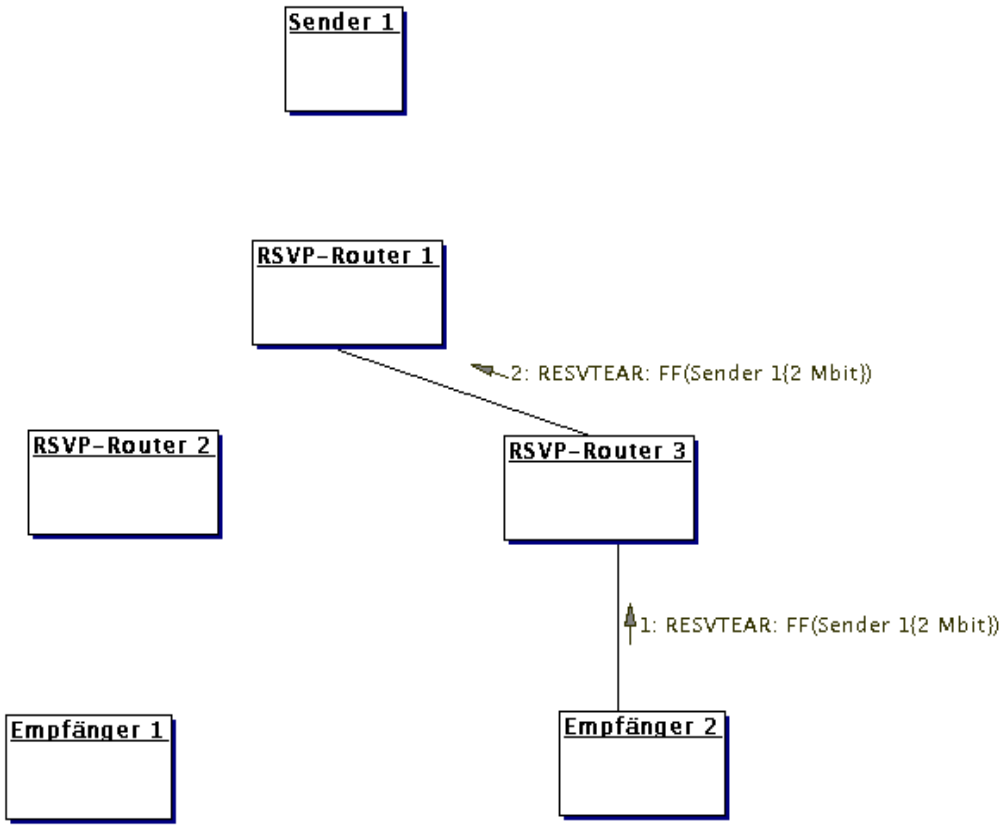
- **Zusammenfassung der RESV-Nachrichten**

Im RSVP-Router 1 werden die RESV-Nachrichten zusammengefasst und als eine RESV-Nachricht mit der Reservierung, die die kleinste obere Schranke für die Reservierungen aus den beiden vorherigen RESV-Nachrichten darstellt, an den Sender 1 versandt.



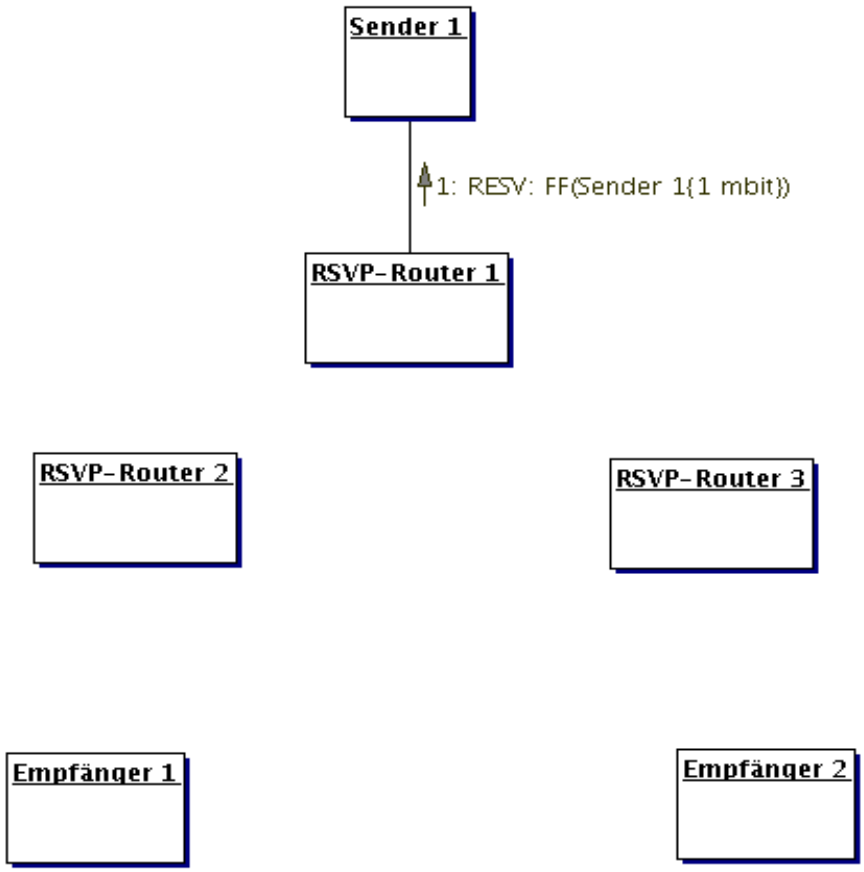
- **Löschung der Reservierung**

Nachdem der Empfänger 2 die Reservierung nicht mehr benötigt, schickt er eine RESVTEAR-Nachricht an den RSVP-Router 3. Der RSVP-Router 3 hatte auf der Netzwerkschnittstelle nur die Reservierungen für den Empfänger 2 und hat deswegen nach dem Empfang der RESVTEAR-Nachricht keine Reservierungen mehr. Der RSVP-Router 3 sendet dann eine RESVTEAR-Nachricht an den RSVP-Router 1, um zu signalisieren, dass in Richtung RSVP-Router 3 keine Reservierung mehr nötig ist.



- **Aktualisierung der Reservierung**

Nach dem Empfang der RESVTEAR-Nachricht stellt der RSVP-Router 1 fest, dass noch eine Reservierung von RSVP-Router 2 für den Sender 1 existiert. Die neue Reservierung unterscheidet sich aber von der zusammengefassten von RSVP-Router 1 und RSVP-Router 3 (sie ist kleiner). Um das zu signalisieren, versendet der RSVP-Router 1 eine neue PATH-Nachricht mit der kleineren Reservierung (1 Mbit) an den Sender 1.



2.4 COPS

Mit dem RSVP-Protokoll wird das Problem gelöst, wie QoS in allen Routern, die an einer Datenübertragung teilnehmen, eingestellt werden kann. Immer noch offen bleibt dabei aber die Frage, ob eine Reservierung authorisiert ist, also ob ein Router eine RESV-Nachricht überhaupt akzeptieren soll. Diese Entscheidung kann für einen Internet-Service-Provider (ISP) durchaus wichtig sein, weil die Bandbreite, die auf ausgehenden und internen Datenpfaden reserviert werden kann, begrenzt ist. Es ist naheliegend, dass die Entscheidung, ob eine Reservierung stattfinden kann, im Router selbst implementiert wird. Leider stellen sich folgende Probleme bei der Implementierung der Entscheidung in Routern:

- Ein Router hat nur sehr begrenzte CPU- und RAM-Ressourcen zur Verfügung.

- Bei jeder Änderung des Authentifizierungsschemas müssen die Implementierungen in Routern angepasst werden, was sehr umständlich werden könnte (vor allem, wenn man homogene ISP-Netzwerke verwendet).
- Die Distribution der Informationen für die eventuelle Authentifizierung ist mit diesem Ansatz immer noch nicht gelöst.

Um diese Probleme zu umgehen, hat es sich angeboten, die Entscheidung an einen zentralen Server zu verlagern. Um die Kommunikation zwischen den Routern und dem zentralen entscheidungstreffenden Server zu standardisieren, wurde das COPS-Protokoll [2] definiert.

COPS ist die Abkürzung für „Common Open Policy Service“. Im COPS-Protokoll wird zwischen dem Client (PEP – Policy Enforcement Point) und dem Server (PDP – Policy Decision Point) unterscheiden. Die Aufgabe des COPS-Protokolls ist die Übertragung der Anfragen (Requests) vom PEP zum PDP und der Entscheidungen (Decisions) vom PDP zum PEP (dem COPS-Server). Das COPS-Protkoll kann nicht nur für die Auslagerung von Entscheidungen auf einen zentralen Server benutzt werden, sondern es kann auch verwendet werden, um die Konfiguration von Routern durch einen zentralen Server zu ermöglichen. Das COPS-Protkoll benutzt das TCP-Protokoll für die Datenübertragung; dabei ist auf der Serverseite die Portnummer 3288 für das COPS-Protokoll reserviert. Der Verbindungsaufbau erfolgt immer von PEP zu PDP. Das COPS-Protokoll hat folgende Eigenschaften:

- **Transparenz**

Für das COPS-Protokoll sind die Daten, die in einer Anfrage (Request) und in einer Entscheidung (Decision) übertragen werden, völlig transparent.

- **Sicherheit**

Das COPS-Protokoll ermöglicht den Einsatz der bidirektionalen Authentifizierung, die in der Lage ist, sowohl den Server als auch den Client zu identifizieren, und sicherstellen kann, dass die übertragenen Daten unterwegs nicht verändert wurden.

- **Zustandbehaftet (Stateful)**

Eine Anfrage (Request) vom Client erzeugt auf dem Server einen Zustand (State), der so lange auf dem Server besteht, bis der Client ihn explizit löscht. Der durch die Anfrage erzeugte Zustand ermöglicht es, dass der Server asynchron mehrere Entscheidungen an den Client versenden kann (zum Beispiel, um die Konfiguration eines Routers zu verändern).

- **Überprüfung der Verbindung**

Um die TCP-Verbindung zu überprüfen, werden vom PEP periodisch so genannte Keep-Alive-(KA)-Nachrichten versandt. Diese Nachrichten werden vom PDP beim Empfang sofort an den PEP zurück verschickt. Falls die KA-Nachricht nicht wieder zum PEP zurückkommt beziehungsweise falls der PDP nach Ablauf der festgelegten Zeit keine KA-Nachricht empfangen hat, wird die Verbindung für ungültig erklärt. Danach muss der PEP immer wieder versuchen, eine neue Verbindung zum PDP aufzubauen.

Die Kommunikation zwischen PEP und PDP erfolgt in Form von Nachrichten.

- **COPS-Nachrichten**

Eine COPS-Nachricht hat folgendes Format:

Version	Flags	Operation-Code	Client-Type
Message-Length			
Objects			

Die Bedeutung der Felder ist:

- **Version (4 Bit)** – Die Version des COPS-Protokolls. Zur Zeit existiert nur Version 1.
- **Flags (4 Bit)** – Die Flags. In der derzeitigen Version existiert nur ein Flag: 1 = Solicited Message. Mit dem Flag werden diejenigen Nachrichten markiert, die direkte Antworten auf eine vorherige Nachricht sind.
- **Operation-Code (8 Bit)** – Der „Typ“ der Nachricht.
- **Client-Type (16 Bit)** – Die Clientnummer.
- **Message-Length (32 Bit)** – Die Länge der ganzen Nachricht (inklusive Header) in Bytes.
- **Objects** – Eine Liste von COPS-Objekten, die die Nachricht beschreiben.

COPS definiert 10 Typen von Nachrichten:

1. **Request** (**REQ**) [PEP -> PDP]: Die Anfrage von PEP an PDP. Die Anfrage hat einen Handle (Client-Handle) und weitere Informationen, die für den PDP relevant sind, um die jeweils notwendige Entscheidung zu treffen. Wichtig ist, dass der Anfrage-Zustand (Request-State) mit der REQ-Nachricht etabliert wird und erhalten bleibt, bis der PEP den Zustand mittels DRQ explizit löscht oder die TCP-Verbindung zum PEP unterbrochen wird. Dabei wird der Anfrage-Zustand durch den Client-Handle eindeutig referenziert. Auch jede Änderung im Anfrage-Zustand wird dem PDP mittels REQ-Nachricht signalisiert.
2. **Decision** (**DEC**) [PEP <- PDP]: Die Antwort von PDP an PEP. Diese Nachricht beinhaltet die Antwort sowie eventuell zusätzliche Daten, die zur Antwort gehören (zum Beispiel POLICY_DATA für ein RSVP-Paket). Der Client-Handle aus der Anfrage (Request) wird mitgesendet, so dass der PEP in der Lage ist, die Antwort einer Anfrage (REQ) zuordnen zu können. Es ist möglich, dass für eine REQ-Nachricht mehrere DEC-Nachrichten vom PDP verschickt werden, um dem PEP zu signalisieren, dass der PDP seine Entscheidung geändert hat. Dabei muss die erste DEC-Nachricht das Solicited-Flag gesetzt haben, alle andere müssen ohne gesetztes Solicited-Flag versandt werden.
3. **Report State** (**RPT**) [PEP -> PDP]: Mit dieser Nachricht signalisiert der PEP Erfolg oder Misserfolg bei der Umsetzung einer PDP-Entscheidung. Auch hier wird der Anfrage-Zustand durch den Client-Handle referenziert.
4. **Delete Request State** (**DRQ**) [PEP -> PDP]: Die Delete-Request-Nachricht signalisiert dem PDP, dass ein Anfrage-Zustand nicht mehr existiert und damit keine weiteren Entscheidungen mehr für ihn notwendig sind.
5. **Synchronize State Req** (**SSQ**) [PEP <- PDP]: Diese Nachricht signalisiert dem PEP, dass er entweder den Zustand einer Anfrage (falls ein Client-Handle in der Nachricht enthalten ist) oder aller Anfragen (falls kein Client-Handle vorhanden ist) nochmals versenden soll. Falls der PEP den Anfrage-Zustand nicht mehr führt, muss er mit einer DRQ-Nachricht antworten.

6. **Client-Open** (**OPN**) [PEP –> PDP]: Eine Client-Open-Nachricht signalisiert dem PDP, dass der PEP einen neuen Client-Type benutzen will. Diese Nachricht muss vom PEP versandt werden, bevor irgendeine andere Nachricht für diesen Client-Type versandt wird. Falls der PDP den Client-Type unterstützt, antwortet er mit einer Client-Accept(CAT)-Nachricht; anderenfalls muss er eine Client-Close(CC)-Nachricht an den PEP zurücksenden. Ausserdem wird bei diesem Nachrichtentyp der Client-Type 0 benutzt, um dem PDP zu signalisieren, dass der PEP die Authentifizierung des COPS-Protokolls benutzen möchte, und um die initiale Sequenz-Nummer für die Nachrichten vom PEP zu versenden.
7. **Client-Accept** (**CAT**) [PEP <– PDP]: Eine Client-Accept-Nachricht wird vom PDP an den PEP verschickt, um dem PEP mitzuteilen, dass der gewünschte Client-Type unterstützt wird und dass der PEP Nachrichten für diesen Client-Type versenden darf. Eine CAT-Nachricht mit dem Client-Type 0 wird als positive Antwort auf die OPN-Nachricht mit dem Client-Type 0 benutzt, um die initiale Sequenz-Nummer für die COPS-Authentifizierung vom PDP an den PEP zu versenden.
8. **Client-Close** (**CC**) [PEP <–> PDP]: Die Client-Close-(CC)-Nachricht wird benutzt, um entweder dem PDP zu signalisieren, dass der PEP einen Client-Type nicht mehr benötigt, oder um dem PEP zu signalisieren, dass der PDP einen Client-Type nicht unterstützt. Falls der PDP eine CC-Nachricht an einen PEP versendet, kann er noch zusätzlich eine IP-Adresse und eine TCP-Socket-Portnummer von einem alternativen COPS-Server angeben, der den gewünschten Client-Type unterstützt. Dieses Verfahren kann verwendet werden, um beim Herunterfahren des PDP die schon verbundenen PEPs an einen alternativen PDP umzuleiten. Darüber hinaus wird die CC-Nachricht mit dem Client-Type 0 benutzt, um dem PEP das Fehlschlagen der Authentifizierung zu signalisieren.
9. **Keep-Alive** (**KA**) [PEP <–> PDP]: Keep-Alive-Nachrichten werden periodisch vom PEP an den PDP und vom PDP sofort wieder zurück an den PEP versandt, um zu überprüfen, ob die TCP-Verbindung immer noch offen ist.
10. **Synchronize-State-Complete** (**SSC**) [PEP –> PDP]: Mit dieser Nachricht signalisiert der PEP dem PDP, dass die Synchronisation der Anfrage-Zustände (Request-States) abgeschlossen ist.

- **COPS-Objekte**

Jedes COPS-Objekt hat folgendes Format:

Length	Class-Number	Class-Type
Contents		

Dabei haben die Felder die folgende Bedeutung:

- **Length (16 Bit)**: die Länge des gesamten COPS-Objekts, inklusive Header.
- **ClassNumber (8 Bit)**: die Klasse des Objekts.
- **ClassType (8 Bit)**: der Sub-Type der Objektklasse.
- **Contents**: Daten des Objekts, wobei der Inhalt von Class-Number und Class-Type abhängt.

Im COPS sind folgende Klassen (Class-Number) von Objekten definiert:

1. **Handle**: Das Handle-Objekt definiert eine eindeutige Referenz für einen Anfrage-Zustand. Handle-Objekte werden in den folgenden Nachrichten benutzt: REQ, DEC, RPT, DRQ, SSQ, SSC.
2. **Context**: Das Context-Objekt besteht aus zwei Teilen: den Request-Flags (eingehende Nachricht, Zuweisung der Ressourcen, ausgehende Nachricht oder Konfigurationsanfrage) und dem Message-Type, der vom Client-Type abhängig ist. Der Context definiert in REQ- und DEC-Nachrichten, auf welche Aktionen sich die Anfrage beziehungsweise die Entscheidung bezieht.
3. **InInterface**: Das InInterface-Objekt in der REQ-Nachricht beschreibt, an welcher Netzwerkschnittstelle eine Nachricht empfangen wurde (es enthält also die IP-Adresse und logische Nummer der eingehenden Schnittstelle).
4. **OutInterface**: Das OutInterface-Objekt in der REQ-Nachricht beschreibt, an welcher Netzwerkschnittstelle eine Nachricht den Router verlassen wird (es enthält also die IP-Adresse und logische Nummer der ausgehenden Schnittstelle).
5. **ReasonCode**: Das ReasonCode-Objekt erscheint nur in DRQ-Nachrichten und beschreibt den Grund, weswegen der Anfrage-Zustand gelöscht wird. Der Reason-Code besteht aus zwei Teilen: der Reason-Code-Nummer (deren Werte vordefiniert sind) und der Reason-Sub-Code-Nummer, die abhängig vom Client-Type weitere Informationen über die Löschungsgründe beinhaltet.
6. **Decision**: Das Decision-Objekt erscheint nur in den DEC-Nachrichten. Decision-Objekte deklarieren die Entscheidung des PDP und können auch noch weitere Informationen transportieren, deren Format vom Client-Type abhängt.
7. **LPDP Decision**: Das LPDP-Decision-Objekt wird in den REQ-Nachrichten übertragen und beschreibt die Entscheidungen, die vom PEP getroffen wurden, während er im lokalen Entscheidungsmodus war.
8. **Error**: Das Error-Objekt erscheint in den DEC-Objekten, wenn der PDP nicht in der Lage war, eine Entscheidung zu treffen, oder wenn ein Fehler aufgetreten ist. Das Error-Objekt erklärt den Grund für das Scheitern der Anfrage und besteht, ähnlich wie das Context- oder das ReasonCode-Objekt, aus einem vordefinierten Teil (Error-Code-Nummer) und einem vom Client-Type abhängigen Teil (Error-Sub-Code-Nummer).
9. **Client Specific Information**: Das Client-Specific-Information-Objekt beinhaltet Informationen, deren Inhalt nur für einen bestimmten Client-Type relevant ist. Dieses Objekt wird überall benutzt, wo zusätzliche Informationen transportiert werden müssen.
10. **Keep-Alive Timer**: Der Keep-Alive-Timer wird in den OPN-Nachrichten verwendet, um die maximale Zeit für das Versenden von Keep-Alive-Nachrichten zu definieren. Da jeder Client-Type OPN einen eigenen KA-Timer definieren kann, muss die COPS-Implementierung das Minimum von allen KA-Timern als den endgültigen KA-Timer verwenden.
11. **PEP Identification**: Das PEP-Identifikations-(PEPID)-Objekt wird in den OPN-Nachrichten übertragen und definiert den eindeutigen Namen (ID) des PEP innerhalb vom COPS-Server.
12. **Report Type**: Das Report-Type-Objekt in den RPT-Nachrichten spezifiziert, ob die Entscheidung des PDP im PEP umgesetzt werden konnte.
13. **PDP Redirect Address**: Das PDP-Redirect-Address-Objekt beinhaltet die TCP-Socketadresse (Internetadresse und Portnummer) eines alternativen COPS-Servers, der wahrscheinlich in der Lage ist, den gewünschten Client-Type zu unterstützen. Das PDP-Redirect-Address-Objekt

wird optional vom PDP in die CC-Nachricht eingebunden, um den Client auf einen anderen COPS-Server umzuleiten.

14. **Last PDP Address**: Das Last-PDP-Adress-Objekt wird in die OPN-Nachricht eingebunden, wenn der PEP die TCP-Verbindung zum ursprünglichen COPS-Server verloren hat. Es gibt dem PDP die Information, welcher COPS-Server der letzte war, der vom PEP benutzt wurde.
15. **Accounting Timer**: Das Accounting-Timer-Objekt kann vom PDP optional in die CAT-Nachricht eingefügt werden, um den minimalen zeitlichen Abstand zwischen Accounting-Updates zu definieren.
16. **Message Integrity**: Das Message-Integrity-Objekt kommt in allen COPS-Nachrichten als letztes Objekt vor, falls man die COPS-Authentifizierung benutzt. Das Message-Integrity-Objekt besteht aus der Sequenznummer der Nachricht, der Key-ID und dem Hash-MAC. Der Hash-MAC wird über die ganze Nachricht mit Ausnahme der Prüfsumme selbst berechnet, so dass auch der Header und die Sequenznummer des Message-Integrity-Objekts einbezogen werden. Dadurch werden Replay-Attacken verhindert: sämtliche Sequenznummern der COPS-Nachrichten müssen aufeinander folgen. Somit unterscheidet sich die Prüfsumme auch für gleiche Nachrichten.

Im Folgenden wird der Ablauf der Datenkommunikation im COPS-Protokoll kurz dargestellt.

- **Aufbau der TCP-Verbindung**

Zu Beginn der Kommunikation muss der PEP eine TCP-Verbindung mit dem PDP herstellen. Den Verlauf der Kommunikation nach der Herstellung der TCP-Verbindung kann man in die nachfolgend dargestellten Phasen aufteilen.

- **Authentifizierungshandshake (optional)**

Falls der PEP eine Authentifizierung initiieren möchte, muss er als erste Nachricht eine OPN-Nachricht mit dem Client-Type 0 versenden. Der PDP muss entweder mit einer CAT-Nachricht (falls die Message-Integrity der OPN-Nachricht mit dem richtigen Schlüssel generiert wurde) oder mit einer CC-Nachricht (falls die Authentifizierung fehlgeschlagen ist) antworten. In beiden Fällen müssen die Nachrichten den Client-Type 0 haben.

- **Herstellen der Client-Type-Verbindung**

Der PEP muss für jeden Client-Type, der in der COPS-Verbindung benutzt werden soll, eine OPN-Nachricht versenden. Falls der PEP keine Authentifizierung initiiert hat, obwohl der PDP eine Authentifizierung verlangt, oder falls der PDP einen Client-Type nicht unterstützt, sendet der PDP eine CC-Nachricht mit entsprechendem Fehlercode zurück. Falls der PDP den Client-Type unterstützt, antwortet der PDP für jeden verlangten Client-Type mit einer CAT-Nachricht.

- **Zustandssynchronisation (optional)**

Abhängig vom Client-Type, der benutzt wird, kann der PDP vom PEP verlangen, für alle oder einzelne Anfrage-Zustände, die auf dem PEP existieren, REQ-Nachrichten zu versenden. Wenn nur ein einzelner Anfrage-Zustand nochmals übertragen werden soll, muss der PDP dem PEP eine SSQ-Nachricht mit dem Client-Handle zukommen lassen; wenn alle Anfrage-Zustände verlangt werden, wird kein Client-Handle beigefügt. Nach dem Empfang der SSQ -Nachricht muss der PEP die gewünschten Zustände mittels REQ-Nachrichten versenden; am Ende folgt eine SSC-Nachricht als Signal für das Ende der Zustandssynchronisation.

- **Bearbeiten von Anfragen (Requests)**

Beim Bearbeiten von Anfragen werden mittels REQ-Nachrichten vom PEP an den PDP neue Anfrage-Zustände hergestellt, durch DEC-Nachrichten vom PDP an den PEP Entscheidungen signalisiert und durch RPT-Nachrichten vom PEP an den PDP der Erfolg der Umsetzung von

Entscheidungen rapportiert. Falls ein Anfrage-Zustand nicht mehr existiert (beispielsweise wenn eine Netzwerkschnittstelle vom PEP abgeschaltet wird), kann der PEP ihn mittels DRQ-Nachricht löschen.

- **Schliessen der COPS-Verbindung**

Falls der PEP oder der PDP heruntergefahren werden, wird die COPS-Verbindung geschlossen. Davor werden in Abhängigkeit davon, wer eine Verbindung schliesst, CC-Nachrichten zum Kommunikationspartner versandt. Falls der PDP die Verbindung schliesst, kann er dem PEP in den CC-Nachrichten die Adresse und Portnummer eines alternativen COPS-Servers mitteilen. Falls die Verbindung unterbrochen wird (etwa durch einen Ausfall von Netzwerkkomponenten), versucht der PEP normalerweise immer wieder, eine neue TCP-Verbindung mit dem PDP aufzubauen. In der Zwischenzeit geht der PEP, falls möglich, auf den lokalen Entscheidungsmodus über.

Im Folgenden werden alle Phasen der COPS-Kommunikation an einem Beispiel erläutert. Dabei wird der Ablauf des Nachrichtenaustauschs dargestellt. In den Diagrammen werden der Auf- und Abbau der TCP-Verbindung selbst dabei nicht angezeigt, sonden nur die COPS-Nachrichten.

- **Authentifizierungshandshake**

Zu Beginn der Kommunikation signalisiert der PEP dem PDP seinen Wunsch, eine authentifizierte Kommunikation zu erstellen, indem er eine OPN-Nachricht mit dem Client-Type 0 versendet. Die Nachricht beinhaltet die initiale Sequenznummer für die Nachrichten, die vom PDP versandt werden, sowie ein Message-Integrity-Objekt, das mit dem geheimen Schlüssel generiert wurde. Der PDP überprüft die Prüfsumme der OPN-Nachricht und antwortet mit einer CAT-Nachricht, die die initiale Sequenznummer für die PEP-Nachrichten und ein Message-Integrity-Objekt beinhaltet, das mit dem geheimen Schlüssel generiert wurde. Der PEP überprüft die Prüfsumme der eingegangenen CAT-Nachricht und ist danach für die Fortsetzung der Kommunikation bereit. Ab diesem Zeitpunkt werden die Nachrichten, die vom PEP beziehungsweise vom PDP empfangen werden, immer auf fortlaufende Sequenznummer und korrekte Prüfsumme hin überprüft. Der Ablauf wird in Abbildung 2 veranschaulicht.

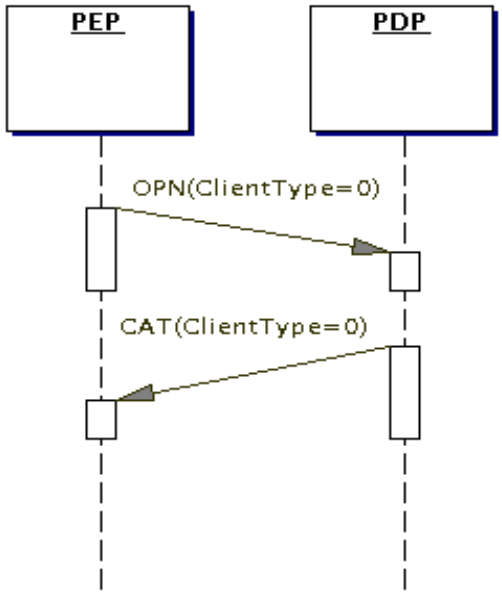


Abbildung 2: Authentifizierungshandshake für eine neu geöffnete COPS-Verbindung

- **Herstellen der Client-Type-Verbindung**

Nachdem der Authentifizierungshandshake erfolgreich abgeschlossen wurde, öffnet der PEP eine Client-Type-Verbindung mit dem Client-Type 1 (COPS-RSVP), indem er eine OPN-Nachricht mit dem Client-Type 1 an den PDP versendet. Die OPN-Nachricht beinhaltet die Adresse des letzten PDP, der vom PEP benutzt wurde. Da der PDP im Beispiel diesen Client-Type unterstützt, antwortet er mit einer CAT-Nachricht mit dem Client-Type 1. Der Ablauf wird in Abbildung 3 veranschaulicht.

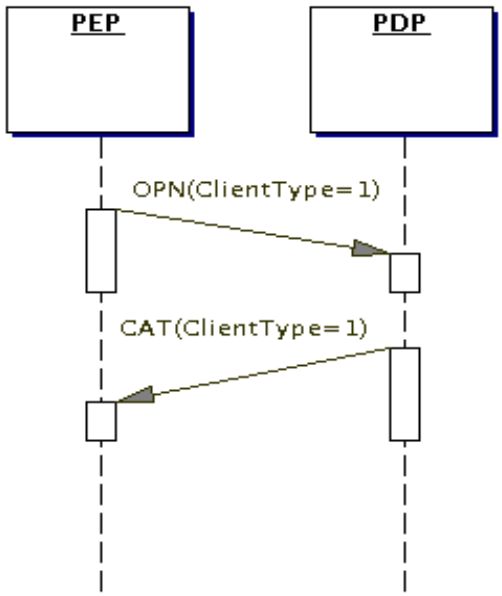


Abbildung 3: Herstellen der Client-Type-Verbindung für den Client-Type 1

- **Zustandssynchronisation (State-Synchronisation)**

Da der PEP in der OPN-Nachricht angegeben hat, dass er vorher einen anderen PDP benutzt hat, verlangt der PDP die Zustandssynchronisation (State-Synchronisation), indem er eine SSQ-Nachricht an den PEP versendet. Der PEP muss jetzt für jeden aktiven Anfrage-Zustand eine REQ-Nachricht versenden und danach eine SSC-Nachricht als Markierung des Endes der Zustandssynchronisation verschicken. Der Ablauf wird in Abbildung 4 veranschaulicht.

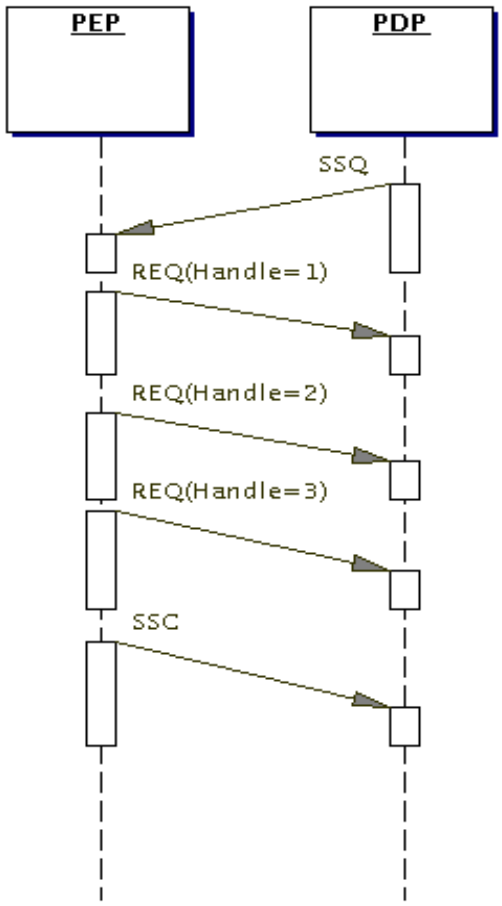


Abbildung 4: Zustandssynchronisation

- **Bearbeiten von Anfragen (Requests)**

Nach der Zustandssynchronisation (State-Synchronisation) können neue Anfragen bearbeitet werden. Der PEP erstellt einen neuen Anfrage-Zustand, indem er eine OPN-Nachricht an den PDP versendet. Der PDP versendet eine Entscheidung für die Anfrage in Form einer DEC-Nachricht. Die erste DEC-Nachricht, die für eine Anfrage versandt wird, muss das Solicited-Flag gesetzt haben. Der PEP antwortet auf die DEC-Nachricht mit einer RPT-Nachricht, die dem PDP signalisiert, dass die Umsetzung der Entscheidung erfolgreich war. Nachdem der PDP die erste Entscheidung an den PEP versandt hat, signalisiert er dem PEP durch den Versand einer neuen DEC-Nachricht für denselben Anfrage-Zustand, diesmal ohne das Solicited-Flag, die Änderung der Entscheidung. Der PEP bestätigt die Umsetzung der neuen Entscheidung mit einer RPT-Nachricht. Der Ablauf wird in Abbildung 5 veranschaulicht.

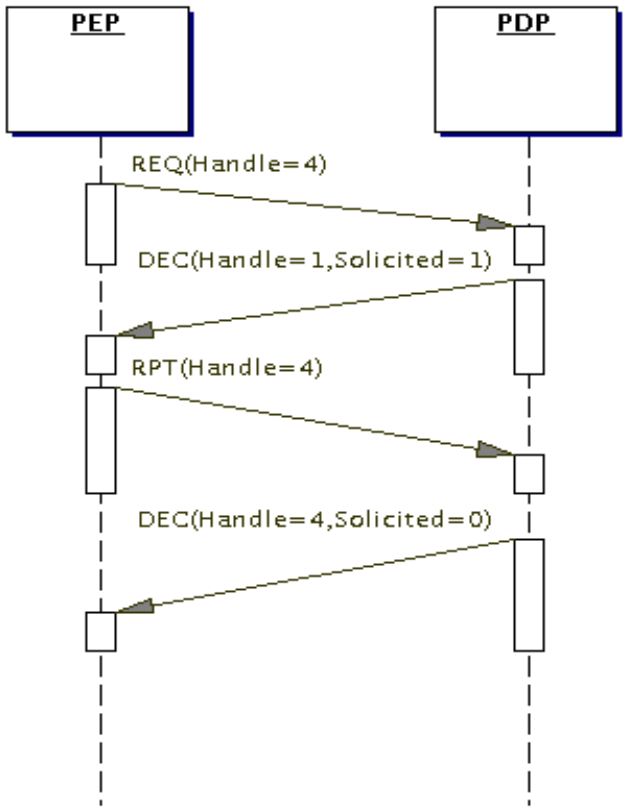


Abbildung 5: Bearbeiten von Anfragen

- **Schliessen der COPS-Verbindung**

Im angenommenen Beispiel muss der PDP jetzt heruntergefahren werden. Die Schliessung der Verbindung wird dem PEP mittels einer CC-Nachricht signalisiert. Da im ISP-Netzwerk noch ein PDP für den Client-Type 1 existiert, beinhaltet die CC-Nachricht ein PDP-Redirect-Adress-Objekt. Die Aufgabe von PDP-Redirect-Adress-Objekt ist die Umleitung vom PEP an den neuen PDP, so dass die Unterbrechung der Anfragenbearbeitung minimiert wird. Nach dem Versenden der CC-Nachricht schliesst der PDP die TCP-Verbindung. Der Ablauf wird in Abbildung 6 veranschaulicht.

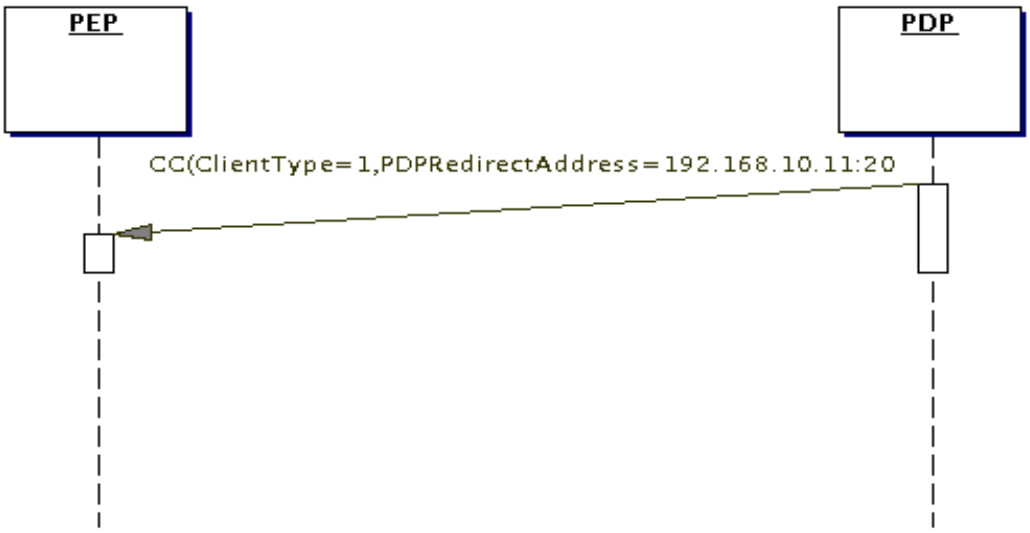


Abbildung 6: Schliessung der COPS-Verbindung mit Umleitung auf neuen PDP

2.5 COPS-RSVP

Speziell für die Authorisierung von RSVP-Nachrichten wurde das COPS-RSVP -Protokoll definiert [15]. Das COPS-RSVP-Protokoll definiert die Semantik und das Datenformat der COPS-Anfragen und Entscheidungen für den Fall, dass die RSVP-Nachrichten authorisiert werden sollen. Der COPS-RSVP-Client-Type verlangt vom PDP nur Entscheidungen für die folgenden RSVP-Nachrichten: PATH, RESV, PATHERR, RESVERR. Das Format der Objekte in den COPS-Nachrichten und deren Semantik wird im Folgenden dargestellt.

- **Die REQ-Nachricht**

Jede REQ-Nachricht, die vom PEP versandt wird, muss folgende Objekte beinhalten:

- **Client-Handle-Objekt**: Das Client-Handle-Objekt referenziert einen Anfrage-Zustand. Der Inhalt ist von PEP definiert und für den PDP transparent. Dieses Objekt wird vom PDP benutzt, um die Zugehörigkeit einer Entscheidung zu einem Anfrage-Zustand zu markieren.
- **In-Interface**: Ein In-Interface-Objekt definiert eine Eingangs-Netzwerkschnittstelle, auf die sich eine Anfrage (Request) bezieht beziehungsweise durch welche die Daten empfangen wurden, für die eine Anfrage entstanden ist. Ein In-Interface-Objekt besteht aus der IP-Adresse (IPV4 oder IPV6) und einer Indexnummer, die für eine Netzwerkschnittstelle innerhalb eines PEPs eindeutig ist.
- **Out-Interface**: Ein Out-Interface-Objekt definiert eine Ausgangs-Netzwerkschnittstelle, auf das sich eine Anfrage (Request) bezieht beziehungsweise durch welche die Daten versandt werden sollen, falls der PDP so entscheidet. Ein Out-Interface-Objekt besteht aus der IP-Adresse (IPV4 oder IPV6) und einer Indexnummer, die für eine Netzwerkschnittstelle innerhalb eines PEPs eindeutig ist.
- **Context-Objekt**: Der Anfragetyp (Request-Type) des Context-Objekts darf nur folgende Werte haben: Incoming-Message, Resource-Allocation und Outgoing-Message. Der Incoming-Message-Anfragetyp wird verwendet, um eine eingehende RSVP-Nachricht zu authentifizieren. Der Ressource-Allocation-Anfragetyp wird verwendet, um die Allozierung der Netzwerkressourcen (Einstellung von QoS für den Datenfluss) zu authorisieren. Der Outgoing-Message-Anfragetyp wird verwendet, um das Versenden von RSVP-Nachrichten zu authorisieren. Der Nachrichtentyp (Message-Type) des Context-Objekts muss den Wert von „message type“ aus der RSVP-Nachricht enthalten.
- **Signaled-ClientSI-Objekt**: Die clientspezifischen Daten der Anfrage beinhalten alle Objekte aus einer RSVP-Nachricht im selben Format, wie sie innerhalb der RSVP-Nachricht dargestellt werden.

- **Die DEC-Nachricht**

Jede Entscheidungsnachricht (DEC-Nachricht), die vom PDP an den PEP versandt wird, muss folgende COPS-Objekte beinhalten:

- **Client-Handle-Objekt**: Das Client-Handle-Objekt dient als Referenz, die es dem PEP ermöglichen soll, eine Entscheidung einem Anfrage-Zustand (Request-State) zuzuordnen. Der Inhalt dieses Objekts ist derselbe wie der Inhalt des Client-Handle-Objekts aus der REQ-Nachricht.

- **Context-Objekt**: Auch in der DEC-Nachricht muss ein Context-Objekt mit dem Wert des Anfragetyps (Request-Type) enthalten sein. Die darin gesetzten Bits dürfen nur eine Untermenge der Bits sein, die im Context-Objekt aus der REQ-Nachricht gesetzt wurden. So kann sich die Entscheidung nur auf einen Teil der Bearbeitung beziehen (zum Beispiel nur auf die Allozierung der Ressourcen). Der Wert des Nachrichtentyps des Context-Objekts muss derselbe sein wie der Wert aus der REQ-Nachricht.
- **Decision-Flags**: Falls die Entscheidung des PDP positiv ist, hat der Befehlscode (Command-Code) aus dem Decision-Flags-Objekt den Wert 1= Install; bei einer negativen Entscheidung ist der Wert 2= Remove. Falls die Entscheidung des PDP negativ ist, muss der PEP die Nachricht, die den Anfrage-Zustand generiert hat, so behandeln, als ob sie nie eingetroffen wäre. Die Flags des Decision-Flags-Objekts dürfen nur den Wert 1 haben. Das Flag 1 signalisiert dem PEP im Fall einer negativen Entscheidung, dass er eine entsprechende Fehlernachricht (PATHERR oder RESVERR) generieren soll.
- **Decision-Replacement-Data**: Falls der PDP dem PEP signalisieren will, dass er die RSVP-Objekte aus einer Objektklasse ersetzen soll, muss er ein Decision-Replacement-Data-Objekt in die DEC-Nachricht einfügen. Es enthält die Liste der neuen Objekte, die anstelle der alten Objekte in die RSVP-Nachricht eingefügt werden sollen. Die Regeln für den Ersatz der RSVP-Objekte sind:
  1. Nur RSVP-Objekte vom Typ POLICY_DATA, ERROR_SPEC and FLOWSPEC können ersetzt werden.
  2. Falls in Decision-Replacement-Data ein oder mehrere RSVP-Objekte mit einer Klassennummer erscheinen, werden in der ursprünglichen RSVP-Nachricht alle Objekte mit dieser Klassennummer durch die neuen Objekte ersetzt.
  3. Falls in Decision-Replacement-Data ein Objekt mit einer Klassennummer erscheint, das keine Daten beinhaltet, werden alle RSVP-Objekte mit dieser Klassennummer aus der ursprünglichen RSVP-Nachricht entfernt.

Durch diese Definition der REQ- und DEC-Nachrichten erhält der PDP volle Kontrolle über die Entscheidung, ob eine RSVP-Nachricht bearbeitet oder verändert werden soll. Insbesondere ist der PDP in der Lage, durch das Entfernen und Einfügen von POLICY_DATA Objekten beliebige Authentifizierungs- und Authorisierungsschemata für RSVP zu erzwingen.

3 Das neu entwickelte Authentifizierungsschema

Für die Entwicklung des Authentifizierungsschemas wurde das nachfolgend beschriebene Szenario zugrunde gelegt.[10]

Ein Inhaltsanbieter stellt über das Internet verschiedene Inhalte (Audio- oder Videoübertragungen) zur Verfügung, wobei er die Vermarktung der Sendungen aber nicht selbst übernimmt, sondern sie einem oder mehreren Händlern überlässt, die auf den Vertrieb dieses Warentyps spezialisiert sind. Potentielle Empfänger – also Kunden, die am Empfang der Sendungen interessiert sind – können gegen Bezahlung von einem Händler so genannte Tickets erhalten. Diese Tickets können als eine Art Gutscheine betrachtet werden, die der Empfänger gegen eine Sendung eintauschen kann. Zusätzlich zu den Tickets erhält der Empfänger vom Händer alle für den Empfang der Daten notwendigen Informationen (also etwa die Adresse der Multicast-Gruppe, die Portnummer etc.) sowie die Adresse eines Ticketservers. Nachdem der Empfänger die Tickets, die Adresse des Ticket-Servers und alle anderen nötigen Informationen für die Datenübetragung erhalten hat, ist er nun in der Lage, Sendungen zu empfangen.

Die Daten einer Sendung sind dabei jedoch verschlüsselt,[11] und um eine Sendung nach dem Empfang auch nutzen (also sichtbar und/oder hörbar machen) zu können, muss der Empfänger einen Schlüssel besitzen. Um einen solchen Schlüssel zu erhalten, muss der Empfänger eine sichere Verbindung zum Ticketserver herstellen, von dem er im Austausch gegen ein Ticket einen gültigen Schlüssel bekommt. Jeder Schlüssel hat dabei nur eine zeitlich beschränkte Gültigkeit, so dass der Empfänger rechtzeitig vor Ablauf der Gültigkeitsdauer im Austausch gegen das nächste Ticket beim Ticket-Server einen neuen Schlüssel anfordern muss. Dieses Verfahren ermöglicht die Rückerstattung des Geldes für nicht verbrauchte Tickets, falls der Empfänger sich entscheidet, die Übertragung nicht bis zum Ende zu verfolgen.

Die wichtigsten Komponenten für das angenommene Szenario und deren Relationen sind in Abbildung 7 dargestellt.

10 Das hier zugrunde gelegte Szenario für die Datenübertragung entspricht dem in [16] beschriebenen.

11 Bei der Erörterung des hier dargestellten Szenarios in [16] wurde vorgeschlagen, dass nicht alle Daten verschlüsselt werden, sondern nur der wichtigste Teil, der den grössten Informationsgehalt aufweist. Für die in der vorliegenden Diplomarbeit angestellten Überlegungen ist dieser Unterschied jedoch nicht von Bedeutung.

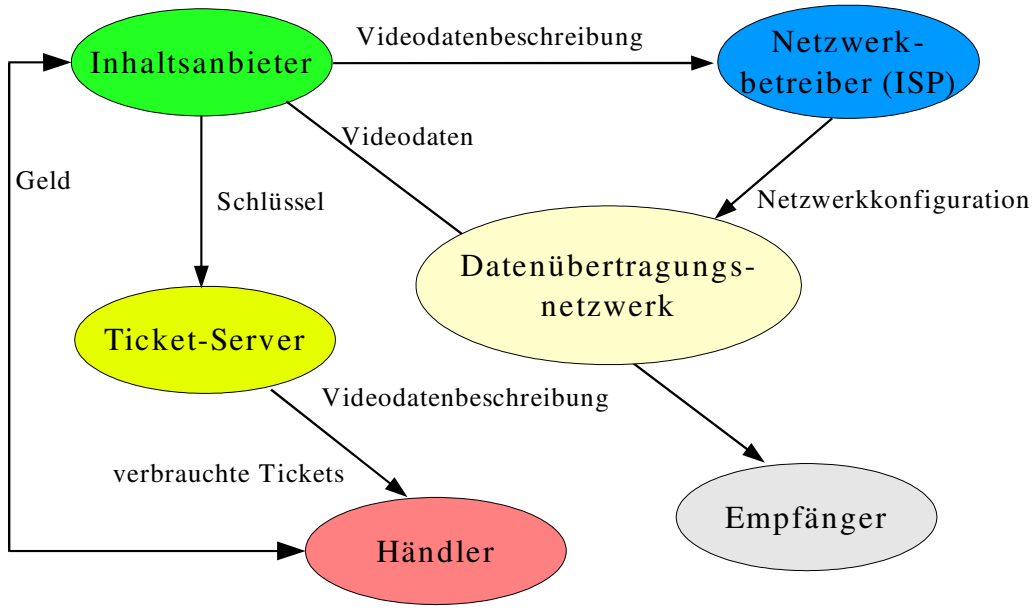


Abbildung 7:Relationen zwischen den Komponenten im angenommenen Szenario. (Abbildung nach [16], Seite 3).

Für die Reservierung der benötigten Netzwerkressourcen (also die Einstellung von QoS) wird das RSVP-Protokoll verwendet. Ein weiterer Aspekt des angenommenen Szenarios, auf den in [16] nicht explizit eingegangen wird, besteht darin, dass an der Datenübetragung mehrere voneinander unabhängige Netzwerkbetreiber (ISPs) beteiligt sind. Sie arbeiten zwar mit dem Inhaltsanbieter zusammen, stehen aber in Konkurrenz zueinander und sind daher mit an Sicherheit grenzender Wahrscheinlichkeit nicht bereit, sich bei der Implementierung des Authentifizierungsschemas auf andere ISPs zu verlassen. Der Grund hierfür liegt darin, dass das Ziel der Authentifizierung letztendlich darin besteht, die eigenen Ressourcen optimal zu nutzen und gegen ungerechtfertigte Nutzung durch andere zu schützen.

Die mögliche Topologie eines Netzwerkes unter Berücksichtigung dieses zusätzlichen Aspektes wird in Abbildung 8 dargestellt.

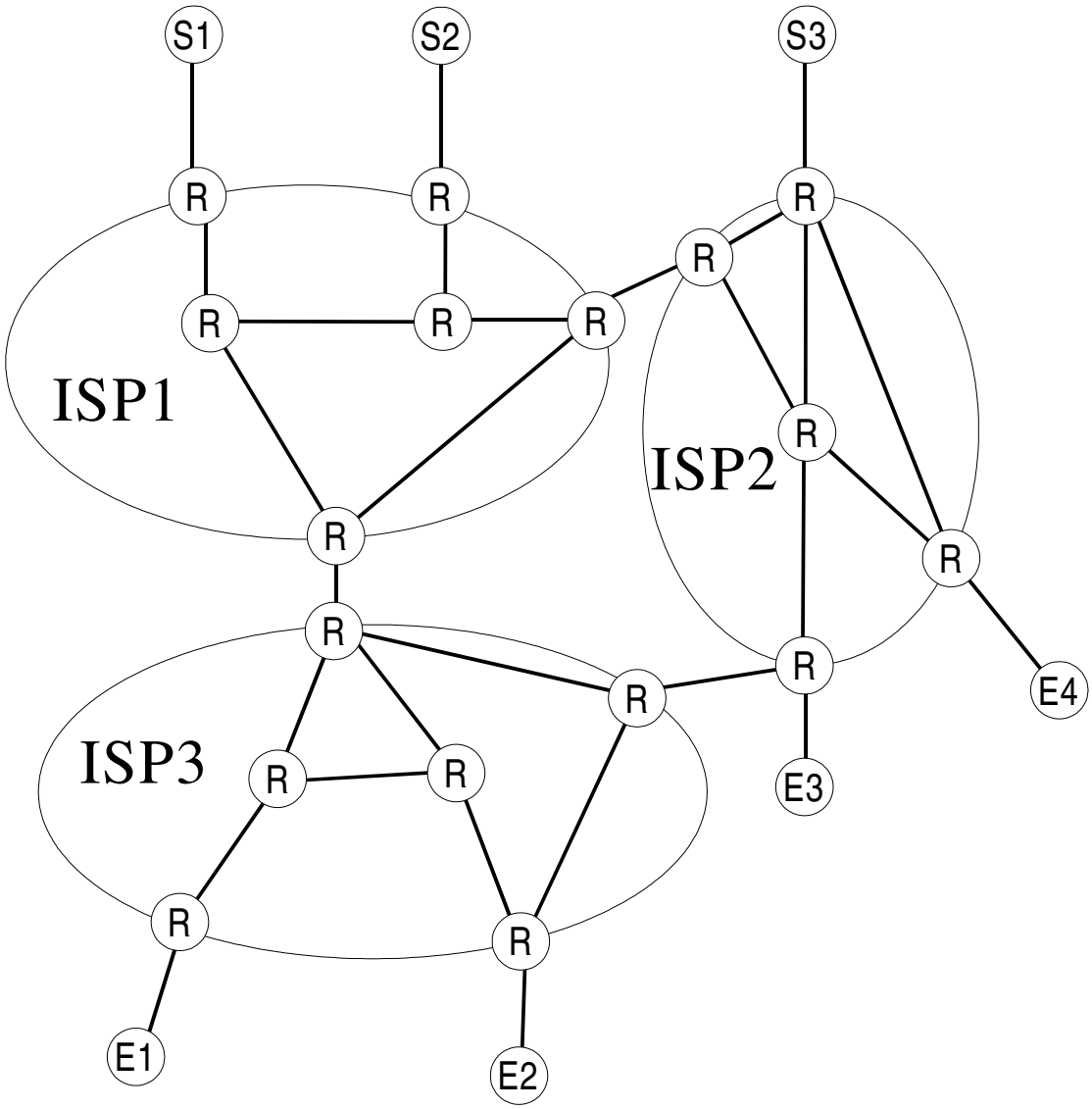


Abbildung 8:Ein Beispiel für die Netzwerktopologie im angenommenen Szenario. S1, S2, S2 = Sender des Inhaltsanbieters; R = Router, ISP1, ISP2, ISP3 = Administrative Domäne eines ISP; E1, E2, E3, E4 = Empfänger

Durch die Verschlüsselung der Audio- und Videodaten wird das Ziel erreicht, dass nur zahlende Empfänger Übertragungen empfangen können. Es ist jedoch immer noch möglich, dass ein Empfänger sich in die Multicast-Gruppe für die Datenübertragung anmeldet und per RSVP Netzwerkressourcen reserviert, obwohl er für den Empfang der Daten keinen Schlüssel besitzt. Falls sehr viele solche Empfänger die Netzwerkressourcen der ISPs belegen, entsteht eine so genannte Denial-Of-Service-Attacke (DoS). Die Konsequenz einer solchen Attacke ist, dass neu hinzu kommende berechtigte Empfänger nicht in der Lage sind, die Netzwerkressourcen für sich zu reservieren, da die Ressourcen schon durch die unberechtigten Reservierungen belegt sind. Dieser Zustand bringt dem Angreifer keinen direkten Nutzen (die Daten sind verschlüsselt und können nur mit einem gültigen Schlüssel entschlüsselt werden, daher kann der Angreifer nichts mit ihnen anfangen), aber die Ressourcen der ISPs werden überbelegt. Damit wird die Infrastruktur lahmgelegt, was zu einem finanziellen Schaden für den ISP führen kann; ausserdem leidet die Qualität der Audio- und Videoübertragungen, was die Attraktivität des Angebots senkt.

Eines der Ziele der vorliegenden Arbeit ist die Entwicklung eines Authentifizierungsschemas für die Reservierung von Netzwerkressourcen, damit sie nur für solche Empfänger erfolgt, die auch

zum Empfang der übertragenen Daten berechtigt sind. Damit lässt sich eine Denial-Of-Service-Attacke verhindern.

3.1 Die Infrastruktur für die RSVP-Authentifizierung innerhalb eines ISP-Netzwerkes

Die übliche Art, wie die RSVP-Authentifizierung innerhalb eines ISP-Netzwerks erfolgt, ist die folgende:

- Die Authentifizierung wird nur innerhalb von Border-Routern[12] implementiert. Die internen Router implementieren aus Effizienzgründen keine Authentifizierung, weil davon ausgegangen wird, dass die Datenpfade innerhalb des ISP-Netzes sicher sind, also dass die Datenkommunikation innerhalb des ISP-Netzwerks weder von Angreifern abgehört noch verändert werden kann.
- Die Authentifizierung wird auf einen zentralen Policy-Server ausgelagert. Normalerweise wird das COPS-Protokoll als Kommunikationsprotokoll zwischen den Border-Routern und dem Policy Server verwendet. In den meisten Fällen wird dabei COPS-RSVP verwendet, ein spezieller Client-Type für das COPS-Protokoll, durch den die RSVP-Authentifizierung auf den Policy-Server ausgelagert wird (vgl. Seite 29).

Abbildung 9 veranschaulicht das Verfahren an einem Beispiel.

12 Border-Router sind Router, die mit Routern ausserhalb des Netzes des ISP Datenverbindungen haben.

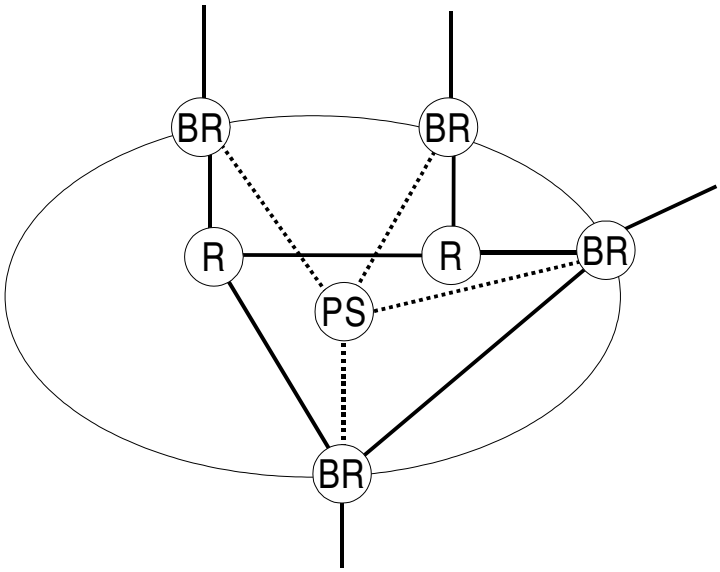


Abbildung 9: Typische Infrastruktur für die RSVP-Authentifizierung innerhalb eines ISP-Netzwerks. BR = Border-Router, R = Interne Router, PS = Policy Server. Halbfette Linien = Datenverbindungen; gestrichelte Linien = COPS-Verbindungen; elliptische einfache Linie = Grenze der administrativen Domain des ISP.

Laut Definition von COPS-RSVP kann der Policy-Server des ISP Entscheidungen für alle PATH-, PATHTEAR-, RESV- und RESTEAR-Nachrichten treffen, die in das ISP-Netzwerk hineinkommen oder es verlassen. Dabei entscheidet er einerseits, ob sie gültig sind, kann aber andererseits auch ihre POLICY_DATA-Objekte nach Belieben löschen, verändern oder neu einfügen. Dadurch kann der Policy-Server ein beliebiges Authentifizierungsschema erzwingen.

Bei der Konzeption der vorliegenden Diplomarbeit gab es auch die Überlegung, den so genannten Bandwidth-Broker[13] von Günther Stattenberger [17] einzusetzen. Ein Bandwidth-Broker ist ein zentraler Dienst innerhalb eines ISP-Netzes, der in der Lage ist, für einen gegebenen Datenfluss in allen an der Datenübertragung beteiligten Routern QoS einzustellen. Für die Kommunikation zwischen dem Policy-Server und dem Bandwidth-Broker sowie zwischen dem Bandwidth-Broker und den Routern wird gewöhnlich eines der so genannten RPC-(Remote Procedure Call)-Protokolle verwendet. Dabei handelt es sich um Protokolle, die den Aufruf von Prozeduren auf anderen Rechnern im Netzwerk ermöglichen.[14]

Der Vorteil der Benutzung des Bandwidth-Brokers bestünde darin, dass die Router innerhalb eines ISP-Netzwerks nicht notwendigerweise das RSVP-Protokoll unterstützen müssten, denn die

13 Auf eine mögliche Übersetzung – etwa mit „Band breiten-Vermittler“ oder Ähnlichem – wurde an dieser Stelle bewusst verzichtet, da es sich um einen Fachbegriff handelt, für den sich bisher noch keine deutsche Entsprechung eingebürgert hat.

14 Auf eine ausführlichere Darstellung der Funktionsweise von RPC wird an dieser Stelle verzichtet, da sie für die vorliegende Arbeit keine Rolle spielt. Für die exemplarische Beschreibung eines der möglichen RPC-Protokolle sei auf [18] verwiesen.

Implementierung des RSVP-Protokolls würde durch den Policy-Server übernommen. Damit wäre das gesamte ISP-Netzwerk ein virtueller RSVP-Router, als dessen Netzwerkschnittstellen diejenigen Netzwerkschnittstellen der realen Border-Router fungieren würden, die Verbindungen zu Routern ausserhalb der ISP-Domain haben (vgl. Abbildung 10). Die Aufgabe des Bandwidth-Brokers wäre dann mit der Aufgabe eines Traffic-Control-Moduls in einem klassischen Router vergleichbar.

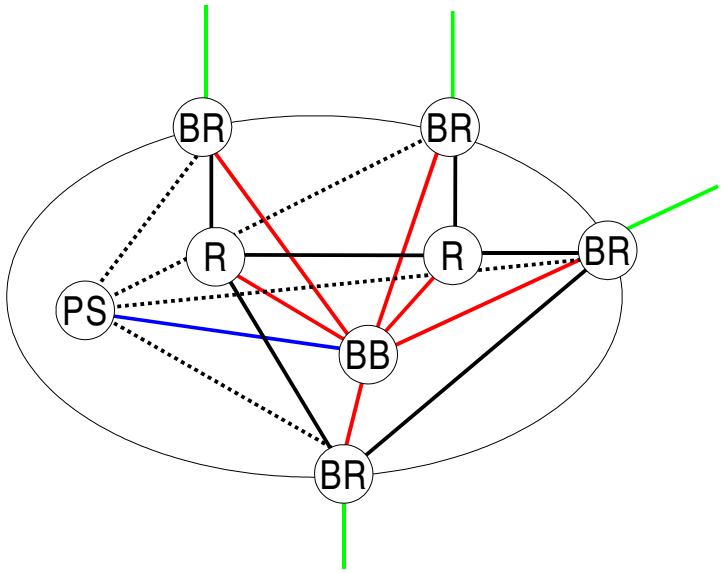


Abbildung 10: Infrastruktur eines virtuellen RSVP-Routers. BR = Border-Router, R = Interner Router, PS = Policy Server, BB = Bandwidth-Broker. Halbfette Linien: schwarz = Datenverbindungen; blau = RPC-Verbindung vom Policy-Server zum Bandwidth-Broker; rot = RPC-Verbindung vom Bandwidth-Broker zu den Routern; grün = Verbindungen zu Routern ausserhalb der ISP-Domain; gestrichelte Linien=COPS-Verbindungen; elliptische einfache Linie = Grenze der administrativen Domain des ISP.

Bei der weiteren Verfolgung dieses Konzepts ergab sich jedoch ein grundlegendes Problem: Die Semantik der Bandbreiten-Reservierungen, die der in Betracht gezogene Bandwidth-Broker unterstützt, erfüllt die Anforderungen an Bandbreiten-Reservierung für das RSVP-Protokoll mit Multicast nicht. Zwar ist der vorgesehene Bandwidth-Broker in der Lage, Reservierungsanforderungen mit folgenden Parametern umzusetzen:

- Adresse, Portnummer und Adressenmaske für den Datenabsender;
- Adresse, Portnummer und Adressenmaske für den Datenempfänger;
- Protokollnummer der Datenpakete;
- Adresse des Eingangsrouters im ISP-Netzwerk;
- QoS-Parameter für den Datenfluss (Bandbreite, Flags mit Informationen zur Datenverlust- und Verzögerungstoleranz etc.);
- Verfallszeit für die Reservierung.

Mit diesen Elementen kann QoS zwar für jede „Punkt-zu-Punkt“-(Unicast)-Datenübertragung problemlos definiert werden. Für die Definition von QoS bei Multicast-Datenübertragungen ergibt sich jedoch ein Problem: Es gibt keine Möglichkeit, die Adresse des Ausgangsrouters anzugeben. Ohne die Angabe des Ausgangsrouters werden aber durch eine Reservierung im Fall von Multicast die QoS-Parameter für den ganzen Abschnitt des Multicast-Baums ab dem Eingangsrouter definiert. Es sollte jedoch nur ein Zweig dieses Abschnitts reserviert werden, und zwar derjenige, der sich zwischen Eingangs- und Ausgangsrouter befindet. Da bei RSVP jeder Zweig des Multicast-Baums eine andere Reservierung haben kann, ist die Reservierung ganzer Multicast-Bäume nicht ausreichend, um den Anforderungen zu genügen, die durch das RSVP-Protokoll gestellt werden. Dies ist der Grund dafür, dass der Bandwidth-Broker nicht als Traffic-Control-Modul für den virtuellen RSVP-Router eingesetzt werden konnte. Daher wurde auch auf die Implementierung eines virtuellen RSVP-Routers verzichtet, denn hierfür wäre die Erweiterung des vorgeschlagenen Bandwidth-Brokers nötig gewesen, was den Rahmen der vorliegenden Arbeit bei Weitem gesprengt hätte.

3.2 Mögliche Lösungen für die Authentifizierung

In [16] wurden zwei Methoden für die Authentifizierung von RESV-Nachrichten vorgeschlagen, wobei sich die Authentifizierung nur durch die Art der Daten unterscheidet, die für die Erstellung des Nachweises verwendet werden, dass der Empfänger sich im Besitz eines gültigen Schlüssels befindet. Die beiden Methoden werden nachfolgend kurz vorgestellt.

3.2.1 Verwendung einer Prüfsumme der empfangenen Videobilder

Da nur ein zahlender Empfänger in der Lage ist, die Videodaten der Übertragung zu entschlüsseln, kann die Authentifizierung durchgeführt werden, indem jeder Empfänger seine Fähigkeit beweist, die Daten zu entschlüsseln. Dies tut er, indem er eine Prüfsumme von einem Teil des empfangenen Bildes als POLICY_DATA-Objekt in der RESV-Nachricht versendet.

Der Ablauf der Authentifizierung bei der Verwendung der Prüfsumme der empfangenen Videobilder wäre folgender:

1. Nach dem Empfang eines gültigen Schlüssels für die Entschlüsselung der Videodaten meldet sich der Empfänger in der Multicast-Gruppe für die Datenübertragung der Sendung an.
2. Sobald der Empfänger die Daten der Übertragung zu empfangen beginnt, kann er den Schlüssel verwenden, um die Daten zu entschlüsseln. Er ist nun in der Lage, die Prüfsumme für ein bestimmtes Bild aus den empfangenen Daten zu berechnen.
3. Der Empfänger verwendet die Prüfsumme aus Schritt 2, um ein POLICY_DATA-Objekt für die RESV-Nachricht zu erstellen. Dadurch wird die RESV-Nachricht authentifiziert.
4. Ein Router überprüft die Prüfsumme aus der eingehenden RESV-Nachricht, indem er sie mit der Prüfsumme vergleicht, die er – je nach gewähltem Verfahren – entweder vom Inhaltsanbieter erhalten oder selbst berechnet hat. Eine RESV-Nachricht wird nur dann bearbeitet, wenn die Prüfsumme der Videodaten mit der Prüfsumme aus der RESV-Nachricht übereinstimmt.

Dieser Ansatz hat den Vorteil, dass nur diejenigen Empfänger, die in der Lage sind, die Videobilder zu entschlüsseln, auch die Möglichkeit haben, eine gültige Prüfsumme zu erstellen.

Dieser Ansatz hat jedoch auch Nachteile, und zwar:

- Entweder muss der Datenverkehr von jedem Router oder wenigstens einmal innerhalb jedes ISP untersucht werden, um die gültige Prüfsumme zu ermitteln, wobei folglich jeder ISP auch selbst einen gültigen Schlüssel besitzen muss. Dieses Verfahren ist nicht gut skalierbar, denn wenn ein ISP Ströme von mehreren Audio- und Videoübertragungen analysieren muss, muss er sehr viele Ressourcen verwenden. Oder aber der Inhaltsanbieter muss den ISPs die gültigen Prüfsummen zur Verfügung stellen, was einen Mechanismus für die Distribution von Prüfsummen erforderlich macht.
- Es ist theoretisch möglich, wenn auch nicht sehr wahrscheinlich, dass der Empfänger bei grossen Netzwerküberlastungen durch den Best-Effort-Traffic nie genug Daten erhällt, um eine gültige Prüfsumme zu erstellen, und dadurch nicht in der Lage ist, eine gültige RESV-Nachricht zu versenden.
- Diese Art der Authentifizierung ist ausserdem anfällig für Copy-And-Paste-Attacken (siehe hierzu ausführlicher im Folgenden, Seite 39).

3.2.2 Verwendung eines Handshake-Verfahrens

Im Unterschied zur Verwendung von Prüfsummen der empfangenen Videobilder, bei denen ein Empfänger die empfangenen Daten zum Nachweis dafür benutzt, dass er in der Lage ist, die Daten zu entschlüsseln, und damit zugleich dazu berechtigt ist, die Reservierung zu betätigen, werden die Empfänger bei einem Handshake-Verfahren von den RSVP-Routern authentifiziert, ohne dafür die transportierten Daten zu verwenden. Ein Beispiel für ein Handshake-Verfahren wäre ein so genanntes Challenge-Response-Verfahren. Bei Challenge-Response-Verfahren werden in die PATH-Nachrichten zufällig generierte Daten (so genannte Challenges, auf Deutsch auch „Anforderungen“) als POLICY_DATA-Objekte eingefügt. Jeder Empfänger muss dann beim Versenden von RESV-Nachrichten ein POLICY_DATA-Objekt, die so genannte Response (deutsch: „Antwort“), in die Nachricht einfügen.[15] Die Antwort wird durch Verschlüsselung mit dem Schlüssel für die empfangenen Daten aus der Anforderung erstellt. Jeder RSVP-Router kann dann anhand der Antwort feststellen, ob die RESV-Nachricht gültig ist beziehungsweise ob sie von einem authorisierten Empfänger stammt. Zu diesem Zweck kann er beispielsweise die Antwort entschlüsseln und das Ergebnis mit der Anforderung vergleichen.

Der Ablauf der Authentifikation bei einem Challenge/Response-Verfahren wäre folgender:

1. Jeder RSVP-Router, der die Reservierung von Netzwerkressourcen durchführen möchte, bindet in die ausgehenden PATH-Nachrichten eine Anforderung (Challenge) ein. Eine solche

15 Im Folgenden werden die deutschen Begriffe „Anforderung“ und „Antwort“ verwendet. Da sie jedoch nicht als Fachtermini definiert sind, können sie in Abhängigkeit vom Kontext gelegentlich uneindeutig sein. Einerseits kann „Antwort“ nicht nur als Übersetzung für englisch „response“, sondern auch für „reply“ und „answer“ verwendet werden, so wie auch „Anforderung“ nicht nur die deutsche Übersetzung für „challenge“, sondern darüber hinaus auch für „request“ und „demand“ ist. Andererseits können die deutschen Begriffe auch mit anderen als den hier erforderlichen technischen Konnotationen gebraucht werden (etwa in einem Kontext wie „die an eine sichere Übertragung zu stellenden Anforderungen“ o. Ä.). Aus diesem Grund wird in der vorliegenden Arbeit an einigen Stellen der englische Begriff entweder in Klammern zum deutschen hinzugefügt oder in Ausnahmefällen auch ersatzweise für den deutschen Begriff verwendet. Zu den grundsätzlichen Problemen, die bei der Übertragung englischer Informatik-Fachbegriffe ins Deutsche auftreten, vgl. auch [19] und [20].

Anforderung kann zum Beispiel aus einem vom Router generierten Datensatz bestehen, der in der Folge auch bei wiederholtem Versenden unverändert bleibt.

2. Nach dem Empfang eines gültigen Schlüssels für die Entschlüsselung der Videodaten meldet sich der Empfänger in der Multicast-Gruppe für die Datenübertragung an.
3. Zusammen mit den Daten, die vom Inhaltsanbieter an die Multicast-Gruppe versandt werden, der er nun ebenfalls angehört, erhält der Empfänger auch die PATH-Nachrichten mit den Anforderungen (Challenges).[16]
4. Der Empfänger verwendet den Schlüssel, um die Anforderung (Challenge) zu verschlüsseln und damit die Antwort (Reply) zu generieren. Diese Antwort wird als POLICY_DATA-Objekt in die ausgehende RESV-Nachricht eingefügt.
5. Der Router überprüft die Antwort (Response) aus der eingehenden RESV-Nachricht, indem er sie mit dem gültigen Schlüssel entschlüsselt. Die RESV-Nachricht gilt als authentifiziert, wenn das Ergebnis der Entschlüsselung der Antwort mit der Anforderung (Challenge) identisch ist.

Der Vorteil dieses Verfahrens ist, dass es von den transportierten Daten unabhängig ist. Der Nachteil ist, dass sich entweder jeder Policy-Server im Besitz des gültigen Schlüssels für die übertragenen Daten befinden muss oder dass er die Verschlüsselung von Anforderungen an den Inhaltsanbieter auslagern muss. Letzteres stellt allerdings keinen wirklichen Nachteil gegenüber der Verwendung von Prüfsummen der empfangenen Bilder dar, da in beiden Fällen Kommunikation mit dem Inhaltsanbieter erfolgen muss.

3.3 Der gemeinsame Nachteil der vorgeschlagenen Authentifizierungsmethoden

Beide vorgeschlagenen Authentifizierungsmethoden haben einen gemeinsamen Nachteil: mit beiden Methoden ist durch die so genannte Copy-And-Paste-Methode immer noch eine DoS-Attacke möglich.

Die Möglichkeit zur Initialisierung einer DoS-Attacke entsteht dadurch, dass die Prüfsumme von empfangenen Bildern oder die Anforderung (Challenge) für jeden Zweig des Multicast-Baums Gültigkeit haben. Mit anderen Worten reicht es aus, wenn der Angreifer in der Lage ist, eine gültige RESV-Nachricht zu analysieren, um selbst weitere gültige RESV-Nachrichten für jeden Zweig des Multicast-Baums erstellen zu können. Es genügt nämlich, einfach das POLICY_DATA-Objekt aus der gültigen RESV-Nachricht in eine neue RESV-Nachricht zu kopieren – daher die Bezeichnung „Copy-And-Paste-Attacke“ – , um zu erreichen, dass die neue RESV-Nachricht authentifiziert wird. Falls der Angreifer in der Lage ist, den Datenverkehr auf einem authentifizierten Zweig des Multicast-Baums längerfristig zu überwachen, kann er permanent Reservierungen auf dem ganzen Multicast-Baum aufrecht erhalten und durch diese DoS-Attacke grossen Schaden verursachen. Da ISPs normalerweise Konkurrenten sind, kommen auch ISPs selbst als potentielle Angreifer in Frage. Vor allem weil jeder ISP in der Lage ist, den Datenverkehr im eigenen Netzwerk zu überwachen (vgl. Abbildung 8, Seite 33), wäre es möglich, dass ein ISP eine DoS-Attacke gegen den anderen durchführt. Dieser Aspekt muss auch vom Authentifizierungsschema berücksichtigt werden.

16 Der Grund hierfür besteht darin, das bei RSVP die PATH-Nachrichten an die Adresse des Empfängers versandt werden; im Fall von Multicast ist dies die Adresse der Multicast-Gruppe.

3.4 Die Anforderungen an ein sicheres Authentifizierungsschema

Aus der Analyse der Schwächen der vorgeschlagenen Authentifizierungsmethoden ergeben sich folgende Anforderungen an das Authentifizierungsschema:

1. Es soll nur einem authentifizierten Empfänger möglich sein, die Reservierung von Ressourcen vorzunehmen.
2. Die POLICY_DATA-Objekte aus den RESV-Nachrichten dürfen nur innerhalb eines Multicast-Zweigs gültig sein.
3. Jeder ISP muss in der Lage sein, selbst die Authentifizierung durchzuführen, ohne sich auf die Informationen von anderen ISPs zu verlassen.
4. Kein ISP darf in der Lage sein, ein gültiges POLICY_DATA-Objekt für einen anderen ISP zu generieren und damit die Möglichkeit zu haben, eine DoS-Attacke gegen andere ISPs durchzuführen.
5. Ein POLICY_DATA-Objekt aus einer RESV-Nachricht muss eine zeitlich begrenzte Lebensdauer haben, um das Aufrechterhalten von Reservierungen zu verhindern, wenn sich keine authentifizierten Empfänger mehr auf einem Zweig des Multicast-Baums befinden.
6. Zudem sollte für den Empfänger Datenschutz gewährleistet werden. Die Identität beziehungsweise die Adresse der Empfänger sollte daher höchstens den direkten ISPs des Empfängers bekannt sein. Der Inhaltsanbieter und die anderen ISPs sollten nicht in der Lage sein, Informationen über die Empfänger zu sammeln.

Im Rahmen des vorliegenden Arbeit wurde ein Authentifizierungsschema entwickelt, das alle hier aufgeführten Anforderungen erfüllt.

3.5 Der Ablauf der neu entwickelten Authentifizierung

Im Folgenden wird der Ablauf des in der vorliegenden Arbeit entwickelten Authentifizierungsschemas im Einzelnen erläutert.

3.5.1 Erstellen und Versenden von Anforderungen (Challenges)

Für jede PATH-Nachricht, die einen ISP durch eine Netzwerkschnittstelle eines Border-Routers verlässt, muss der ISP eine Anforderung (Challenge) definieren. Die Anforderung wird zusammen mit der passenden Antwort auf diese Anforderung von einem ISP des Inhaltsanbieters generiert und übertragen. Die Anforderung ist eine Zufallszahl, die mindestens 160-Bit (20 Bytes) lang sein muss. Der Grund für die Angabe einer Mindestlänge für die Anforderung liegt darin, dass die Wahrscheinlichkeit, dass zwei zufällig generierte Anforderungen dieser Länge identisch sind, minimal ist. Die Anforderung wird dann in Form eines POLICY_DATA-Objekts in die PATH-Nachricht eingebunden. Das POLICY_DATA-Objekt mit der Anforderung besteht aus einem POLICY_ELEMENT mit dem P-Type = 53248. Der Inhalt des POLICY_ELEMENTs sieht folgendermassen aus:

ISPID Length	Reserved(0)
ISPID	
Data	

Die Bedeutung der Felder ist:

- **ISPID Length (16 Bit)**: Die Länge des ISPID (in Bytes).
- **ISPID (Variable)**: Die ID eines ISP. Dabei handelt es sich um einen ASCII-String mit dem Domain-Namen des ISP (zum Beispiel „hispeed.ch“) .
- **Data (Variable)**: Die Anforderung (Challenge) für den ISP, dessen ID in ISPID definiert wurde. Der Anfang dieses Felds wird so aus der Länge von ISPID berechnet, dass das Offset in Relation zum Anfang des Objekts durch 4 teilbar ist. Wenn ISPID beispielsweise die Länge 3 hat, dann befindet sich der Anfang der Anforderung 8 Bytes nach dem Anfang des Inhalts des POLICY_ELEMENTs.

Jeder ISP, durch den eine PATH-Nachricht versandt wird, fügt der Nachricht also ein eigenes POLICY_DATA-Objekt mit einer Anforderung hinzu, so dass beim Empfänger eine PATH-Nachricht mit Anforderungen von allen ISPs ankommt. Da die Kommunikation zwischen dem ISP und dem Inhaltsanbieter eine Verzögerung der Bearbeitung von PATH-Nachrichten verursachen kann, wird empfohlen, dass der ISP einen Block (mehrere Challenge/Response-Paare auf einmal) vom Inhaltsanbieter anfordert und bei Bedarf verwendet. Die Lebensdauer einer Antwort (Response) ist als Refreshperiode der PATH-Nachricht*3 definiert.

3.5.2 Erstellen und Einfügen von Antworten (Responses)

Jeder Empfänger muss für die empfangene PATH-Nachrichten alle Anforderungen von allen ISPs speichern. Beim Versenden einer RESV-Nachricht muss der Empfänger für die letzte Anforderung von jedem ISP eine Antwort generieren und als POLICY_DATA-Objekt in die RESV-Nachricht einfügen. Eine Antwort ist eine SHA-1 [21] Prüfsumme der Anforderung und des Schlüssels für die Entschlüsselung der übertragenen Daten. Die Gründe für diese Auswahl der Antwort sind:

- Aus einer Prüfsumme können die Daten, die für die Herstellung der Prüfsumme verwendet wurden, nicht rekonstruiert werden.
- Die Prüfsumme über die Anforderung und den Schlüssel ist für jede Anforderung anders. Dabei gilt allerdings die Einschränkung, dass bei einer Prüfsumme immer so genannte Kollisionen möglich sind, dass also für unterschiedliche Daten dieselbe Prüfsumme berechnet werden kann.
- Die SHA-1 Prüfsumme wurde wegen bekannter Schwächen von MD5 [22] gewählt.

Eine Antwort wird als POLICY_DATA-Objekt in die RESV-Nachricht eingefügt. Dieses POLICY_DATA-Objekt hat nur ein POLICY_ELEMENT mit dem P-Type = 53249. Der Inhalt des POLICY_ELEMENTs ist derselbe wie bei der Anforderung, mit dem Unterschied, dass sich im Data-Feld nicht die Anforderung, sondern die Antwort befindet.

3.5.3 Die Authentifizierung von RESV-Nachrichten

Beim Eintreffen einer RESV-Nachricht im Border-Router eines ISP wird die RESV-Nachricht authentifiziert. Die Authentifizierung erfolgt durch Vergleich aller Antworten für den ISP (identifiziert durch den ISPID im POLICY_ELEMENT des POLICY_DATA-Objekts) mit den gespeicherten Antworten. Der Grund, wieso eine RESV-Nachricht mehr als eine Antwort für einen ISP beinhalten kann, ist die Zusammenfassung von RESV-Nachrichten. Bei der Zusammenfassung von RESV-Nachrichten werden alle POLICY_DATA-Objekte aus den Nachrichten, die zusammengefasst werden, in die resultierende RESV-Nachricht eingefügt. Ein weiterer Grund, wieso eine RESV-Nachricht mehrere Antworten beinhalten kann, liegt in der Tatsache, dass eine PATH-Nachricht einen ISP auf unterschiedlichen Pfaden mehrmals durchqueren kann und jedes Mal ein Challenge für den ISP generiert wird (vgl. Abbildung 11).

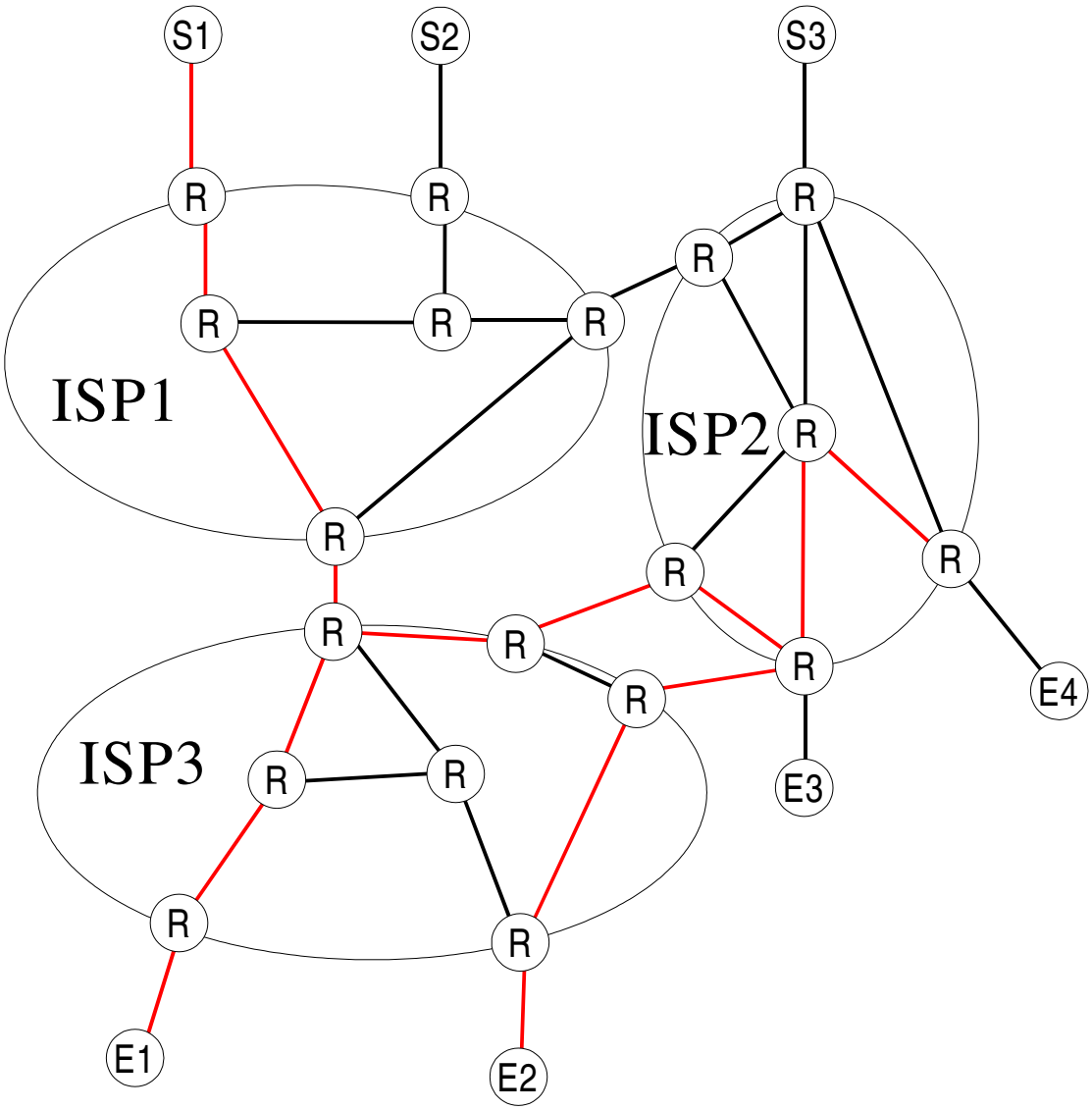


Abbildung 11:Ein Beispiel für einen Multicast-Baum, der das Netzwerk eines ISP mehrmals durchquert. Der Pfad vom Sender S1 zum Empfänger E3 verlässt zweimal das Netzwerk von ISP3. S1, S2, S2 = Sender, R = Router, ISP1, ISP2, ISP3 = Domain eines ISP, E1, E2, E3, E4 = Empfänger. Rote Linie = Multicast Baum für den Sender S1.

Eine RESV-Nachricht gilt als authentifiziert, falls sie mindestens eine gültige Antwort für den ISP enthält. Falls eine RESV-Nachricht gültig ist, werden alle gültigen Antworten für diesen ISP aus der Nachricht entfernt. Der Grund dafür, dass nur aktuell gültige Antworten aus der Nachricht entfernt werden, liegt darin, dass die PATH-Nachricht das Netzwerk des ISP auf unterschiedlichen Pfaden mehrmals durchqueren kann (vgl. Abbildung 11). Wenn beim ersten Eintreffen der RESV-Nachricht alle Antworten für den ISP entfernt werden, ist die RESV-Nachricht beim zweiten Eintreffen in das Netzwerk des ISPs nicht mehr gültig.

Falls eine RESV-Nachricht nicht authentifiziert ist, wird sie einfach nicht bearbeitet, sondern so behandelt, als ob sie nie eingetroffen wäre.

3.6 Die Analyse des Authentifizierungsschemas

Um zu überprüfen, ob das Authentifizierungsschema sicher ist, muss untersucht werden, ob alle Anforderungen an ein sicheres Authentifizierungsschema erfüllt sind. Diese sind:

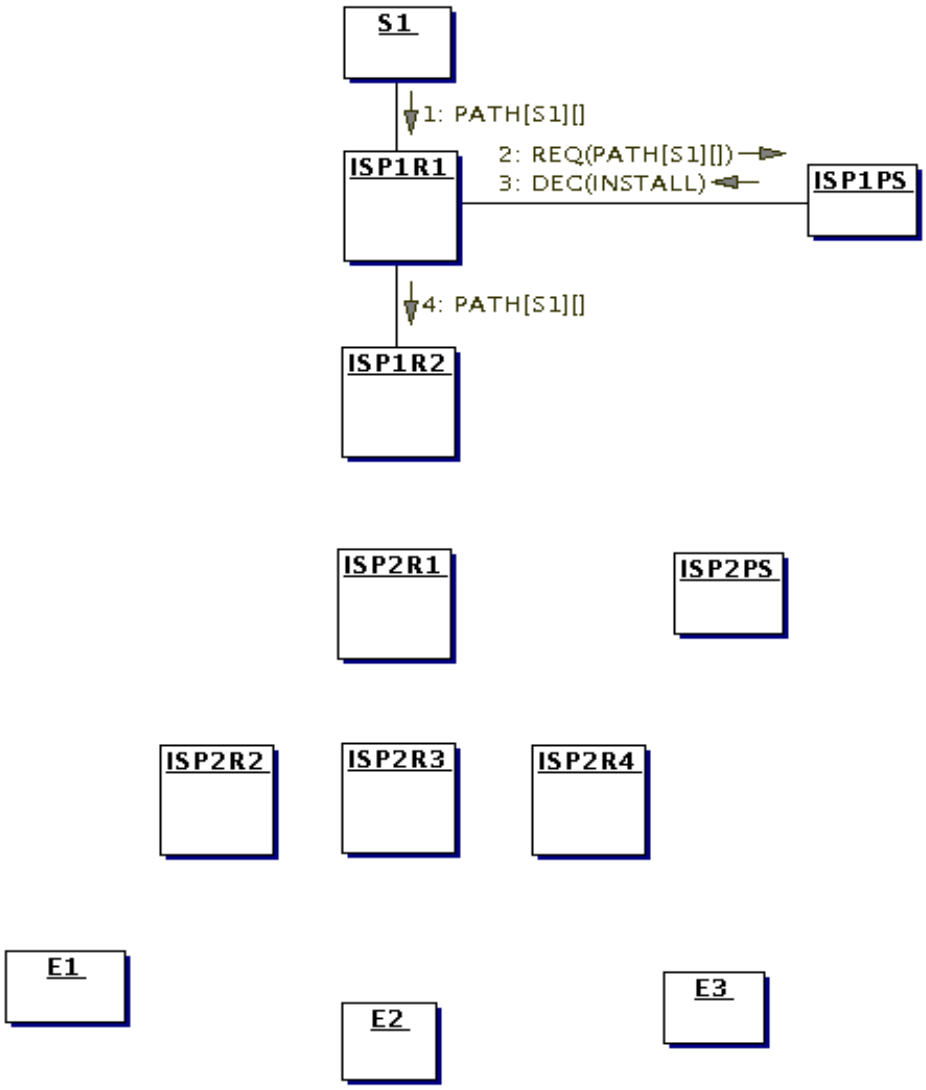
1. Nur ein Empfänger, der sich im Besitz eines Schlüssels für die Entschlüsselung der Daten befindet, ist in der Lage, die Antwort (die Prüfsumme über die Anforderung und den Schlüssel) zu generieren.
2. Die Anforderungen sind für jeden Zweig des Multicast-Baums verschieden, da unterschiedliche Anforderungen für die PATH-Nachrichten, die das Netzwerk eines ISP auf unterschiedlichen Datenverbindungen verlassen, generiert werden.
3. Da jeder ISP die Authentifizierung für die eigenen Anforderungen durchführt, ist er auf die Zusammenarbeit mit anderen ISPs nicht angewiesen.
4. Kein ISP befindet sich im Besitz des Schlüssels für die Entschlüsselung der Daten. Gleichzeitig werden die Anforderungen und Antworten vom Inhaltsanbieter erstellt. Damit hat keiner der ISPs die Möglichkeit, aus einer Anforderung eine gültige Antwort für einen anderen ISP zu erhalten.
5. Die Lebensdauer einer Antwort ist definiert als Refreshperiode der PATH-Nachricht*3 und damit begrenzt.
6. Die Adresse eines Empfängers ist nur dem ersten Router bekannt, der die RSVP-Nachrichten vom Empfänger erhält; die Empfängerdaten werden nicht innerhalb der RESV-Nachricht geführt, was den Schutz der Empfängeridentität ermöglicht.

3.7 Ein Beispiel für den Einsatz des Authentifizierungsschemas

Der Einsatz des Authentifizierungsschemas wird im Folgenden an einem konkreten Beispiel veranschaulicht. In diesem Beispiel werden Videodaten von einem Sender aus per Multicast durch zwei ISP-Netzwerke versandt. Die RSVP-Router eines ISP haben dabei Namen der Form IPSxRy, wobei x die Nummer des ISP und y die Nummer des jeweiligen ISP-Routers ist. Die Reservierung der Netzwerkressourcen geschieht per RSVP-Protokoll, das mit dem Authentifizierungsschema abgesichert ist. Die Authentifizierung der RSVP-Nachrichten jedes ISP wird auf die Policy-Server der ISPs ausgelagert. Die Namen der Policy-Server haben die Form ISPxPS, wobei x die Nummer des ISP ist. Die Videodaten werden von drei Empfängern (E1, E2, E3) empfangen. E1 und E2 befinden sich im Besitz des Schlüssels für die Entschlüsselung der Videodaten und können die richtigen Antworten auf die Anforderungen generieren. E3 besitzt keinen Schlüssel für die Entschlüsselung der Videodaten, versucht aber mit einer falschen Antwort eine Reservierung zu erreichen. Später unternimmt E3 mittels Copy-And-Paste eine Replay-Attacke (vgl. Seite 8). Das Beispiel nimmt folgenden Verlauf:

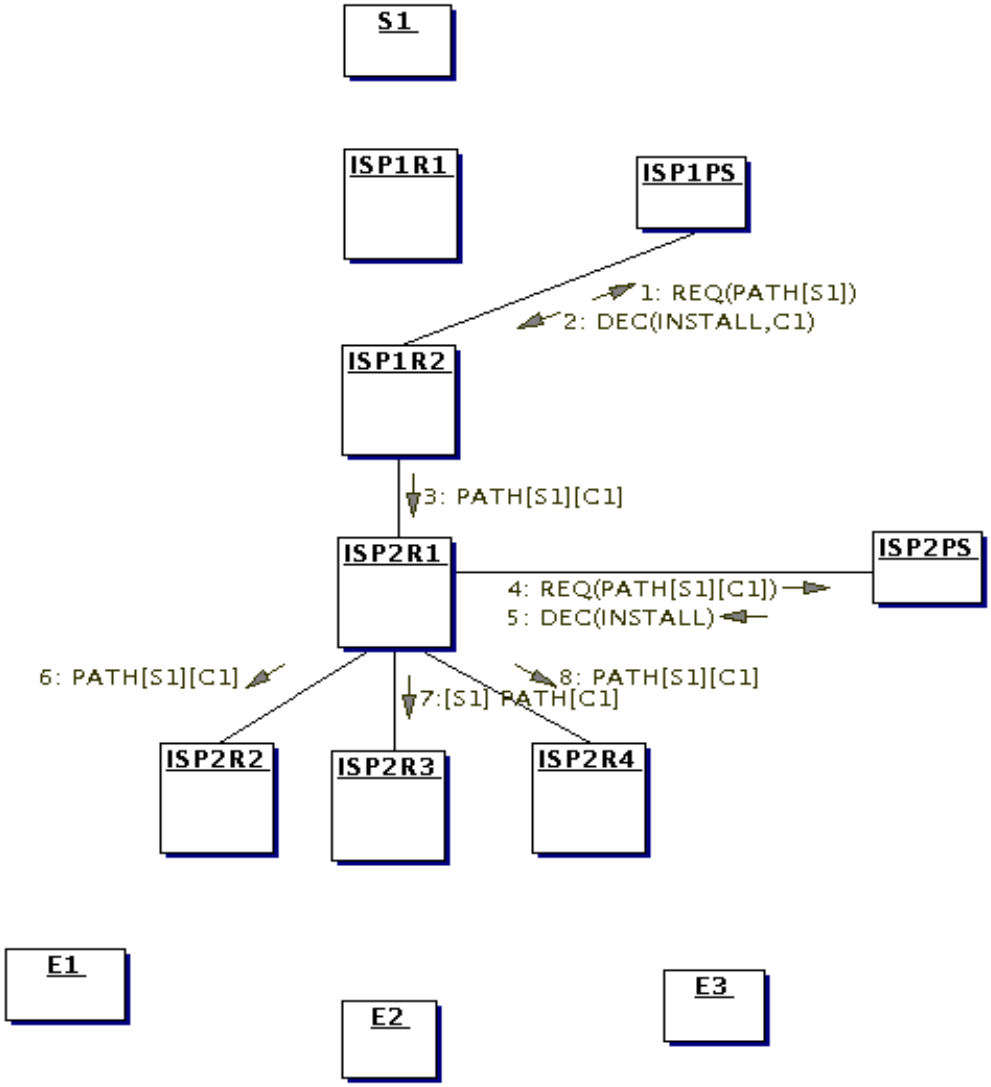
- **Versenden von PATH**

Zu Beginn versendet der Sender die PATH-Nachricht. Beim Eingang in das Netzwerk von ISP1 wird die PATH-Nachricht an den Policy-Server vom ISP1 versandt. Der Policy-Server entscheidet, dass die Nachricht ohne Änderung akzeptiert wird:



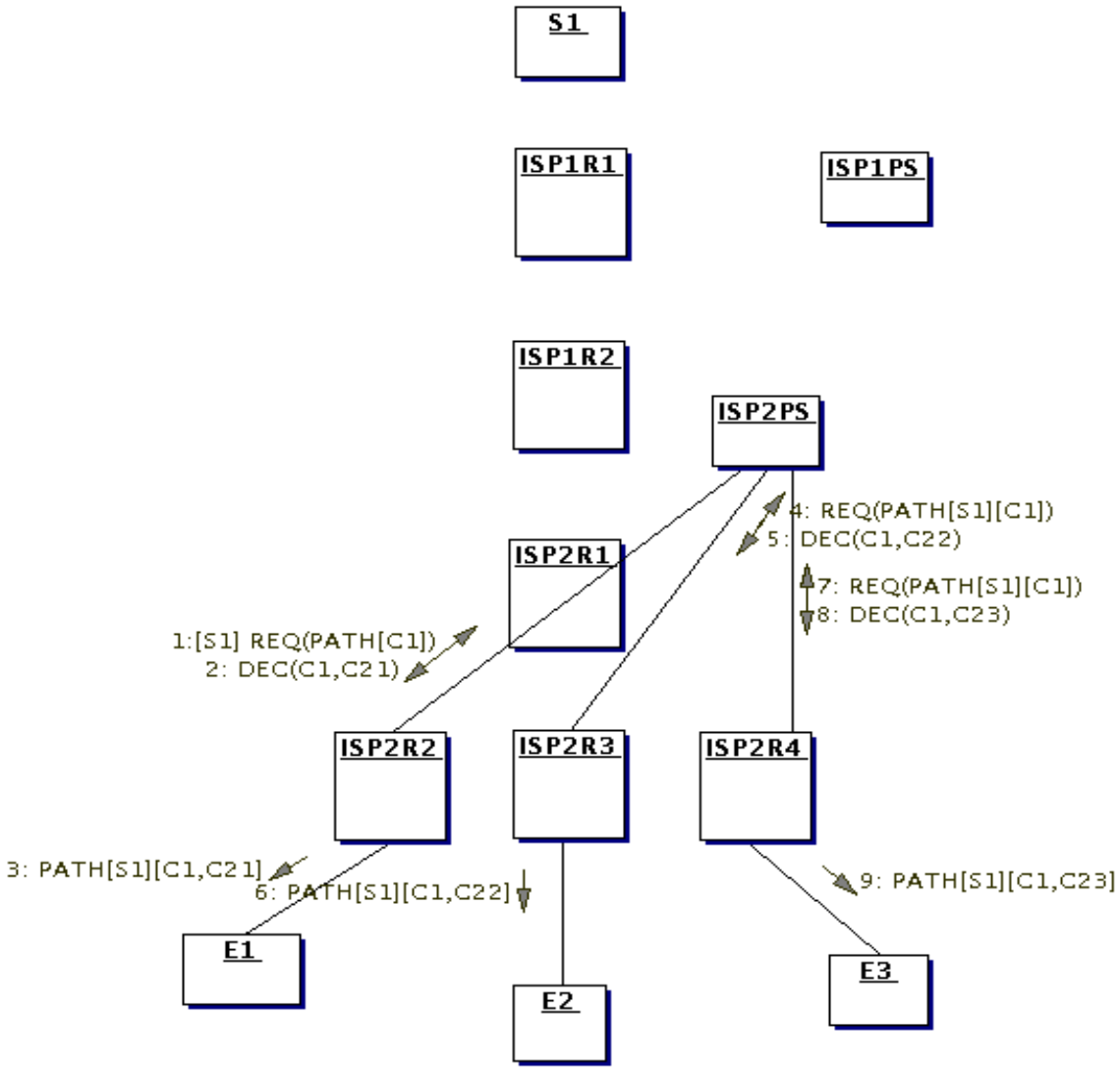
- **Generieren der ISP1-Anforderung (Challenge)**

Beim Verlassen des Netzwerks von ISP1 wird die PATH-Nachricht an den Policy-Server weitergeleitet. Der Policy-Server verlangt vom Inhaltsanbieter ein Challenge/Reponse-Paar und sendet dem Router die Entscheidung, das POLICY_DATA-Objekt mit der Anforderung1 (Challenge1, abgekürzt C1) in die PATH-Nachricht einzufügen. Die gültige Antwort wird vom Policy-Server gespeichert. Die PATH-Nachricht mit der Anforderung C1 wird dann durch den ISP1R2-Router an den ISP2 weitergeleitet. Nach dem Empfang der PATH-Nachricht wird an den Policy-Server die PATH-Nachricht versandt, die Entscheidung vom Policy-Server ist die Nachricht ohne Änderungen zu akzeptieren.



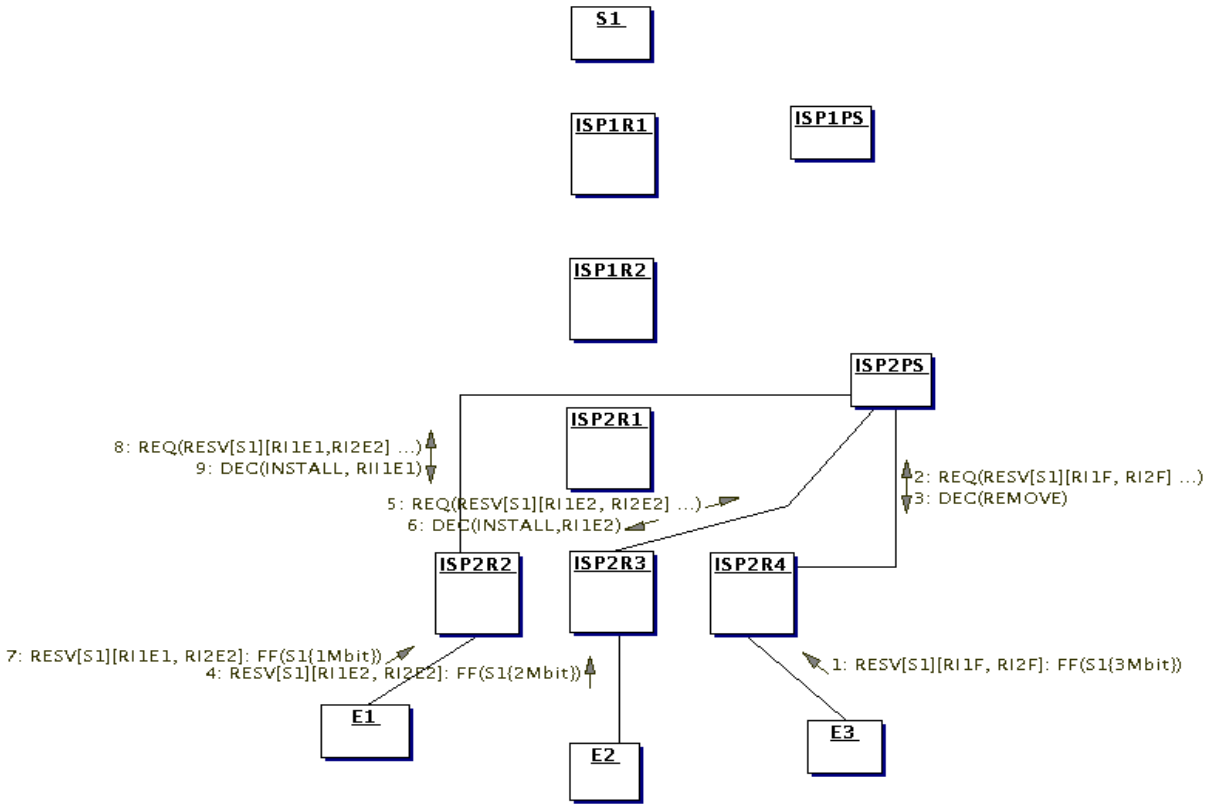
- **Generieren der ISP2-Anforderungen (Challenges)**

Da sich der Multicast-Baum im Netz von ISP2 verzweigt, wird die PATH-Nachricht repliziert und in Richtung aller Empfänger versandt. Beim Verlassen des Netzwerks von ISP2 werden die PATH-Nachrichten an den Policy-Server von ISP2 versandt. Der Policy-Server von ISP2 erhält vom Inhaltsanbieter drei unterschiedliche Anforderungen (C21, C22, C23), aus denen die POLICY_DATA-Objekte erstellt werden, die er dann als Entscheidungen an die Router weiterleitet. Die Router binden die entsprechenden POLICY_DATA-Objekte in die ausgehenden PATH-Nachrichten ein und versenden sie an die Empfänger.



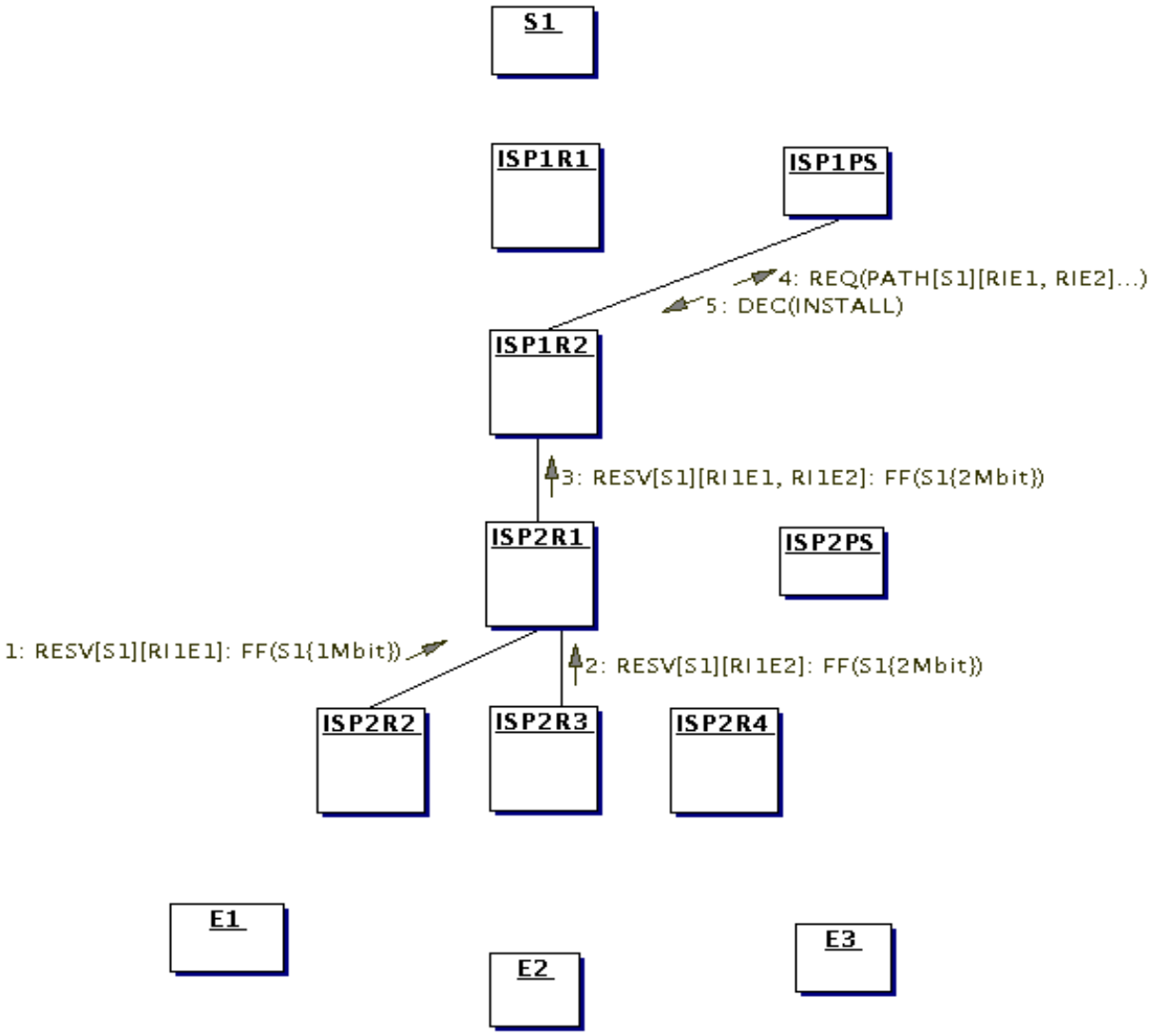
- **Versenden von RESV-Nachrichten mit Antworten (Responses), Authentifizierung bei ISP2**

Jeder Empfänger, der im Besitz eines Schüssels für die Entschlüsselung der Videodaten ist, generiert eine Antwort pro ISP-Anforderung (E1: RI1E1, RI2E1; E2: RI1E2, RI2E2). E3, der keinen Daten-Schlüssel besitzt, versucht, zwei gefälschte Antworten (RI1EF, RI1EF) zu generieren. Nach dem Empfang einer Nachricht versendet jeder Router eine Anfrage mit der empfangenen RESV-Nachricht an den Policy-Server von ISP2. Der Policy-Server vergleicht die Antworten mit den gültigen Antworten, die für jede Netzwerkschnittstelle jedes Routers verschieden sind, und versendet die entsprechende Entscheidung (Entfernen der gültigen Antworten und Akzeptieren der RESV-Nachrichten für IS2R2 und ISP2R3 sowie das Ignorieren der RESV-Nachricht für ISP2R4).



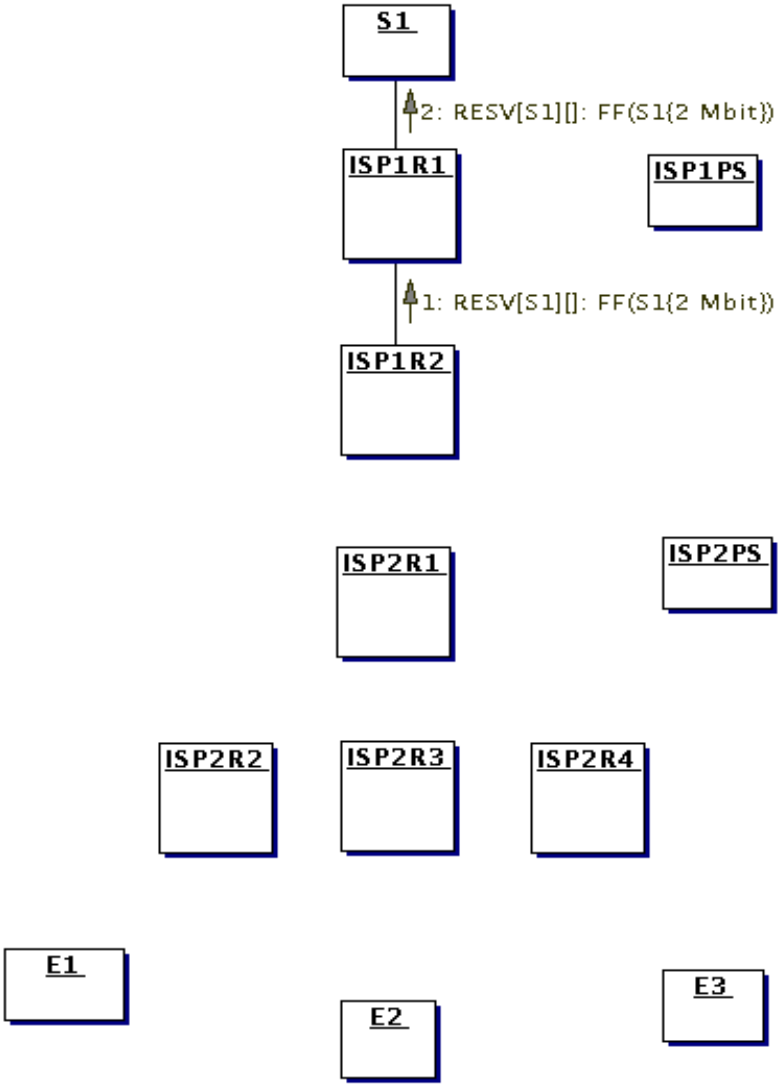
- **Zusammenfassung von RESV-Nachrichten, Authentifizierung bei ISP1**

Die authentifizierten RESV-Nachrichten werden im ISP2R1 zusammengefasst und an ISP1R2 weitergeleitet. Dabei werden alle POLICY_DATA-Objekte aus den ursprünglichen RESV-Nachrichten in die neue eingefügt. Beim Empfang der RESV-Nachricht durch ISP1R2 wird die PATH-Nachricht an den Policy-Server von ISP1 versandt. Da beide Antworten aus der PATH-Nachricht gültig sind, versendet der Policy-Server die Entscheidung, dass alle POLICY_DATA-Objekte gelöscht werden sollen und dass die RESV-Nachricht akzeptiert wird:



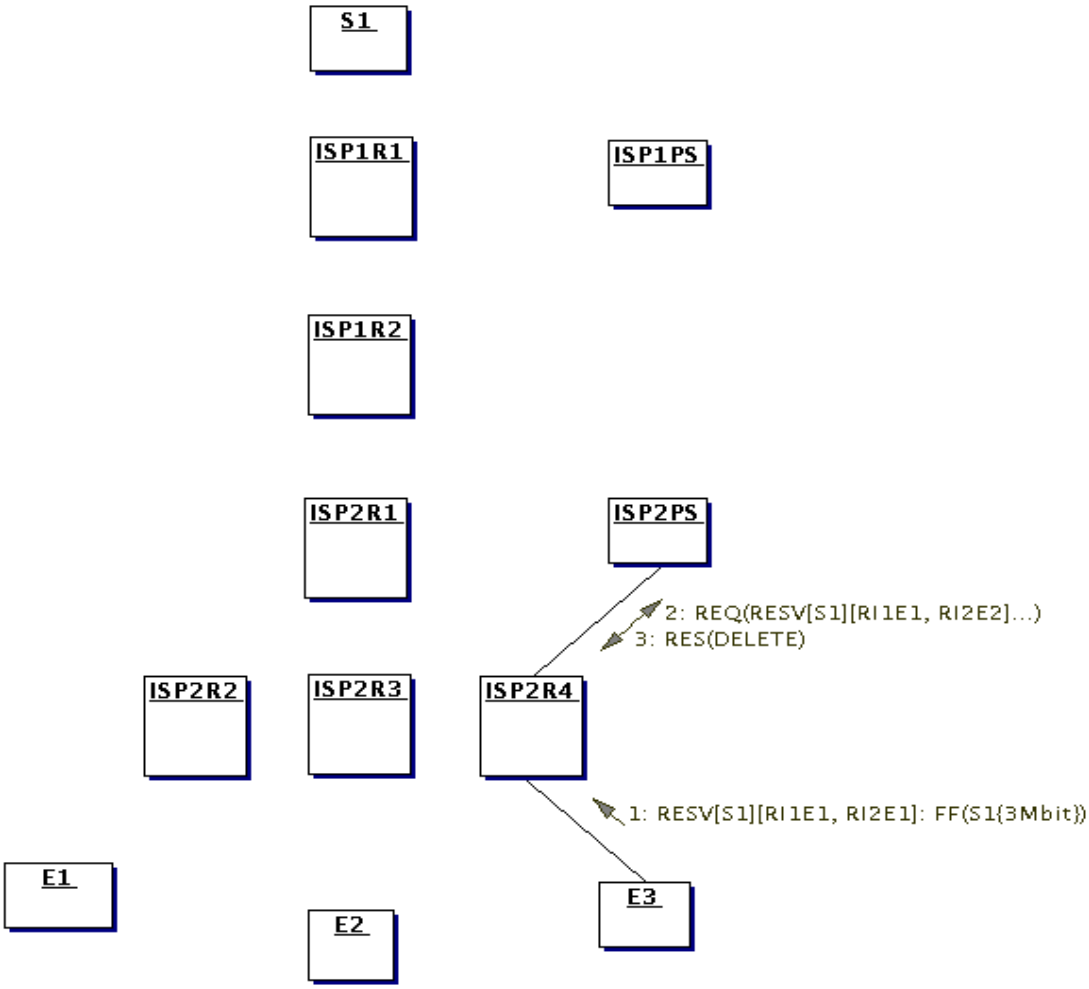
- **Weiterleitung der RESV-Nachricht an S1**

Nach der Authentifizierung der RESV-Nachricht und dem Löschen der POLICY_DATA-Objekte aus der Nachricht wird die RESV-Nachricht an S1 weitergeleitet.



- **Versuch einer Replay-Attacke**

E3 versucht mit dem Versand von Antworten, die von E1 generiert wurden (RI1E1, RI2E1), eine Reservierung zu authentifizieren. Der Versuch scheitert, weil die Antworten von E1 nur auf der Netzwerkschnittstelle von ISP2R2 in Richtung E1 gültig sind.



4 Java Policy Sever

Eines der Ziele der vorliegenden Arbeit war es, eine Implementierung eines Policy-Servers zu erstellen. Für die Implementierung des Policy-Servers wurde die Programmiersprache JAVA[23] gewählt. Alternativ wären die Programmiersprachen in Frage gekommen, die üblicherweise eingesetzt werden, wenn es darum geht, einen Netzwerkservice zu implementieren: C oder C++. Die Gründe für die Wahl von JAVA als Programmiersprache der Implementierung waren:

- Durch die Entwicklung von HOTSPOT-Java-Virtual-Machine [24] hat sich die Geschwindigkeit der Ausführung von JAVA-Programmen sehr an die Geschwindigkeit der Ausführung von C oder C++ Programmen angenähert.
- Im Gegensatz zu C oder C++ bietet JAVA ein sicheres Memory-Modell an. Das bedeutet, dass JAVA-Programme nicht in der Lage sind, die Daten auf dem Heap oder Stack durch Pointer- oder Arraymanipulationen zu überschreiben. Dies führt dazu, dass die Fehlersuche viel einfacher als bei C oder C++ ist.
- JAVA beinhaltet die automatische Freigabe von nicht benutztem Speicher (so genannte Garbage-Collection). Dadurch ist die Speicherverwaltung viel einfacher und die Wahrscheinlichkeit, dass so genannte Memory-Leaks (das Vergessen der Freigabe von nicht mehr benötigtem Speicher) entstehen, deutlich geringer.
- Die Unterstützung für Multithreading und für Synchronisation (Monitore und Conditional-Variablen) sind Teil der Programmiersprache.
- JAVA-Programme sind plattformunabhängig, was die Anwendbarkeit der Implementierung erhöht.
- Es existieren sehr viele Open-Source-Code-Bibliotheken für JAVA, die die Entwicklung des Policy-Servers beschleunigen können.

Im vorliegenden Kapitel werden die Grundlagen der Programmiersprache JAVA und der objektorientierten Programmierung als bekannt vorausgesetzt und daher nicht mehr eigens erläutert.

Die Implementierung des Policy-Servers sollte sehr einfach zu konfigurieren, zu verändern und zu erweitern sein. Deshalb wurde die Entscheidung getroffen, den Policy-Server aus mehreren voneinander getrennten Komponenten zu entwickeln. Jede Komponente implementiert eine oder mehrere Komponentenschnittstellen, in denen die für die anderen Komponenten verfügbaren Methoden definiert sind. Die Kommunikation zwischen den Komponenten erfolgt ausschliesslich durch deren Komponentenschnittstellen. Für die Konfiguration, den Lebenszyklus (Instantiieren, Starten und Stoppen der Komponenten) und die so genannte Composition (die Zusammenfügung der Komponenten) wurde ein so genannter Microkernelcontainer verwendet: Avalon-Phoenix [25].

4.1 Überblick über Avalon-Phoenix

Die Aufgabe eines Microkernelcontainers ist die Ausführung von Applikationen, die aus Komponenten[17] bestehen, und die Bereitstellung von Diensten wie Logging, Security,

17 Beim Avalon-Phoenix werden die Komponenten als „Blocks“ bezeichnet. Siehe hierzu auch im Folgenden, Seite 53.

Konfiguration etc. für die Komponenten. Avalon-Phoenix ist eine Open-Source-Implementierung eines Microkernelcontainers.

Eine Phoenix-Applikation besteht aus den nachfolgend beschriebenen Teilen:

- Eine oder mehrere Komponenten, die im Folgenden als Blocks bezeichnet werden. Jeder Block besteht aus einer Klasse und einer Beschreibung in Form einer XML-Datei.
- Eine Beschreibung der Abhängigkeiten zwischen den Blocks und eine Liste spezieller Klassen (Event-Listeners), die auf Applikations-Events reagieren: „assembly.xml“ – Assembly-Descriptor.
- Eine Konfigurationsdatei, in der sich die Konfiguration aller Blocks befindet: „config.xml“.
- Eine Beschreibung der internen Dienste des Containers wie Logging und Java-Security-Permissions für den Byte-Code: „environment.xml“.

Jede Avalon-Phoenix-Applikation wird dem Container in Form so genannter Server-Application-Archive (SAR) zur Verfügung gestellt. Ein Server-Application-Archiv ist ein spezielles ZIP-Archiv mit folgenden Eingenschaften:

- Die Extension der Daten ist „.sar“ statt „.zip“.
- Alle JAR-Dateien (die Dateien mit dem JAVA-Bytecode) befinden sich im Verzeichnis „SAR-INF/lib/“.
- Alle XML-Deskriptoren der Komponenten (Blocks) in der Avalon-Phoenix-Applikation befinden sich im Verzeichnis „SAR-INF/“.

4.1.1 Die Avalon-Phoenix-Blocks

Die Grundkomponenten von Avalon-Phoenix sind die so genannten Blocks. Ein Block ist im Prinzip eine Klasse, die eine oder mehrere Schnittstellen (Services) implementiert. Jeder Block hat eine Beschreibung (Block-Information) im XML-Format [26]. Die Beschreibung enthält folgende Informationen über den Block :

- den Namen des Blocks (optional);
- die Versionsnummer des Blocks;
- eine Liste der Schnittstellen (Services), die vom Block implementiert werden (optional);
- eine Liste der Schnittstellen (Services), die der Block benötigt (optional);
- eine Liste von Management-Services, die der Block für das Management zur Verfügung stellt (optional).

Eine BlockInfo-Datei könnte etwa folgendermassen aussehen:[18]

```
<xml version="1.0"?>
<blockinfo>
    <block>
         <version>1.2.3</version>
    </block>
```

18 Das Beispiel wurde aus der Webseite des Avalon-Phoenix-Projekts [25] übernommen.

```
    <services>
         <service name="com.biz.cornerstone.services.MyService"
version="2.1.3" />
    </services>

    <dependencies>

         <dependency>

<role>com.biz.cornerstone.services.Authorizer</role>
    <service name="com.biz.cornerstone.service.Authorizer"
version="1.2"/>
         </dependency>

         <dependency>

                   <!-- note that role is not specified and defaults
to name of service. The service version is not specified and it
defaults to "1.0" -->
                   <service
name="com.biz.cornerstone.service.RoleMapper"/>
</dependency>

    </dependencies>

<blockinfo>
```

Eine oder mehrere Block-Klassen werden zusammen mit den Block-Deskriptoren in so genannten JAR-Archiven [27] zusammengefasst.[19] Der Container instantiiert beim Starten einer Applikation alle definierten Blocks und stellt den Blocks alle gewünschten internen Dienste[20] zur Verfügung. Der Zugang zu den internen Diensten wird durch das so genannte Inversion-of-Control-(IoC)-Pattern ermöglicht: ein Block deklariert durch Implementierung der entsprechenden Schnittstelle, dass er einen Dienst braucht. So kann ein Block sein Interesse am Konfigurationsdienst deklarieren, indem er die Schnittstelle „**org.apache.avalon.framework.configuration.Configurable**“ implementiert. Diese Schnittstelle definiert die Methode, die vom Container aufgerufen wird, um dem Block eine Implementierung der Schnittstelle „**org.apache.avalon.framework.configuration.Configuration**“ zu übergeben, durch die der Block den Zugang zur Konfiguration erhält. Der Zugang zu Services, von denen dieser Block abhängig ist, wird auch durch IoC ermöglicht. Wenn ein Block einen Service benötigt, muss er die Schnittstelle „**org.apache.avalon.framework.service.Serviceable**“ implementieren. Wenn ein Block diese Schnittstelle implementiert, stellt der Avalon-Phoenix-Container nach dem Instantiieren des Blocks durch den Aufruf der Methode „**service()**“ dem Block eine Implementierung der Schnittstelle „**org.apache.avalon.framework.service.ServiceManager**“ zur Verfügung. Durch diese Implementierung kann der Block dann die gewünschten Services erhalten.

19 JAR-Archive sind ZIP-Archive, die die Dateien mit dem Java-Bytecode enthalten sowie ein „META-INF“ Verzeichnis, in dem sich die Dateien mit den Metainformationen über den Bytecode befinden.

20 In Avalon-Phoenix wird zwischen zwei Typen von Diensten unterschieden. Dabei handelt es sich einerseits um solche, die den Komponenten vom Container zur Verfügung gestellt werden; diese werden hier und im Folgenden als „interne Dienste“ bezeichnet. Als „Services“ werden demgegenüber diejenigen Dienste bezeichnet, die von den Komponenten selbst implementiert und den anderen Komponenten durch den Service-Manager zur Verfügung gestellt werden.

4.1.2 Assembly-Descriptor

Der Assembly-Descriptor ist eine XML-Datei, deren Inhalt die Namen der Blocks und ihre gegenseitige Abhängigkeit definiert. Im Assembly-Descriptor wird beschrieben, ob und wenn ja welche so genannten Application-Listeners instantiiert werden. Ein Application-Listener ist ein Objekt, welches die Schnittstelle „**org.apache.avalon.phoenix.ApplicationListener**“ implementiert und damit alle applikationsabhängigen Ereignisse empfängt (Start/Stop der Applikation, Instantiieren von Blocks etc.). Der Assembly-Descriptor befindet sich in der Datei „SAR-INF/assembly.xml“ innerhalb der Server-Applikations-Archive (SAR).

Ein Assembly-Descriptor könnte etwa folgendermassen aussehen:[21]

```
<?xml version="1.0"?>
<assembly>

    <block name="myAuthorizer"
class="com.biz.cornerstone.blocks.MyAuthorizer">
    </block>

    <block name="myBlock" class="com.biz.cornerstone.blocks.MyBlock">
         <provide name="myAuthorizer"
role="com.biz.cornerstone.services.Authorizer"/>
    </block>

    <listener name="myListener"
class="com.biz.cornerstone.listeners.MyListener">
    </listener>

</assembly>
```

4.1.3 Konfiguration der Blocks

Für jeden Block, der in „application.xml**“** definiert wurde, muss eine Konfiguration in der „SAR-INF/config.xml“-Datei vorhanden sein. Diese Konfiguration kann vom Block ausgewertet werden, wenn der Block die Schnittstelle „**org.apache.avalon.framework.configuration.Configurable**“ implementiert. Für jeden Block existiert in der Konfigurationsdatei ein XML-Tag unter dem Root-Tag mit dem Namen des Blocks. Das Layout der SubTags wird im Block definiert.

Eine Konfiguration könnte etwa folgendermassen aussehen:

```
<?xml version="1.0"?>
<config>

    <myAuthorizer>
         <!-- ...configuration data here... -->
    </myAuthorizer>

    <myBlock>
         <param1>param1-value</param1>
         <an-integer>2</an-integer>
         ...
    </myBlock>
```

21 Dieses sowie die beiden nachfolgenden Beispiele für Konfiguration und Umgebungsbeschreibung wurden wie schon das vorangegangene aus der Webseite des Avalon-Phoenix-Projekts [25] übernommen.

```
</config>
```

4.1.4 Umgebungsbeschreibung

Die Umgebungsbeschreibung definiert die Konfiguration von Logging und Java-Security-Permissions. Die Umgebungsbeschreibung befindet sich in der Datei „SAR-INF/assembly.xml“ innerhalb der Server-Applikations-Archive (SAR).

Eine Umgebungsbeschreibung könnte etwa folgendermassen aussehen:

```
<?xml version="1.0"?>
<environment>
    <logs>
         <category name="" target="default" priority="DEBUG" />
         <category name="myAuthorizer" target="myAuthorizer-target"
priority="DEBUG" />
         <log-target name="default" location="/logs/default.log" />
         <log-target name="myAuthorizer-target"
location="/logs/authorizer.log" />
    </logs>

    <policy>
         <keystore name="foo-keystore" location="sar:/conf/keystore"
type="JKS" />
         <grant code-base="file:${app.home}${/}some-dir${/}*" key-
store="foo-keystore" >
                  <permission class="java.io.FilePermission"
target="${/}tmp${/}*" action="read,write" />
         </grant>

         <grant signed-by="Bob" code-base="sar:/SAR-INF/lib/*" key-
store="foo-keystore" >
                  <permission class="java.io.FilePermission"
target="${/}tmp${/}*" action="read,write" />
         </grant>
    </policy>
</environment>
```

4.2 Die Komponenten des Policy-Servers

Durch den Einsatz von Avalon-Phoenix wurde die Infrastruktur der Applikation (Composition, Logging, Lifecycles, Configuration) vom Container übernommen, so dass es bei der Implementierung nur noch notwendig war, sich auf die Implementierung der eigentlichen Funktionalität zu konzentrieren.

Der Java-Policy-Server besteht aus folgenden Komponenten (Blocks):

- Core

- ClusteredAuthStateManager
- COPS-Connector
- ChallengeResponseGenerator

Die gegenseitigen Abhängigkeiten der einzelnen Komponenten können Abbildung 12 entnommen werden.

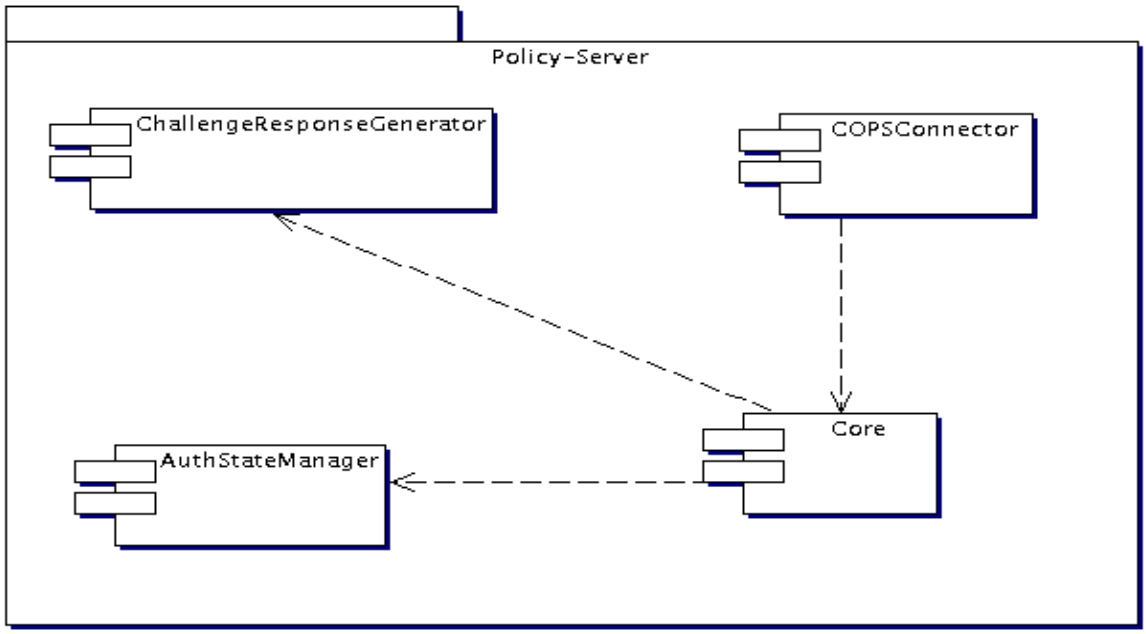


Abbildung 12: UML-Komponentendiagramm des Java Policy Servers. Die Pfeile markieren die Abhängigkeiten einzelner Komponenten voneinander.

Die Funktionen der einzelnen Komponenten (Blocks) des Policy-Servers werden im Folgenden dargestellt.

4.2.1 Core

Die Core-Komponente ist der Kern des Policy-Servers. Im Zusammenspiel mit der Encryption-Komponente implementiert die Core-Komponente die gesamte Logik des Policy-Servers und stellt eine Schnittstelle (Service) für die Benutzung zur Verfügung, die vom Connector (in diesem Fall dem COPS-Connector) verwendet wird.

Core implementiert den folgenden Service (Schnittstelle):

```
package com.crsn.policyserver;
```

```
public interface CoreService {

    public boolean authenticateResponse(String pepId, RouterInterface
iface, Session session, byte[][] responses);

    public byte[] createChallenge(String pepId, RouterInterface iface,
Session session, long timeout);

}
```

Die Core-Komponente benötigt folgende Services:

- **AuthStateManager**: Die Aufgabe von AuthStateManager ist es, den Zustand, das heisst die gültigen Antworten (Responses), so lange zu speichern, bis ihre Lebensdauer abgelaufen ist. Der AuthStateManager-Service hat folgende Schnittstelle:

```
package com.crsn.policyserver;


public interface AuthStateManagerService {

    public boolean responseAcceptable(String pepId, RouterInterface
iface,Session session, byte[] response);

    public void addResponse(String pepId, RouterInterface iface,
Session session, byte[] response, long timeout);

}
```

- **ChallengeReponseGenerator**: Die ChallengeReponseGenerator-Komponente hat die Aufgabe, ein Challenge/Response-Paar zu generieren. Die Implementierung des ChallengeReponseGenerators soll die Verbindung zum Inhaltsanbieter herstellen und entweder nach Bedarf einzelne Challenge/Response-Paare anfordern oder mehrere Paare auf Vorrat herunterladen. Der ChallengeResponseGenerator-Service hat folgende Schnittstelle:

```
package com.crsn.policyserver;

public interface ChallengeResponseGeneratorService {

    public ChallengeResponsePair generateChallengeResponsePair(Session
session);

}
```

4.2.2 ClusteredAuthStateManager

Der AuthStateManager-Service wird von der Core-Komponente benutzt, um den Zustand des Policy-Servers zu speichern. Die ClusteredAuthStateManager-Komponente implementiert diesen Service, indem der Zustand im Speicher gespeichert und gleichzeitig per Multicast an die anderen Instanzen des Policy-Servers repliziert wird (siehe hierzu im Einzelnen Kapitel 4.4 der vorliegenden Arbeit).

4.2.3 COPS-Connector

Die COPS-Connector-Komponente benutzt die JCOPS-Bibliothek, um den COPS-Server für den ClientType 1 (COPS-RSVP) zu implementieren. Die COPS-Connector-Komponente dient als Adapter zwischen dem COPS-Protokoll und dem Core-Service.

Der COPS-Connector implementiert keine Services, er benutzt nur den Core-Service.

4.2.4 DummyChallengeResponseGenerator

Da in der vorliegenden Arbeit das Protokoll für das Einholen von Challenge/Response-Paaren beim Inhaltsanbieter nicht festgelegt ist, wird der ChallengeResponseGenerator-Service durch eine so genannte Dummy-Implementierung nur provisorisch implementiert. Die Implementierung ist in der Lage, anhand eines in der Konfiguration festgelegten Schlüssels (nämlich des Schlüssels für die Entschlüsselung der Videodaten) ein Challenge/Response-Paar zu generieren. Diese Implementierung wird für den Funktionstest des Policy-Servers verwendet. Beim Einsatz des Servers in der Praxis müsste diese Komponente durch eine Implementierung von ChallegeResponseGenerator-Service ersetzt werden, die die Challenge/Response-Paare vom Inhaltsanbieter einholt.

4.3 Die Implementierung des COPS-Protokolls in JAVA (JCOPS)

Obwohl schon eine Implementierung des COPS-Protokolls für Java existiert (JCAPI, vgl. [28]), wurde in der vorliegenden Arbeit eine eigene COPS-Implementierung für JAVA erstellt: JCOPS. Dies hat folgende Gründe:

1. Um JCAPI zu kompilieren, muss Borland J-Builder 3.0 unter Windows 98 verwendet werden.[22]
2. Laut Information auf der Website von JCAPI [28] darf der Code nur von Universitäten und nichtkommerziellen Institutionen verwendet werden.[23]
3. IPV6 wird von JCAPI nicht unterstützt.

4.3.1 Architektur von JCOPS

Um die Implementierung des COPS-Protokolls gut zu strukturieren und das Design einzelner Teile der Implementierung orthogonal zu gestalten, war es notwendig, die Implementierung in mehrere Schichten, so genannte Layers, zu unterteilen. Der Einsatz von Layering beruht auf der Überlegung, dass in jedem Layer ein Aspekt des Protokolls implementiert werden kann, wobei die Layers zugleich unabhängig voneinander sind. Was die Architektur von JCOPS von einer klassischen Layer-Architektur (vgl. [29], Seite 31) unterscheidet, ist die Tatsache, dass die Zahl

22 Der Versuch, den Code unter Linux und JDK 1.4.2 zu kompilieren, ist an verschiedenen Kompilations- und Syntaxfehlern gescheitert.

23 Ob der Code verändert oder als Teil anderer Projekte verwendet werden darf, bleibt dabei unklar.

und Anordnung der Layers nicht fest vorgegeben, sondern dynamisch ist. Darüber hinaus sind die Schnittstellen der Layers identisch, und somit erinnert die Architektur an die „Pipes and Filters“-Architektur (vgl. [29], Seite 53), mit dem Unterschied, dass von den einzelnen Filtern Datenflüsse (die Flüsse der COPS-Nachrichten) in beide Richtungen, also sowohl ein- als auch ausgehend, bearbeitet werden.

4.3.1.1 MessageFilter

Die Hauptkomponente von JCOPS ist der so genannte MessageFilter. Ein MessageFilter ist eine Erweiterung des „Chain of Responsibility“-Design-Patterns (vgl. [30], Seite 223). Im Gegensatz zu diesem Pattern, bei dem Anfragen nur in eine Richtung delegiert werden, ist der MessageFilter bidirektional, das heisst ein MessageFilter kann die eingehenden Nachrichten an den nächsten MessageFilter und die ausgehenden Nachrichten an den vorherigen Message-Filter delegieren.

Der Grund für diese Erweiterung liegt darin, dass die Kommunikation im COPS-Protokoll asynchron ist; durch eine COPS-Verbindung werden mehrere Anfragen (Requests) gleichzeitig bearbeitet. Das bedeutet, dass nicht von einer bestimmten Reihenfolge der eingehenden und ausgehenden Nachrichten ausgegangen werden kann.

Jeder MessageFilter kann eine der folgenden Operationen ausführen:

- **Senden von COPS-Nachrichten**: Jeder Filter kann eine neue COPS-Nachricht versenden.
- **Filtern von eingehenden COPS-Nachrichten**: Falls der Filter eine eingehende COPS-Nachricht erhält, darf er die Nachricht beliebig verändern und nötigenfalls auch verhindern, dass sie den nächsten Filter erreicht.
- **Filtern von ausgehenden COPS-Nachrichten**: Jeder MessageFilter, der eine ausgehende COPS-Nachricht erhält, darf die Nachricht verändern und nötigenfalls deren Weiterleitung an den vorherigen MessageFilter verhindern.
- **Schliessen der COPS-Verbindung**: Jeder MessageFilter darf nötigenfalls jederzeit die COPS-Verbindung schliessen.
- **Empfang von Lifecycle-Events**: Auf jedem MessageFilter wird nach der Öffnung einer COPS-Verbindung die „**start()**“-Methode aufgerufen. Vor der Schliessung der COPS-Verbindung wird auf jedem MessageFilter die „**stop()**“-Methode aufgerufen.

Die folgenden Schnittstellen sind für die Implementierung von Message-Filtern relevant:

```
package com.crsn.policyserver.cops.comm.common;

import java.nio.ByteBuffer;
public interface MessageFilter {

    public void setFilterContext(FilterContext ctx);

    public void handleIncoming(ByteBuffer messageBuffer,
IncomingMessageFilterChain chain);
    public void handleOutgoing(ByteBuffer messageBuffer,
OutgoingMessageFilterChain chain);

    public void start();
    public void stop();
```

```
}
```

```
package com.crsn.policyserver.cops.comm.common;

import java.nio.ByteBuffer;

public interface IncomingMessageFilterChain {

    public void handleIncoming(ByteBuffer messageBuffer);

}
```

```
package com.crsn.policyserver.cops.comm.common;

import java.nio.ByteBuffer;

public interface OutgoingMessageFilterChain {

    public void handleOutgoing(ByteBuffer messageBuffer);

}
```

```
package com.crsn.policyserver.cops.comm.common;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;

public interface FilterContext {

    public void sendMessage(ByteBuffer buffer);

    public void closeConnection();

    public InetSocketAddress getRemoteSocketAddress();

}
```

4.3.1.2 Ein Beispiel für die Verwendung von MessageFilters

Der Lebenszyklus und die Verwendung von Filtern für die Bearbeitung von COPS-Nachrichten werden im Folgenden an einem Beispiel erläutert. In diesem Beispiel wird ein COPS-Server implementiert, der aus zwei Filtern besteht (Filter1 und Filter2). Der Ablauf für die Implementierung des COPS-Clients ist analog zu dem für den COPS-Server.

- **Starten des COPS-Servers**

Zu Beginn wird der COPS-Server gestartet. Dabei werden zuerst die Filter instantiiert, dann die Implementierungen von FilterChains- und FilterContext-Schnittstellen erstellt. Die Implementierungen von FilterContext-Schnittstellen werden den Filtern mittels „**setContext**()“-Methode übergeben, und die Filter werden mittels „**start**()“-Methode gestartet. Der Ablauf der Initialisierung ist in Abbildung 13 zu sehen:

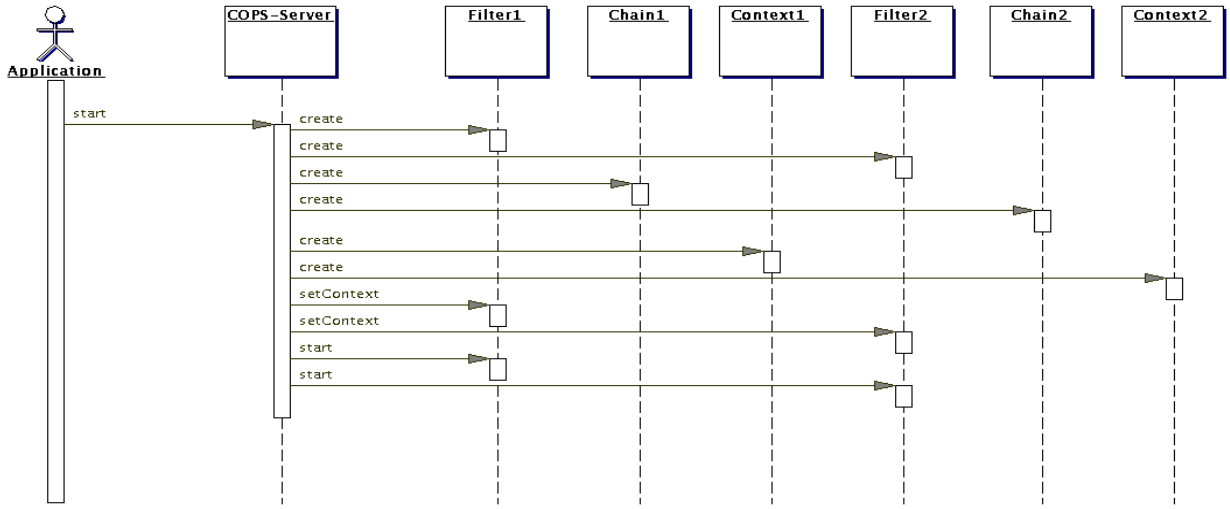


Abbildung 13: Sequenzdiagramm der Initialisierung der Filter.

- **Bearbeiten von eingehenden Nachrichten**

Nachdem die MessageFilters initialisiert wurden, sind sie bereit, die eingehenden Nachrichten zu bearbeiten. Die Bearbeitung der Nachrichten wird vom COPS-Server an die Filter ausgelagert, wobei jeder Filter in der Lage ist, eine COPS-Nachricht beliebig zu verändern und nach Bedarf dann mittels Filter-Chain-Implementierung dem nächsten Filter zu übergeben. Falls ein Filter in der Lage ist, eine Nachricht selbst zu verarbeiten, leitet er die Nachricht nicht an den nächsten Filter weiter. Der letzte Filter ist normalerweise derjenige, der die wichtigen Nachrichten (zum Beispiel REQ-Nachrichten) empfängt und bearbeitet. Jeder Filter ist auch jederzeit in der Lage, eine ausgehende Nachricht zu erzeugen. Das Beispiel in Abbildung 14 zeigt, wie der letzte Filter eine neue COPS-Nachricht versendet, indem er die entsprechende Methode der Implementierung der FilterContext-Schnittstelle aufruft.

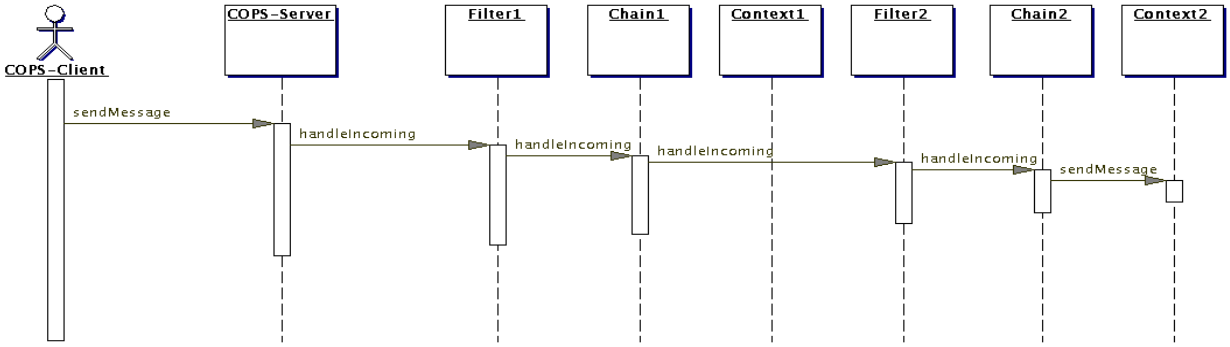


Abbildung 14: Bearbeitung einer eingehenden COPS-Nachricht durch MessageFilters.

- **Bearbeiten von ausgehenden Nachrichten**

Wenn ein Filter mittels seiner Context-Implementierung eine Nachricht versendet, wird die Nachricht durch alle Filter bearbeitet, die sich vor diesem Filter in der Kette befinden. Jeder dieser Filter hat die Möglichkeit, die Nachricht zu verändern und nötigenfalls auch nicht weiterzuleiten. Falls die Nachricht vom ersten Filter weitergeleitet wird, wird sie an den COPS-Client versandt. Einen typischen Ablauf für den Versand von COPS-Nachrichten zeigt Abbildung 15.

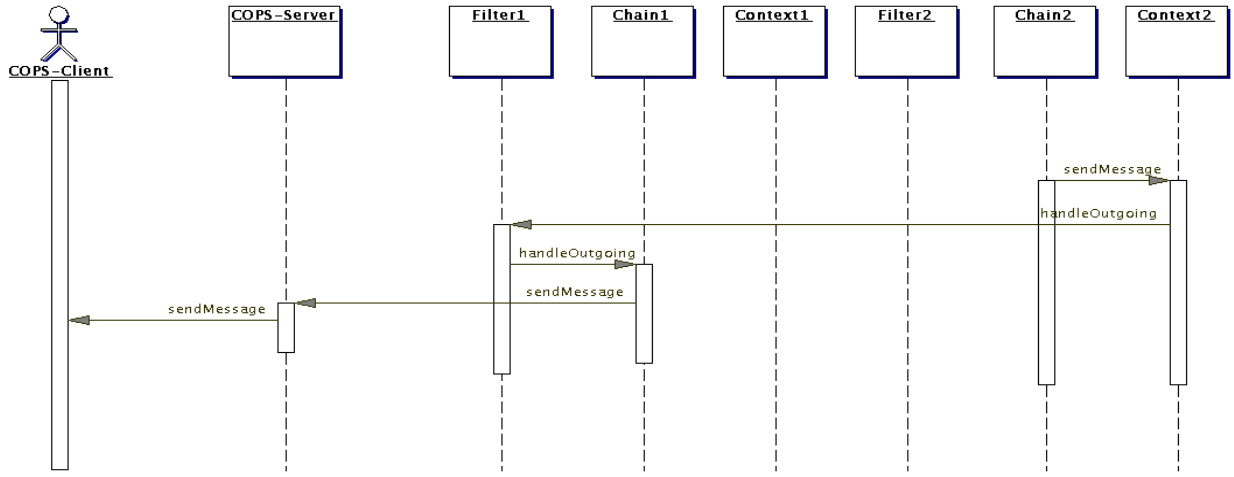


Abbildung 15: Bearbeitung von ausgehenden COPS-Nachrichten durch MessageFilters.

- **Schliessen der COPS-Verbindung durch einen MessageFilter**

Jeder MessageFilter kann jederzeit entscheiden, die COPS-Verbindung zu schliessen. Dies kann er durch Aufruf der „**close**()“-Methode auf der Implementierung von FilterContext erreichen. Letztere wurde dem MessageFilter beim Aufruf der „**setContext**()“-Methode beim Start des COPS-Servers übergeben. Beim Schliessen der Verbindung wird in jedem MessageFilter die Methode „**stop**()“ aufgerufen, um ihm eine Chance zu geben, noch die letzten COPS-Nachrichten versenden zu können. Die Reihenfolge der Aufrufe von „**stop**()“-Methoden ist umgekehrt zur Reihenfoge der Bearbeitung der eingehenden Nachrichten. So wird zuerst der letzte Filter gestoppt, dann der vorletzte etc. Der Ablauf vom Schliessen der COPS-Verbindung ist in Abbildung 16 veranschaulicht.

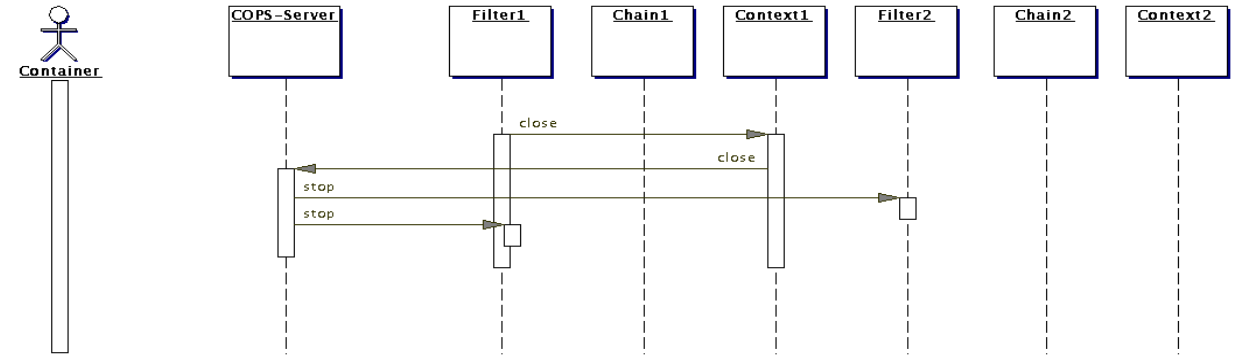


Abbildung 16: Schliessen der COPS-Verbindung durch einen MessageFilter.

- **Stoppen des COPS-Servers**

Beim Stoppen des COPSServers wird praktisch derselbe Ablauf in Gang gesetzt wie beim Schliessen der COPS-Verbindung durch einen Filter. Die „**stop**()“-Methoden auf den Message-Filtern werden in umgekehrter Reihenfolge zur Bearbeitung der eingehenden COPS-Nachrichten aufgerufen, so dass jeder Message-Filter die Chance erhält, die letzten COPS-Nachrichten noch zu versenden. Ein Beispiel für das Stoppen des COPS-Servers ist in Abbildung 17 zu sehen.

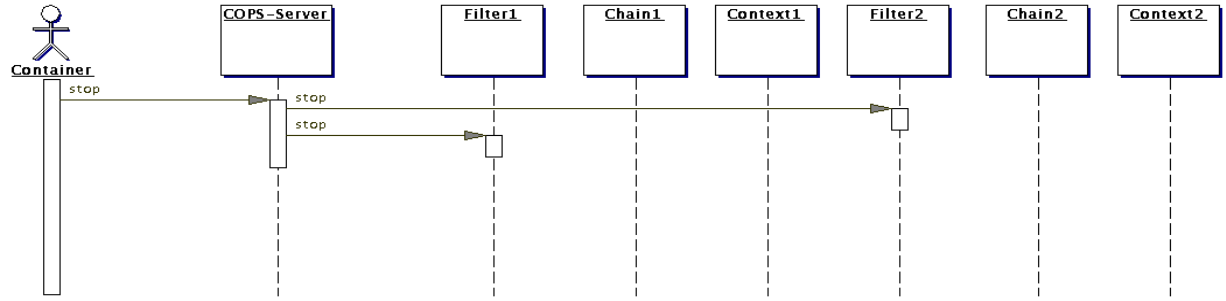


Abbildung 17: Ein Beispiel für das Stoppen des COPS-Servers.

4.3.2 Implementierung von JCOPS

Das Design und die Architektur von JCOPS machen es möglich, das ganze Protokoll in Form von Message-Filtern zu implementieren, so dass jeder Message-Filter einen Aspekt des COPS-Protokolls übernehmen kann. Da sich die Bearbeitung von COPS-Nachrichten im Server und im Client wesentlich unterscheiden, war es notwendig, für den Server und den Client getrennte Implementierungen von Message-Filtern zu erstellen. Die existierenden Filter werden in den folgenden beiden Unterkapiteln kurz vorgestellt.

4.3.2.1 MessageFilter für den COPS-Client

- **ClientIntegrityFilter**: Der Client-Integrity-Filter behandelt die clientseitige Authentifizierung der COPS-Verbindung. Dieser Filter entfernt die Integrity-Objekte aus eingehenden Nachrichten und fügt den ausgehenden COPS-Nachrichten Integrity-Objekte hinzu. Falls die Authentifizierung fehlschlägt, wird die COPS-Verbindung geschlossen.
- **ClientKAFilter**: Der Keep-Alive-Filter implementiert die Keep-Alive-Funktionalität von COPS. Alle eingehenden CAT-Nachrichten werden nach KA-Timer-Objekten durchsucht, um den kleinsten KA-Timer zu ermitteln. KA-Nachichten werden periodisch vom ClientKAFilter versandt, und falls nach dem Ablauf von Timeout keine KA-Nachricht empfangen wird, wird die COPS-Verbindung geschlossen. Die empfangenen KA-Nachrichten werden dann nicht an den nächsten Filter weitergeleitet.
- **ClientSideRequestHandlingFilter**: Der ClientSideRequestHandlingFilter bearbeitet die eingehenden DEC-, CAT-, CC- und SSQ-Nachrichten, indem die Nachrichten an entsprechende COPSClientTypeHandler delegiert werden. Dieser Filter versendet auch die ausgehenden OPN-, REQ-, SSC- und CC-Nachrichten, die vom COPSClientTypeHandler generiert werden.

4.3.2.2 MessageFilter für den COPS-Server

- **ServerIntegrityFilter**: Der ServerIntegrityFilter implementiert die serverseitige Authentifizierung der COPS-Verbindung. Falls die Authentifizierung fehlschlägt, schliesst dieser Filter die COPS-Verbindung.
- **ServerKAFilter**: Der ServerKAFilter überprüft alle ausgehenden CAT-Nachrichten, um den minimalen KA-Timeout festzustellen. Alle eingehenden KA-Nachrichten werden sofort zurückgeschickt (wie in [2] beschrieben). Falls nach dem Ablauf des KA-Timeouts keine KA-Nachricht empfangen wurde, schliesst dieser Filter die COPS-Verbindung.
- **ServerSideRequestHandlingFilter**: Der ServerSideRequestHandlingFilter instantiiert den ClientTypeHandler bei eingehenden OPN-Nachrichten und delegiert REQ-, RPT- und SSC-Nachrichten an die entsprechenden ClientTypeHandler. Ausserdem werden in diesem Filter die DEC-, CC- und SSQ-Nachrichten versandt.

4.3.3 Die Benutzung von JCOPS

Die Benutzung von JCOPS soll so einfach wie möglich sein. Dabei darf kein Aspekt des COPS-Protokolls ausgelassen werden. Deswegen wurde von JCOPS selbst kein Client-Type für das COPS-Protokoll definiert, sondern es wurde ein Gerüst für die Implementierung von beliebigen Client-Types zur Verfügung gestellt. JCOPS stellt sowohl ein Gerüst für die Entwicklung von client- als auch von serverseitigen Implementierungen von COPS-Client-Typen dar.[24]

4.3.3.1 JCOPS-Client

Um eine COPS-Verbindung zum Server herzustellen, muss ein Objekt der Klasse „**com.crsn.policyserver.cops.comm.client.COPSClient**“ instantiiert werden. Im Konstruktor werden PEPID, eine Liste mit allen Adressen des COPS-Servers (falls dieser mehrere alternative Adressen hat) und, falls der Client die Authentifizierung benutzen sollte, eine Implementierung der Schnittstelle „**com.crsn.policyserver.cops.comm.common.filter.IntegrityImplementation**“ übergeben. Mit der Methode „**start()**“ kann eine Verbindung zum COPS-Server hergestellt werden. Danach kann mit der Methode „**registerClient()**“ ein Client-Type-Handler registriert werden.

JCOPS selbst definiert keinen Client-Type-Handler, sondern stellt ein Framework für das Implementieren von Client-Types zur Verfügung. Eine Client-Type-Implementierung muss folgende Schnittstelle implementieren:

```
package com.crsn.policyserver.cops.comm.client;

import com.crsn.policyserver.cops.message.*;
```

24 Einzelheiten zum COPS-Protokoll und den COPS-Client-Typen finden sich auf Seite 18 der vorliegenden Arbeit.

```
public interface COPSClientTypeHandler {

    public void onAccept(ClientAcceptMessage msg);
    public void onSynchronizeStateRequest
(SynchronizeStateRequestMessage msg);
    public void onDecision(DecisionMessage msg);

    public void onClientClose(ClientCloseMessage msg);

    public void setClientMessageSender(ClientMessageSender sender);

    public void start();
    public void stop();
}
```

In der Schnittstelle existieren drei Arten von Methoden:

- **COPS-Message-Methoden**

COPS-Message-Methoden werden beim Eintreffen der entsprechenden COPS-Nachricht aufgerufen. Die Namen dieser Methoden beginnen immer mit „**on**“.

- **Context-Methode**

Die „**setClientMessageSender()**“-Methode übergibt eine Implementierung der folgenden Schnittstelle an den Client-Type-Handler:

```
package com.crsn.policyserver.cops.comm.client;

import com.crsn.policyserver.cops.message.LPDPDecision;
import com.crsn.policyserver.cops.message.object.*;

public interface ClientMessageSender {

    public void sendRequest(ClientHandle handle, int context,
NetworkInterface inInterface, NetworkInterface outInterface,
ClientSpecificInformation[] clientSis, LPDPDecision[] lpdpDecisions);

    public void sendDeleteRequest(ClientHandle handle, int reason);
    public void sendReportState(ClientHandle handle, short reportType,
ClientSpecificInformation clientSi);
    public void sendClientClose(int reason);
    public void sendSynchronizeStateComplete(ClientHandle
clientHandle);


}
```

Beim Aufruf einer Methode aus dieser Schnittstelle wird die entsprechende COPS-Nachricht an den PDP versandt.

- **Lifecycle-Methoden**

Die „**start()**“- und „**stop()**“-Methoden markieren die Zeitspanne, innerhalb derer die Implementierung des Client-Type selbst Nachrichten versenden darf. Die „**start()**“-Methode wird aufgerufen, sobald der PDP eine CAT-Nachricht für den entprechenden Client-Type generiert hat. Die „**stop()**“-Methode wird aufgerufen, nachdem der PDP eine CC-Nachricht versandt hat oder wenn die TCP-Verbindung zum PDP unterbrochen wurde.

4.3.3.2 JCOPS-Server

Ähnlich wie beim Client muss auch bei der Initialisierung eines Servers ein Objekt der Klasse „**com.crsn.policyserver.cops.comm.server.COPSServer**“ instantiiert werden. Im Constructor werden die Socket-Adresse des COPS-Servers, eine Client-Type-Handler-Factory (vgl. [30], Seite 87) für Client-Types, und falls Authentifizierung verlangt wird, eine Implementierung der Schnittstelle „**com.crsn.policyserver.cops.comm.common.filter.IntegrityImplementation**“ übergeben. Mit der „**start()**“- Methode wird der COPS-Server gestartet, mit der „**stop()**“-Methode wird er gestoppt.

4.3.3.3 Client-Type-Handler-Factory

Analog zum COPSClient implementiert der COPSServer keine eigenen Client-Type-Handler. Statt dessen muss für jeden Client-Type ein eigener Client-Type-Handler implementiert werden. Gleichzeitig muss jede Implementierung des Client-Type für jede COPS-Verbindung einmal instantiiert werden. Um dies zu erreichen, wird eine Instanz einer Factory-Klasse übergeben, die die Schnittstelle „**com.crsn.policyserver.cops.comm.server.ClientTypeHandlerFactory**“ implementieren muss.

```
package com.crsn.policyserver.cops.comm.server;

import com.crsn.policyserver.cops.comm.common.*;
import com.crsn.policyserver.cops.message.ClientOpenMessage;

public interface ClientTypeHandlerFactory {
    public ClientTypeAccept open(DecisionSender sender,
ClientOpenMessage msg) throws ClientNotAcceptedException,
RedirectClientException;
}
```

Beim Eintreffen einer OPN-Nachricht wird vom COPS-Server die Methode „**open()**“ aus dieser Schnittstelle aufgerufen. Die Implementierung der ClientTypeHandlerFactory hat dann folgende Möglichkeiten:

- Sie kann eine Instanz der Klasse „**com.crsn.policyserver.cops.comm.server.ClientTypeAccept**“ zurückgeben, falls der Client-Type unterstützt wird.
- Sie kann eine Exception der Klasse „**com.crsn.policyserver.cops.comm.server.RedirectClientException**“ zurückgeben, falls der COPS-Client umgeleitet werden soll.
- Sie kann eine Exception der Klasse „**com.crsn.policyserver.cops.comm.server.ClientNotAcceptedException**“ werfen, falls der Client-Type nicht unterstützt wird.

4.3.3.4 *ClientTypeAccept-Objekt*

Falls die Implementierung der ClientTypeHandlerFactory den Client-Type akzeptiert, wird in der „**open**()“-Methode ein Objekt der Klasse „**com.crsn.policyserver.cops.comm.server.ClientTypeAccept**“ zurückgegeben. Dieses Objekt kapselt „**com.crsn.policyserver.cops.comm.server.ClientTypeHandler**“.

4.3.3.5 *ClientTypeHandler*

Die Schnittstelle „**com.crsn.policyserver.cops.comm.server.ClientTypeHandler**“ hat folgende Signatur:

```
package com.crsn.policyserver.cops.comm.server;

import com.crsn.policyserver.cops.message.*;

public interface ClientTypeHandler {

    public void onReportState(ReportStateMessage msg);
    public void onSynchronizeStateComplete
(SynchronizeStateCompleteMessage msg);
    public void onRequest(RequestMessage msg);
    public void onDeleteRequestState(DeleteRequestStateMessage msg);
    public void close();

}
```

Die Funktionaltät der Methoden ist ähnlich wie im COPSClientTypeHandler. Dabei wurden aber folgende Änderungen vorgenommen:

- Da bei jeder OPN-Nachricht eine neue Instanz der ClientTypeHandler-Implementierung erstellt wird, gibt es keine „**start**()“-Methode in der Schnittstelle (der Konstruktor der Implementierung ist praktisch zugleich die „**start**()“-Methode)
- Die Implementierung von „DecisionSender“ wird nicht explizit durch diese Schnittstelle gesetzt, sondern wird der Factory übergeben, welche die Implementierung instantiiert. Der genaue Mechanismus für das Setzen von „DecisionSender“ ist dann dem Aufbau von Factory überlassen.

4.3.3.6 *DecisionSender*

Die Implementierung von ClientTypeHandler sollte Zugriff auf eine Implementierung der Schnittstelle „**com.crsn.policyserver.cops.comm.server. DecisionSender**“ haben. Die Implementierung wird der Client-Type-Handler-Factory übergeben. Die Schnittstelle „DecisionSender“ hat folgende Methoden:

```
package com.crsn.policyserver.cops.comm.server;

import com.crsn.policyserver.cops.message.Decision;
import com.crsn.policyserver.cops.message.object.ClientHandle;
```

```
public interface DecisionSender {
    public void sendSynchronizeStateRequest(ClientHandle handle);
    public void sendDecision(boolean solicited, ClientHandle handle,
Decision[] decisions);
    public void sendDecisionError(boolean solicited, ClientHandle
handle, int error);
}
```

Dadurch wird das Versenden von Nachrichten ermöglicht, die eine PDP-ClientType-Implementierung auch versenden darf.

4.3.4 Eigenschaften von JCOPS

Bei der Implementierung des COPS-Protokolls, die im Rahmen dieser Diplomarbeit entstanden ist, wurden besonders die im Folgenden beschriebenen Aspekte berücksichtigt.

- **Hohe Leistungsfähigkeit**

Beim Design der JCOPS-Komponenten wurde sehr darauf geachtet, dass so wenig wie möglich zusätzlicher Overhead bei der Bearbeitung der COPS-Nachrichten eingeführt wird. Ausserdem wird in der Implementierung die neue Anwendungsprogrammschnittstelle (API) I/O (JAVA NIO) benutzt, die einen direkten Zugriff auf Speicher-Byte-Buffers und deren Übertragung durch Sockets ermöglicht. Auf jeder Stufe der Implementierung wurden zudem Leistungsmessungen durchgeführt, um die Auswirkungen der Design-Entscheidungen auf die Leistungsfähigkeit zu überprüfen. Gleichzeitig wurde die Generierung von neuen Objekten in kritischen Abschnitten des Codes auf ein Minimum reduziert, um mögliche Unterbrechungen der Bearbeitung durch den Garbage-Collector der Java-Virtual-Machine (JVM) zu minimieren.

- **Einfache Erweiterbarkeit**

Das zentrale Design von JCOPS stützt sich auf das „Chain of Responsibility“-Design-Pattern ([30], Seite 223) und ermöglicht dadurch Layering sowie die Trennung der verschiedenen Aspekte des COPS-Protokolls (Sicherheit, Überprüfung der Verbindung, Bearbeitung von Requests etc.). Auf diese Weise ist es sehr einfach, die Implementierung des COPS-Protokolls zu erweitern und an künftige Versionen der COPS-Spezifikation anzupassen.

- **Plattform-Unabhängigkeit**

JCOPS wurde als plattformunabhängige Applikation entwickelt. Es sollte unter jedem Betriebssystem ausgeführt werden können, das JAVA Virtual Machine 1.4 oder höher unterstützt.

- **Open-Source**

JCOPS wird unter der LGPL-Lizenz [31] publiziert. Dies ermöglicht den Einsatz von JCOPS in kommerziellen Produkten bei gleichzeitiger Verpflichtung, jede Änderung von JCOPS selbst als Open-Source zu veröffentlichen.

4.4 Clustering von Java Policy Server: HA und HP Clustering

Um den Policy-Server horizontal skalierbar (erweiterbar durch Hinzufügen neuer Rechner) und ausfallsicher zu machen, ergab sich die Notwendigkeit, den Policy-Server clusterfähig zu entwerfen. Der Gedanke, der hinter dem Clustering steht, ist der folgende:

Der Policy-Server hat einen Datenzustand, in diesem Fall eine Menge von Antworten (Responses), die von verschiedenen Ausgangspunkten des ISP auf entsprechende Anforderungen (Challenges) hin versandt wurden, und die zugehörigen Verfallszeiten dafür. Wenn man diesen Zustand auf mehrere COPS-Server repliziert, kann man

1. beim Ausfall eines Rechners den Service einfach durch ein so genanntes Reconnect vom Client auf einen anderen COPS-Server aufrecht erhalten (das heisst, es entsteht kein Datenverlust);
2. die Last auf mehrere Server verteilen und somit horizontale Skalierbarkeit erreichen.

4.4.1 Clustering mit Multicast und Einsatz von JGROUPS

Um den Datenzustand zwischen mehreren Instanzen des Policy-Servers zu synchronisieren, wurde Datenreplikation durch Multicast eingesetzt. Multicast wurde gewählt, weil es genau die Verbreitung der Nachrichten an mehrere Empfänger, die für die Zustand-Replizierung notwendig ist, zur Verfügung stellt.

Der Nachteil von Multicast ist, dass es für die versandten Pakete keine Auslieferungsgarantien gibt. Dafür muss eine Schicht über Multicast geschaffen werden, die die Auslieferung der Pakete überprüft und gegebenenfalls wiederholt.

Ein weiteres Problem der Datenreplikation ist, dass jede neue Instanz des Policy-Servers, die dem Cluster beitritt, den schon vorhandenen Zustand erhalten muss.

Glücklicherweise existiert für alle oben beschriebenen Probleme der Datenreplikation durch Multicast schon eine Lösung: die JGROUPS-Bibliothek [32]. JGROUPS bietet unter anderem eine Abstraktion über der Multicast-Paket-Kommunikation, die eine garantierte Auslieferung von Nachrichten durch Multicast gewährleistet. Weiter bietet JGROUPS die Möglichkeit, den Zustand der Gruppe auf neue Gruppenmitglieder zu transferieren, und verfügt über weitere Optionen wie die Benachrichtigung beim Beitritt und Austritt von Mitgliedern zu/aus einer Gruppe etc.

4.4.2 Weitere Replikationsprobleme

Mit dem Einsatz von JGROUPS sind natürlich noch nicht alle Clustering-Probleme gelöst. Probleme, die durch JGROUPS nicht gelöst werden, sind das Locking von Daten und die Zeitunterschiede zwischen den Rechnern im Cluster.

- **Sperren (Locking) von Daten**

Es gibt verschiedene Möglichkeiten, den Datenzustand in mehreren Instanzen einer Applikation gleich zu halten. Eine Möglichkeit wäre, den gesamten Zustand (State) oder auch nur die Zustandsänderungen aus einer Cluster-Instanz bei Bedarf an die anderen Instanzen zu verteilen. Das Problem dabei ist, dass zwei Instanzen im Cluster den gleichen Teil des Zustands ändern können, wodurch ein Konflikt der Zustandsänderungen entsteht. Um diesen Konflikt zu lösen,

muss man einen Sperrmechanismus benutzen, der sicherstellt, dass Daten zu einem gegebenen Zeitpunkt nur von einer Instanz im Cluster verändert werden können.

Ein anderer Ansatz, der in diesem Projekt verwendet wurde, besteht darin, nicht den Zustand (State) selbst zu replizieren, sondern die Ereignisse (die Methodenaufrufe), die den Zustand verändern. Dieser Ansatz ist auch als „Hot-Replication“ bekannt. Dabei werden alle Methodenaufrufe, die neue Zustände erzeugen, als Cluster-Messages an alle Instanzen verteilt. Damit hat jede Instanz im Cluster denselben Zustand, und es ist nicht notwendig, das Sperren der Zustände im Cluster zu implementieren. Diese Lösung ist nur deshalb möglich, weil die Objekte, die die Zustände ausmachen, nicht explizit gelöscht werden können, sondern eine gewisse Lebensdauer haben und danach automatisch nicht mehr gültig sind.

- **Zeitunterschiede zwischen Rechnern im Cluster**

Ein weiteres Problem, das mit der Datenreplikation zusammenhängt, sind die Zeitunterschiede zwischen Rechnern, die sich im Cluster befinden. Das Problem besteht darin, dass der Zustand des Policy-Servers unter anderem die Verfallszeiten von Objekten (Responses) beinhaltet. Das wäre natürlich nicht von Bedeutung, wenn die Uhren auf allen Rechnern im Cluster synchron wären. Da man davon aber nicht ausgehen kann, muss man dafür sorgen, dassbei der Replikation von Daten nicht fixe Zeitpunkte übertragen werden, sondern die Zeitdifferenz relativ zur lokalen Zeit auf dem Rechner, der die Daten versendet. So kann der empfangende Rechner den absoluten Zeitpunkt ausrechnen, indem er die Zeitdifferenz zur lokalen Zeit addiert. Dabei entsteht durch die Zeit, die die Datenübertragung selbst in Anspruch nimmt, eine gewisse Ungenauigkeit, die man aber durchaus in Kauf nehmen kann, da die potentiellen Zeitunterschiede zwischen den lokalen Uhren auf den Rechnern wesentlich grösser sein können. Vorausgesetzt wird dabei, dass sich alle Cluster-Rechner im Hinblick auf die Datenübertragungsverzögerung in grosser „Nähe“ zueinander befinden und die Übertragungszeiten daher sehr gering sind.

4.5 Leistungsmessungen des Java Policy Servers

Um die Belastbarkeit des Policy-Servers zu überprüfen, wurden zwei Leistungsmessungen durchgeführt: Die Leistungsmessung der COPS-Implementierung (JCOPS) und die Leistungsmessung des Policy-Servers selbst (Java Policy Server). Die Leistungsmessungen von JCOPS wurden durchgeführt, um den Overhead festzustellen, der durch den Einsatz von COPS-Protokoll entsteht, und dann die Ergebnisse der Leistungsmessungen des Java-Policy-Servers beurteilen zu können. Beide Leistungsmessungen wurden auf demselben Rechner durchgeführt, so dass der Overhead der Netzwerkkommunikation minimiert wurde.[25] Die Messungenwurden so durchgeführt, dass es für Server und Client immer jeweils einen eigenen Java-Virtual-Machine-Prozess gab. Der Grund dafür liegt darin, dass die Messungen realitätsnah gestaltet werden sollten – es ist selten der Fall, dass sowohl der Client als auch der Server innerhalb derselben Java-Virtual-Machine ausgeführt werden.

25 Die IP-Kommunikation erfolgt ausschliesslich durch die Loopback-Netzwerkschnittstelle, so dass der Overhead für das Versenden von Paketen durch Netzwerkkartentreiber und Hardware entfällt.

4.5.1 Faktoren, die einen Einfluss auf die Messergebnisse haben

Es existieren zwei Faktoren, die einen Einfluss auf die Ausführungszeit von JAVA-Programmen haben, auf die jedoch umgekehrt die Programme selbst nur bedingt Einfluss nehmen können: Hot-Spot Java Virtual Machine und Garbage-Collection.

4.5.1.1 HotSpot Java Virtual Machine

JAVA-Programme werden zwar kompiliert, aber das Ergebnis der Kompilation ist nicht der Maschinencode für eine Rechnerplattform, sondern ein maschinenunabhängiger Bytecode. In den ersten Versionen der Java-Virtual-Machine wurde dieser Bytecode von der Java-Virtual-Machine interpretiert. Dies hat dazu geführt, dass die Ausführung der JAVA-Programme im Vergleich zu kompiliertem C oder C++ Code sehr langsam war. In der Versionen 1.2.x von Java-Virtual-Machines von SUN wurde ein anderer Ansatz der Ausführung gewählt: Just-In-Time(JIT)-Compiler. Bei diesem Ansatz wurde der JAVA-Bytecode nicht mehr interpretiert, sondern der Code wurde während der Ausführung in den Maschinencode übersetzt und dann als Maschinencode ausgeführt. Dieser Ansatz hat zu erheblicher Beschleunigung der Ausführung von JAVA-Programmmen geführt. Der Nachteil dieses Ansatzes war der Speicherverbrauch, der durch das Speichern des Maschinencodes entsteht. Da der Arbeitsspeicher immer begrenzt ist, konnte nur ein Teil des Bytecodes gespeichert werden; wurde ein anderer Teil des Bytecodes benötigt, so musste die Ausführung des Programms gestoppt werden, bis der JIT-Compiler den JAVA-Bytecode in den Maschinencode übersetzt hatte. Um dieses Problem zu umgehen, wurde die derzeitige Technologie eingeführt: die HotSpot-Ausführung von JAVA-Bytecode. Bei der HotSpot-Ausführung von JAVA-Bytecode wird am Anfang der Bytecode interpretiert und die Java-Virtual-Machine führt Buch darüber, wie oft ein Teil des Codes ausgeführt wird. Wenn die Zahl der Ausführungen eines Codeabschnitts eine Grenze überschreitet, wird dieser Codeabschnitt zu einem so genannten HotSpot.[26] erklärt und in den Machinencode übersetzt. Dadurch wird die Ausführung der kritischen Teile des JAVA-Bytecodes erheblich beschleunigt und zugleich der Speicherbedarf gesenkt.

Durch den Einsatz von HotSpot-Java-Virtual-Machine können die Messergebnisse insofern beeinflusst werden, als die Ausführungszeit zu Beginn der Messungen grösser ist, da am Anfang JAVA-Bytecode interpretiert wird.

4.5.1.2 Garbage-Collection

Ein weiterer Faktor, der die Ergebnisse der Leistungsmessungen beinflussen kann, ist die Speicherverwaltung beziehungsweise die so genannte Garbage-Collection von Java-Virtual-Machine.

Im Gegensatz zu Programmiersprachen wie C oder C++, wo der nicht mehr benötigte Speicher explizit freigegeben werden muss, wird in JAVA ein so genannter Garbagecollector eingesetzt. Der Einsatz von Garbage-Collection in JAVA wird durch folgende Eigenschaften der Programmiersprache ermöglicht:

- Der Speicher wird nur durch JAVA-Objekte belegt.

26 Als HotSpot bezeichnet man diejenigen Teile eines Programms, die für das Laufzeitverhalten ausschlaggebend sind. Normalerweise handelt sich dabei um Abschnitte, die besonders oft durchlaufen werden.

- In JAVA existieren keine Zeiger (Pointers); der Zugang zu Objekten ist nur durch Objektreferenzen möglich.
- Arrays sind in JAVA ebenfalls Objekte. Sie können weder erweitert noch verkleinert werden. Der Zugriff auf die Werte in einem Array wird durch Java-Virtual-Machine überprüft, so dass im Gegensatz zu C oder C++ ein Zugriff auf Werte, die sich ausserhalb des Arrays befinden, nicht möglich ist.

Auf Grund dieser Restriktionen ist es der Java-Virtual-Machine möglich zu erkennen, welche Objekte nicht mehr benötigt werden – also welche Objekte nicht mehr von anderen Objekten referenziert werden – , so dass sie freigegeben werden können. Es existieren mehrere Algorithmen, wie die Garbage-Collection, also die Erkennung und die Freigabe von nichtreferenzierten Objekten, implementiert werden kann. Bei der Entwicklung von JAVA wurden in JAVA-Virtual-Machines unterschiedliche Algorithmen für die Garbage-Collection eingesetzt. Das allen Methoden Gemeinsame ist die Tatsache, dass Garbage-Collection die Ausführung eines Programms unterbrechen kann, wenn zusätzlicher Speicherplatz benötigt wird. Der Zeitpunkt der Garbage-Collection kann nicht vom Programm selbst bestimmt werden; die einzige Möglichkeit, Einfluss auf die Garbage-Collection zu nehmen, ist der Aufruf der „**java.lang.System.gc()**“-Methode. Wenn diese Methode aufgerufen wird, erhält die Java-Virtual-Machine den Tipp, dass jetzt ein guter Zeitpunkt für die Garbage-Collection wäre. Laut Spezifikation ist die Java-Virtual-Machine aber nicht verpflichtet, nach dem Aufruf dieser Methode auch wirklich eine Garbage-Collection durchzuführen.

Die Folge des Einsatzes der Garbage-Collection ist, dass bei der Ausführung von JAVA-Programmen Verzögerungen entstehen können, die zu Abweichungen der Messergebnisse führen. Je mehr Objekte im Java-Code, für den Leistungsmessungen durchgeführt werden, bei jedem Durchlauf generiert werden, desto grösser ist die mögliche Abweichung der Ergebnisse.

4.5.2 Testrechner

Die Eigenschaften des Testrechners sind:

- Betriebssystem: Linux Redhat 9
- Kernel: 2.4.20 (Redhat Posix Threads)
- CPU (Informationen aus /proc/cpuinfo)
    - vendor_id : AuthenticAMD
    - cpu family : 6
    - model : 8
    - model name : AMD Athlon(TM) XP2400+
    - stepping : 1
    - cpu MHz : 2014.579
    - cache size : 256 KB

    - fdiv_bug : no
    - hlt_bug : no
    - f00f_bug : no
    - coma_bug : no
    - fpu : yes
    - fpu_exception : yes
    - cpuid level : 1
    - wp : yes
    - flags : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat pse36 mmx fxsr sse syscall mmxext 3dnowext 3dnow
    - bogomips : 4010.80
- RAM: 1024 MB
- JDK 1.4.2 (SUN)

4.5.3 Leistungsmessungen

Um die Effizienz der Implementierung zu testen, wurden zunächst zwei Reihen von Leistungsmessungen von JCOPS durchgeführt: eine Reihe mit und eine Reihe ohne Authentifizierung.

Das Ergebnis jeder Leistungsmessung beinhaltet die Zeit, die eine Schleife mit 100000 Anfragen (Requests) auf der Client-Seite braucht, um ausgeführt zu werden. Dabei wurde für jede Anfrage (Request) folgende Sequenz von Nachrichten erzeugt: (REQ (PEP->PDP), DEC (PDP->PEP), RPT(PEP->PDP), DRQ (PEP->PDP)). Die Zeitmessung beginnt mit der Entsendung der ersten REQ-Nachricht und endet mit der Entsendung der letzten (im vorliegenden Fall also der 100000.) DRQ-Nachricht. Die Nachrichten wurden dabei nicht paralellisiert, sondern serialisiert versandt.

Der Ablauf des Austauschs der COPS-Nachrichten für eine einzelne Anfrage (Request) ist Abbildung 18 zu entnehmen.

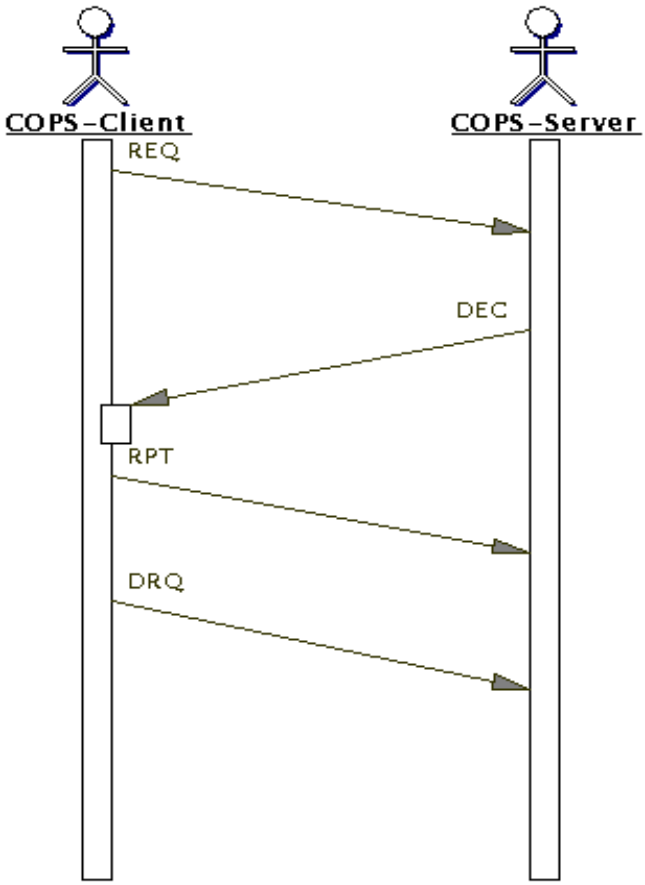


Abbildung 18: Ausgetauschte COPS-Nachrichten für eine einzelne Anfrage (Request) aus der Leistungsmessungen.

Die Ergebnisse der Leistungsmessungen finden sich in Tabelle 1.

Messung	Zeit für 100000 Anfragen mit Authentifikation	Zeit für 100000 Anfragen ohne Authentifikation
1	32490	13573
2	31521	12029
3	31533	12001
4	31922	12344
5	31548	11990
6	31470	11961
7	31441	11984
8	31527	11923
9	31444	12003
10	31478	12009
Mittelw.	31637.4	12181.7
Std. Abw.	330.31	502.4

Tabelle 1: Messegebnisse der Leistungsmessungen von JCOPS (in msec)

Die durchschnittliche Bearbeitungszeit für die COPS-Requests mit JCOPS ist:

- für eine nicht authentifizierte COPS-Verbindung: 0.122 msec/Anfrage(Request)
- für eine authentifizierte COPS-Verbindung: 0.316 msec/Anfrage(Request).

Die durchschnittliche Bearbeitungszeit für eine Anfrage (Request) ist bei Verwendung der COPS-Authentifizierung fast dreifach so gross wie ohne Authentifizierung. Der Grund dafür ist der Overhead der Berechnung der MD5-Prüfsumme.[27]

JCOPS ist in der Lage, ohne COPS-Authentifizierung 8209 Requests pro Sekunde zu bearbeiten.

Im Anschluss an die im Vorigen geschilderten Messungen wurden nunmehr Leistungsmessungen des Policy-Servers durchgeführt.

Da es unter realen Bedingungen sehr schwer ist, eine Implementierung der Authentifizierung von RSVP auf Leistungsfähigkeit zu untersuchen (RSVP-Nachrichten werden nur periodisch und nicht ständig generiert), war es für die Leistungsmessungen notwendig, eine Simulation durchzuführen. Die Last für den Policy-Server wurde von einem COPS-Client erzeugt, der ohne Pausen 10000 PATH-Nachrichten für die Authentifizierung (genauer: für das Generieren von Anforderungen [Challenges]) an den Policy-Server versandt hat. Die Zeitmessung begann jeweils mit der Entsendung der ersten Anfrage mit einer PATH-Nachricht und endet mit dem Empfang der letzten (im vorliegenden Fall also der jeweils 10000.) Entscheidung des Policy-Servers. Die Nachrichten wurden dabei nicht paralellisiert, sondern serialisiert versandt. Die Kommunikation zwischen Client und Server erfolgte über das COPS-Protokoll; sowohl der Client als auch der Server verwendeten dabei die JCOPS-Bibliothek ohne COPS-Authentifizierung.

Die Ergebnisse von 10 Messungen für je 10000 PATH-Nachrichten finden sich in der Tabelle:

27 Es wurde die Default-Implementierung von MD5 von SUN verwendet.

Messung	Zeit für 10000 Path-Nachrichten
1	29658
2	41786
3	35802
4	38015
5	34780
6	38784
7	36276
8	37955
9	37816
10	35319
Mittelw.	36619.1
Std. Abw.	3177.4

Tabelle 2: Messergebnisse der Leistungsmessungen für Java Policy Server (in msec)

Aus den Messergebnissen kann man eine durchschnittliche Bearbeitungszeit von 3.67 msec pro Anfrage (Request) ableiten, die zum Erzeugen einer Anforderung (Challenge) benötigt wird. Die deutlichen Unterschiede bei den ersten drei Messungen lassen sich damit erklären, dass bei Messung (1) ein leerer Speicher vorliegt, so dass der Garbage-Collector nicht tätig wird. Bei Messung (2) hingegen liegt eindeutig eine Verzögerung durch Garbage-Collection vor. Das im Vergleich hierzu wieder bessere Messergebnis aus Messung (3) kann darauf zurückgeführt werden, dass nun die kritischen Stellen des Codes durch die HotSpot-Ausführung von JAVA Virtual Machine in Maschinencode umgewandelt und dadurch die Ausführung beschleunigt wurden. Die – deutlich kleineren – Unterschiede zwischen den Ergebnissen der Messungen (3) bis (10) sind auf die in unregelmässigen Abständen erfolgende Garbage-Collection zurückzuführen.

5 Implementierung des Authentifizierungsschemas unter Linux

Da der Java Policy Server nur eine Implementierung eines Policy-Servers ist, der das in dieser Arbeit entwickelte Authentifizierungsschema auf der Serverseite erzwingen kann, war es für einen Test des Authentifizierungsschemas unter realen Bedingungen[28] notwendig, die Router und Empfänger an das Authentifizierungsschema anzupassen. Um dies zu erreichen, mussten die Implementierungen selbst angepasst werden. Glücklicherweise kann das Authentifizierungsschema unter Linux dank dem so genannten Packet-Filtering (IPTABLES) und NETLINK, die von Linux-Kernels ab Version 2.4 unterstützt werden, sowohl für einen RSVP-Router als auch für eine RSVP-Empfängerapplikation transparent, das heisst ohne Änderung der Applikation, durchgeführt werden. Zur Beschreibung der Vorgehensweise, die dies ermöglicht, müssen zunächst die Eigenschaften der Paketverarbeitung im Kernel von Linux näher betrachtet werden.

5.1 Paket-Filter (Packet-Filtering) in Linux-Kernels ab Version 2.4 (IPTABLES)

Da Linux auch als Betriebssystem für Router verwendet wird, muss es ihm möglich sein, Entscheidungen über eingehende, ausgehende und weiterzuleitende IP-Pakete zu treffen, bevor sie von Benutzerprozessen empfangen und bearbeitet werden. Mit anderen Worten: die Pakete müssen gefiltert werden. Das Filtern der Pakate ermöglicht es, den Sicherheitsschutz des Rechners selbst oder – im Fall der Weiterleitung von Paketen, der auftritt, wenn Linux als Betriebssystem für Router eingesetzt wird – auch für ein ganzes Subnetz zu gewährleisten. Letzteres ist beispielsweise der Fall, wenn der Router als Firewall zwischen dem Internet und dem lokalen Netz eingesetzt wird. Diese Funktionen werden unter Linux durch die so genannten IPTABLES angeboten. IPTABLES bilden ein System, das in der Lage ist, für ein beliebiges IP-Paket zu entscheiden, was mit ihm geschehen soll.

Die komplette Beschreibung, was mit Paketen geschehen soll, wird bei IPTABLES als eine Liste (oder Tabelle) von Regeln dargestellt. Eine Regel besteht dabei jeweils aus einem Filter und einer Aktion. Ein Filter ist eine Funktion, die ein IP-Paket als Parameter bekommt und als Ergebnis einen booleschen Wert (true/false) zurückgibt. Eine Aktion beschreibt, was mit einem Paket, das durch den Filter selektiert[29] wird, geschehen soll. Die wichtigsten Aktionen sind:

- Anwendung einer angegebenen Tabelle - die Entscheidung über das Paket wird an eine andere Tabelle ausgelagert;
- DROP – das Paket wird fallengelassen beziehungsweise wird nicht weiterverarbeitet;
- REJECT – das Paket wird fallengelassen, aber es wird ein ICMP-„port unreachable“- Paket an den Absender versandt;
- ACCEPT - das Paket wird für die Weiterverarbeitung akzeptiert.

28 „Unter realen Bedingungen“ bedeutet in diesem Zusammenhang einen Test mit realen Sendern und Empfängern.

29 Ein Paket ist selektiert, falls die Filter-Funktion den Wert „true“ zurückgibt.

Jede Liste (Tabelle) hat auch eine Default-Aktion, die angewendet wird, falls keiner der Filter ein Paket selektiert hat. Die Entscheidung, was mit einem Paket geschehen soll, wird anhand des folgenden Algorithmus getroffen:

1. Zu Beginn ist die aktuelle Regel die erste Regel aus der Liste (Tabelle).
2. Die Filterfunktion für die erste Regel aus der Liste (Tabelle) wird auf das Paket angewandt. Falls die Filterfunktion den Wert „true“ zurückgibt, wird die Aktion aus der Regel angewandt. Anderenfalls ist die aktuelle Regel die nächste Regel aus der Liste (Tabelle).
3. Falls keine der Filterfunktionen das Paket selektiert hat, wird die Default-Regel für die Liste (Tabelle) angewandt.

In IPTABLES können beliebig viele Tabellen definiert werden, aber drei sind vordefiniert: INPUT, OUTPUT und FORWARD. Diese Tabellen haben eine spezielle Bedeutung, da die eingehenden, ausgehenden oder weiterzuleitenden Pakete zuerst durch jeweils eine dieser Tabellen gefiltert werden. In Abhängigkeit davon, ob die Pakete vom System selbst neu erstellt, von ihm empfangen oder weitergeleitet werden sollen, werden sie in unterschiedlicher Weise bearbeitet. Die Regeln für die Verarbeitung sind:

- **Eingehende Pakete**: Für ein eingehendes Paket wird zuerst eine Routing-Entscheidung getroffen. Die Routing-Entscheidung kommt entweder zu dem Ergebnis, dass ein Paket für das lokale System bestimmt ist oder dass es weitergeleitet werden soll. Falls die Routing-Entscheidung besagt, dass das Paket für das lokale System bestimmt ist, wird durch die Anwendung der INPUT-Tabelle entschieden, ob das Paket wirklich bearbeitet werden soll.
- **Ausgehende Pakete**: Für die IP-Pakete, die vom lokalen System erzeugt wurden, wird die Entscheidung über die Verarbeitung durch die Anwendung der OUTPUT-Tabelle getroffen.
- **Weiterzuleitende Pakete**: Weiterzuleitende Pakete werden zunächst wie eingehende Pakete behandelt. Falls die Routing-Entscheidung besagt, dass ein Paket zur Weiterleitung bestimmt ist, wird in einem zweiten Schritt durch Anwendung der FORWARD-Tabelle über die Weiterleitung des Pakets entschieden.

5.2 Das Kommunikationsprotokoll (NETLINK)

NETLINK [33] ist ein Kommunikationsprotokoll für den Datenaustausch zwischen dem Kernel und den Benutzerprozessen im Linux-Betriebssystem. Die Kommunikation in NETLINK erfolgt durch den Austausch asynchroner Nachrichten, die entweder von den Benutzerprozessen an den Kernel oder vom Kernel an die Benutzerprozesse versandt werden. Das NETLINK-Protokoll findet vor allem bei der Kommunikation zwischen Benutzerprozessen und Netzwerkdiensten des Kernels Verwendung. So kann durch NETLINK beispielsweise der Routing-Dienst des Kernels (Lesen und Ändern der Routingtabellen, Konfiguration der Netzwerkschnittstellen etc.) komplett gesteuert werden. Eines der bekanntesten Dienstprogramme, das auf NETLINK basiert, ist IPROUTE2 [34].

Von der Seite eines Benutzerprozesses aus betrachtet ist der Ablauf der NETLINK-Kommunikation folgender:

- **Eröffnung des NETLINK-Sockets**: Um NETLINK zu benutzen, muss der Benutzerprozess zuerst einen NETLINK-Socket eröffnen. Unter Linux werden alle Sockets mit dem „socket ()“-Systemaufruf eröffnet. Dabei müssen folgende Parameter angegeben werden:

Protokolldomäne, Socket-Typ und Protokollnummer. Für NETLINK wurde in Linux eine eigene Protokollfamilie (auch: Protokolldomäne) definiert: PF_NETLINK. Der Socket-Typ beziehungsweise die Protokollsemantik, die vom NETLINK-Protokoll unterstützt wird, ist entweder SOCK_PACKET oder SOCK_RAW.[30] Für jeden Service, der das NETLINK-Protokoll unterstützt, ist eine eigene Protokollnummer definiert. So steht etwa NETLINK_ROUTE für die Kommunikation mit dem Routing-Service im Kernel.

- **Versenden und Empfangen von NETLINK-Nachrichten**: Durch den NETLINK-Socket werden nunmehr die NETLINK-Nachrichten versandt. Die Kommunikation verläuft bei NETLINK-Sockets asynchron, das heisst, dass die Antworten des Kernels auf die Anfragen des Benutzerprozesses nicht in chronologischer Reihenfolge erfolgen müssen, sondern beliebig angeordnet werden können. Die Zuordnung einer Antwort zur jeweiligen Anfrage erfolgt durch die Sequenznummer. Jede NETLINK-Nachricht hat eine Sequenznummer, und die NETLINK-Nachricht mit der Antwort erhält dieselbe Sequenznummer wie die Nachricht mit der Anfrage.

5.2.1 Format der NETLINK-Nachrichten

Die Grundlage des NETLINK-Protokolls ist die NETLINK-Nachricht. Jede NETLINK-Nachricht hat folgendes Format:

<table>
<tr><td colspan="2">Length</td></tr>
<tr><td>Type</td><td>Flags</td></tr>
<tr><td colspan="2">Sequence Number</td></tr>
<tr><td colspan="2">ProcessID</td></tr>
<tr><td colspan="2">Service Template</td></tr>
<tr><td colspan="2">Service Specific Data</td></tr>
</table>

Die Bedeutung der Felder in der Nachricht ist folgende:

- **Length (32 Bit)**: Die Länge der Nachricht in Bytes inklusive Header.
- **Type (16 Bit)**: Der Typ der Nachricht. Die NETLINK-Spezifikation definiert bereits drei Typen von Nachrichten: NLMSG_NOOP (eine Nachricht ohne Funktion), NLMSG_ERROR (eine Fehlernachricht) und NLMSG_DONE (das Ende einer fragmentierten Nachricht). Weitere Nachrichtentypen sind protokollspezifisch.
- **Flags (16 Bit)**: Die Nachrichten-Flags. Für alle Protokolle sind folgende Flags vordefiniert: NLM_F_REQUEST (markiert die Request-Nachricht), NLM_F_MULTI (markiert einen Teil einer fragmentierten Nachricht), NLM_F_ACK (wird gesetzt, falls eine ACK-Nachricht nach

30 Das NETLINK-Protokoll macht keinen Unterschied zwischen diesen zwei Protokolltypen; in beiden Fällen wird paketorientierte Kommunikation mit demselben Datenformat verwendet.

der Ausführung des Request verlangt wird) und NLM_F_ECHO (wird gesetzt, falls die Nachricht als Echo zurückgeschickt werden soll).

- **Sequence-Number (32 Bit)**: Die Sequenznummer der Nachricht. Diese Nummer wird benötigt, um die Antworten den Anfragen zuordnen zu können.
- **ProcessID (32 Bit)**: Die Prozessnummer des Absenders. Die ProcessID ist 0, falls der Absender der Kernel ist.
- **Service Template (variabel, abhängig von Protokollnummer und Type)**: Die Daten der Nachricht. Die Struktur und die Länge hängen von der Protokollnummer und dem Nachrichtentyp ab.
- **Service Specific Data (variabel, im Form von TLVs)**: Eine Liste mit zusätzlichen Informationen zum Service-Template. Jede zusätzliche Information ist im so genannten TLV (Type Length Value)-Format dargestellt. Das TLV-Format besteht aus der Länge (16 bit), dem Typ (16 bit) und dem Wert (den restlichen Daten).

5.2.2 NETLINK-Protokolle

Zur Zeit sind in NETLINK folgende Protokolle definiert:

- **NETLINK_ROUTE**: Das NETLINK-Route-Protokoll definiert eine Schnittstelle für die Abfrage, Änderung und Konfiguration von Netzwerkschnittstellen, Routing und Traffic-Control (TC) für das IPV4-Protokoll.
- **NETLINK_ENSKIP**: NETLINK_ENSKIP ist ein Protokoll für die Konfiguration von ENSKIP (Verschlüsselung von TCP/IP-Paketen).
- **NETLINK_USERSOCK**: NETLINK_USERSOCK ist für künftige Protokolle reserviert.
- **NETLINK_FIREWALL**: Dieses Protokoll vermittelt die Entscheidung über die Bearbeitung von IPV4-Paketen, auf die die QUEUE-Aktion von IPTABLES angewandt wurde.
- **NETLINK_TCPDIAG**: Das Protokoll dient der Überwachung von TCP-Sockets.
- **NETLINK_NFLOG**: Das Protokoll ist für den Empfang von IP-Paketen zuständig, auf die die ULOG-Aktion von IPTABLES angewandt wurde.
- **NETLINK_ARPD**: Das Protokoll wird für die Bearbeitung von ARP-Nachrichten durch einen Benutzerprozess (ARPDaemon) verwendet.
- **NETLINK_ROUTE6**: Das NETLINK-Route-Protokoll definiert eine Schnittstelle für die Abfrage, Änderung und Konfiguration von Netzwerkschnittstellen, Routing und Traffic-Control (TC) für das IPV6-Protokoll.
- **NETLINK_IPV6_FW**: Dieses Protokoll vermittelt die Entscheidung über die Bearbeitung der IPV6-Pakete, auf die die QUEUE-Aktion von IPTABLES angewandt wurde.
- **NETLINK_DNRTMSG**: Das Protokoll ist für die Bearbeitung von DECNet-Routing-Nachrichten durch einen Benutzerprozess zuständig.
- **NETLINK_TAP_X (16 Protokolle)**: Das Protokoll dient der Anbindung an die Ethernet-Simulation-Netzwerkschnittstellen (TAP-Interfaces). Für jedes TAP-Interface existiert eine Protokollnummer, angefangen mit NETLINK_TAP_BASE.

Im Rahmen der vorliegenden Arbeit sind die folgenden beiden NETLINK-Protokolle von Bedeutung: NETLINK_ROUTE und NETLINK_FIREWALL.

- **NETLINK_ROUTE**

Das NETLINK_ROUTE-Protokoll ist für die Implementierung des Authentifizierungsschemas wichtig, da durch dieses Protokoll Informationen über Netzwerkschnittstellen erhalten werden können. Zu diesen Informationen gehören unter anderem die IP-Adresse und die logische Nummer (Logical-Interface-Handle) der Netzwerkschnittstelle. Diese Informationen werden verwendet, um Werte für die entsprechenden Felder in IN-Interface- und OUT-Interface-COPS-Objekten zu erhalten.

Der Ablauf für das Auslesen von Informationen über Netzwerkschnittstellen ist folgender:

- Der Benutzerprozess versendet eine NETLINK-Nachricht mit dem Typ RTM_GETADDR an den Kernel.
- Der Kernel versendet für jede Netzwerkschnittstelle an den Benutzerprozess eine NETLINK-Nachricht des Typs RTM_NEWADDR, in der die vollständige Beschreibung der Netzwerkschnittstelle enthalten ist. Nach der letzten Nachricht mit der Beschreibung der Netzwerkschnittstelle versendet der Kernel eine Nachricht des Typs NLMSG_DONE, um das Ende der Datenübertragung zu signalisieren.

Dieser Ablauf wird in Abbildung 19 dargestellt.

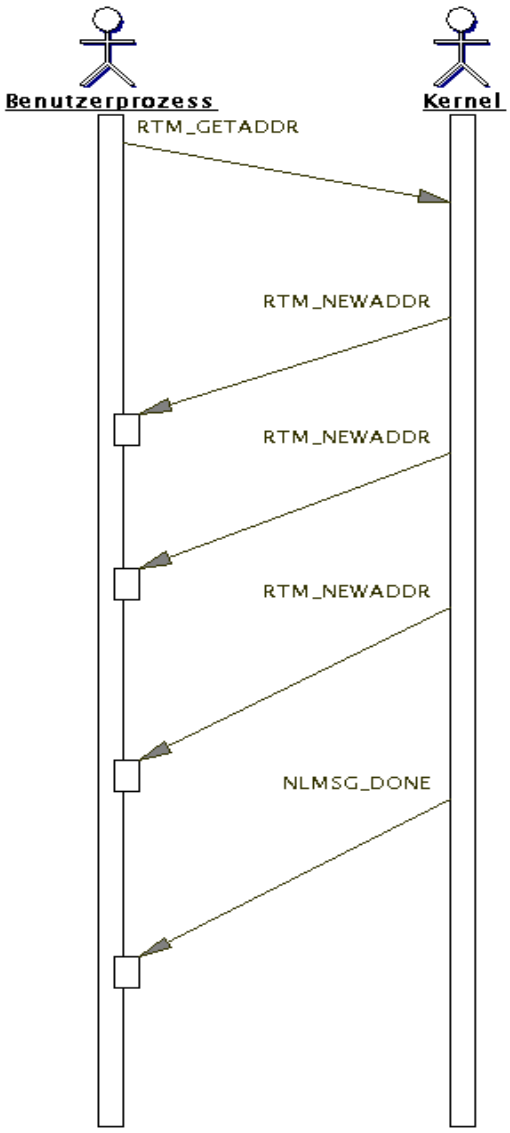


Abbildung 19: Ablauf der Kommunikation für den Erhalt von Netzwerkschnittstelleninformationen vom Kernel.

- **NETLINK_FIREWALL**

Das NETLINK_FIREWALL-Protokoll ermöglicht Entscheidungen über IP-Pakete, die durch IPTABLES mittels der QUEUE-Aktion an den Benutzerprozess ausgelagert wurden. Dieses Protokoll wird von der Implementierung des Authentifizierungsschemas benutzt, um RSVP-Nachrichten zu empfangen und gegebenenfalls zu ersetzen oder zu verwerfen. Der Ablauf der Kommunikation beim NETLINK_FIREWALL-Protokoll ist folgender:

- Der Benutzerprozess setzt den Modus für den Empfang der Pakete durch den Versand einer NETLINK-Nachricht mit dem Typ IPQM_MODE. In dieser Nachricht wird der Modus für den Empfang der Pakete festgelegt. Dieser Modus betrifft die Entscheidung darüber, ob der Benutzerprozess nur die Metainformationen über die Pakete oder auch die Pakete selbst empfangen soll.

- Der Kernel versendet für jedes Paket, für das die Entscheidung über die Bearbeitung von IPTABLES an den Benutzerprozess ausgelagert wird, eine NETLINK-Nachricht mit dem Typ IPQM_PACKET an den Benutzerprozess.
- Der Benutzerprozess versendet für jede NETLINK-Nachricht mit dem Typ IPQM_PACKET, die er vom Kernel empfängt, eine Entscheidung über die Bearbeitung an den Kernel, indem er eine NETLINK-Nachricht mit dem Typ IPQM_VERDICT versendet. Der Inhalt der Nachricht beinhaltet im jeden Fall den Code der Entscheidung, und falls das Paket ersetzt werden soll, den Inhalt des neuen Pakets.

Dieser Ablauf wird in Abbildung 20 veranschaulicht.

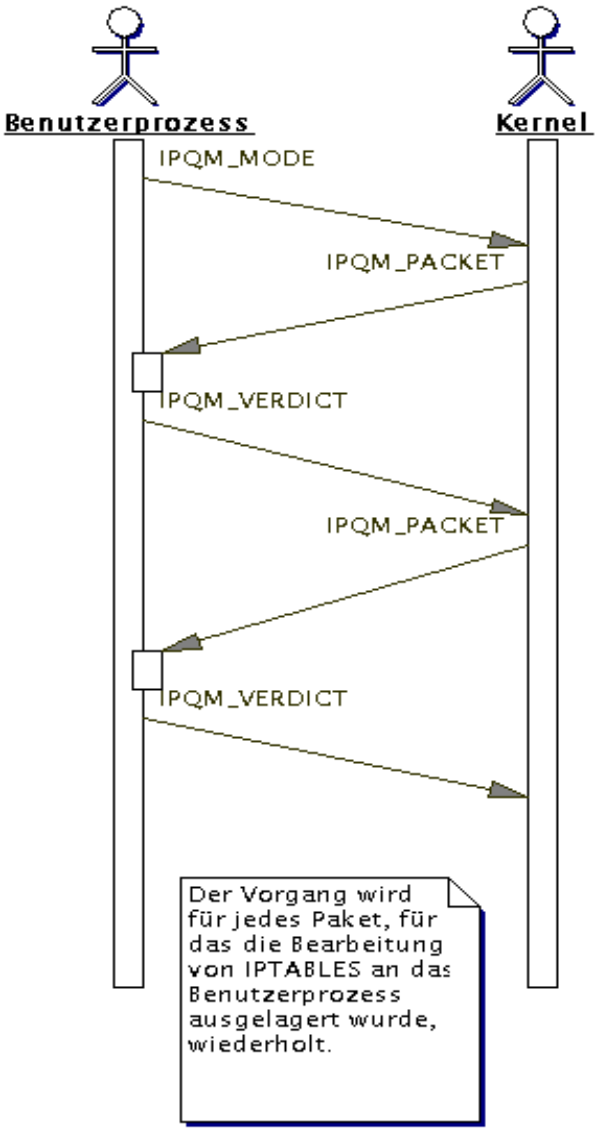


Abbildung 20: Ablauf der Kommunikation bei NETLINK_FIREWALL.

- **Mögliche Entscheidungen des Benutzerprozesses**

Der Benutzerprozess kann über jedes IP-Paket, das per NETLINK_FIREWALL-Protokoll empfangen wurde, eine der folgenden Entscheidungen treffen:

1. das Paket akzeptieren (äquivalent zur ACCEPT-Aktion von IPTABLES);
2. das Paket verändern und akzeptieren (durch diese Entscheidung kann der Inhalt des Pakets ausgetauscht werden);
3. das Paket verwerfen (äquivalent zur DROP-Aktion von IPTABLES).

Um Pakete empfangen zu können, für die die Entscheidung an den Benutzerprozess ausgelagert wurde, muss der Benutzerprozess zuerst eine Anfrage (Request) in Form einer NETLINK-Nachricht des Typs IPQM_MODE mit folgendem Service-Template an den Kernel versenden:

Mode	Reserved	Reserved
Range		

Die Bedeutung der Felder ist:

- **Mode (8 Bit)**: Dies ist der Modus für den Empfang der Pakete. IPQ_COPY_NONE: es werden keine Pakete empfangen; IPQ_COPY_META: nur Metainformationen werden empfangen; IPQ_COPY_PACKET: Metainformationen und das Paket selbst werden empfangen.
- **Range (32 Bit)**: Falls der Modus den Wert IPQ_COPY_PACKET hat, definiert dieser Wert die maximale Länge (in Bytes) der Paketdaten, die übertragen werden.

Nach der Registrierung des Empfängers wird jedes Paket, für das die Entscheidung über die Bearbeitung an den Benutzerprozess ausgelagert wurde, durch den NETLINK-Socket an den Benutzerprozess versandt. Dabei hat die NETLINK-Nachricht den Typ IPQM_PACKET und folgendes Service-Template:

PacketID	
Mark	
Timestamp_m	
Timestamp_u	
Hook	
Input Device Name	
Output Device Name	
HW_Protocol	HW_Type
HW_Addrlen	Reserved
Hardware Address	
Data Length	
Payload	

Die Bedeutung der Felder ist folgende:

- **PacketID (32 Bit)**: Die eindeutige ID des Pakets. Diese Zahl wird beim Versenden der Entscheidung benutzt, um das Paket referenzieren zu können.
- **Mark (32 Bit)**: Die Markierung des Pakets (falls das Paket von Netfilter/IPTables markiert wurde).
- **Timestamp_m (32 Bit)**: Die Empfangszeit des Pakets (in Sekunden seit 00:00 01.01.1970).
- **Timestamp_u (32 Bit)**: Der Zusatz zur Empfangszeit in Millisekunden.
- **Hook (32 Bit)**: Die ID des Firewall-Moduls, das das Paket aufgefangen hat.
- **Input Device Name (128 Bit)**: Der Name der Eingangsschnittstelle.
- **Output Device Name (128 Bit)**: Der Name der Ausgangsschnittstelle.
- **HW_Protocoll (16 Bit)**: Die Hardware-Protokollnummer (in Netzwerk-Byte-Reihenfolge).
- **HW_Type (16 Bit)**: Der Hardwaretyp.
- **HW_Addrlen (16 Bit)**: Die Länge der Hardwareadresse.
- **Hardware Address (64 Bit)**: Die Hardwareadresse.
- **Data Length (32 Bit)**: Die Länge des Pakets (in Bytes).
- **Payload (Abhängig von Data Length)**: Der Inhalt des Pakets.

Nach dem Empfang eines Pakets durch NETLINK-Socket kann der Benutzerprozess eine Entscheidung über die Bearbeitung des Pakets treffen. Die Entscheidung wird per NETLINK-Socket in Form einer Nachricht des Typs IPQM_VERDICT an den Kernel versandt. Das Service-Template für die Nachricht ist:

Value
PacketID
Length
Payload

Die Bedeutung der Felder ist:

- **Value (32 Bit)**: Die Entscheidung des Benutzerprozesses (NF_ACCEPT: das Paket weiterverarbeiten oder NF_DROP: das Paket verwerfen).
- **PacketID (32 Bit)**: Die ID des Pakets (der Wert aus dem IPQM_PACKET).
- **Length (32 Bit)**: Falls der Inhalt des Pakets ersetzt werden soll, die Länge (in Bytes) des neuen Pakets.
- **Payload (die Länge hängt vom Feld „Length" ab)**: Falls der Inhalt des Pakets ersetzt werden soll, die Daten für das neue Paket.

5.2.3 Abweichungen in der Implementierung von NETLINK_FIREWALL in Linux-Kernels 2.4.x

Bei den ersten Tests von NETLINK_FIREWALL war es trotz völlig korrektem Format der NETLINK-Nachrichten unmöglich, einen Benutzerprozess als den Empfänger der QUEUE-Pakete zu registrieren. Bei der Untersuchung der Kernel-Source hat sich gezeigt, dass die Implementierung von NETLINK_FIREWALL in Linux Kernel 2.4 nicht genau mit der Beschreibung des Protokolls [33] übereinstimmt. Auf Seite 24 von [33] wurde das Format des Service-Templates für die MODE-Nachricht definiert als:

Mode	Reserved	Reserved
Range		

Daraus ergibt sich eine Länge von 8 Bytes für das Service-Template der MODE-Nachricht.

In der Datei '**net/ipv4/netfilter/ip_queue.c**' hingegen wurde die Funktion '**ipq_receive_peer**()', die die NETLINK-Nachrichten aus Benutzerprozessen bearbeitet, folgendermassen implementiert:

```
static int
ipq_receive_peer(struct ipq_peer_msg *pmsg,
                 unsigned char type, unsigned int len)
{
        int status = 0;

        if (len < sizeof(*pmsg))
                return -EINVAL;

        switch (type) {
        case IPQM_MODE:
                status = ipq_set_mode(pmsg->msg.mode.value,
                                      pmsg->msg.mode.range);
                break;

        case IPQM_VERDICT:
                if (pmsg->msg.verdict.value > NF_MAX_VERDICT)
                        status = -EINVAL;
                else
                        status = ipq_set_verdict(&pmsg->msg.verdict,
                                                 len - sizeof(*pmsg));
                        break;
        default:
                status = -EINVAL;
        }
        return status;
}
```

Eigentlich wäre diese Implementierung der Funktion richtig, aber wenn man die Definition der Struktur '**ipq_peer_msg**' untersucht (die Datei '**include/linux/netfilter_ipv4/ip_queue.h**' ,)also:

```
typedef struct ipq_verdict_msg {
        unsigned int value;             /* Verdict to hand to
netfilter */
        unsigned long id;               /* Packet ID for this verdict
*/
        size_t data_len;                /* Length of replacement data
*/
        unsigned char payload[0];       /* Optional replacement packet
*/
} ipq_verdict_msg_t;

typedef struct ipq_mode_msg {
        unsigned char value;            /* Requested mode */
        size_t range;                   /* Optional range of packet
requested */
} ipq_mode_msg_t;

typedef struct ipq_peer_msg {
        union {
                ipq_verdict_msg_t verdict;
                ipq_mode_msg_t mode;
        } msg;
} ipq_peer_msg_t;
```

stellt man fest, dass sizeof(ipq_peer_msg_t) = max(sizeof(ipq_verdict_msg_t),sizeof (ipq_mode_msg_t))= 12 Bytes ist.

Aufgrund der Zeile mit dem Code:

```
    if (len < sizeof(*pmsg))
              return -EINVAL;
```

werden alle Nachrichten mit der Länge von weniger als 12 Bytes nicht akzeptiert. Deswegen ist es in der Praxis notwendig, das folgende Service-Template für die MODE-Nachricht zu benutzen:

Mode	Reserved	Reserved
Range		
Reserved		

5.3 Implementierung des neuen Authentifizierungsschemas durch IPTABLES und NETLINK

Die Kontrolle der Bearbeitung von IP-Paketen, die sich aus der Kombination von IPTABLES und NETLINK (vor allem NETLINK_FIREWALL) ergibt, ermöglicht es, das Authentifizierungsschema in Linux zu implementieren, ohne die RSVP-Applikationen (RSVP-Router beziehungsweise RSVP-Daemon und RSVP-Empfängerapplikationen) verändern zu müssen. Für die Realisierung dieser Möglichkeit waren noch folgende Überlegungen nötig:

- **RSVP-Router**

RSVP-Router werden unter Linux in der Regel als so genannter Daemon-Prozess[31] implementiert. Das heisst, dass die RSVP-Implementierung ein Benutzerprozess ist, der die IP-Pakete vom Kernel empfängt. Da alle IP-Pakete, die vom Rechner empfangen werden, sowie alle IP-Pakete, die vom ihm versandt werden, zunächst durch IPTABLES gefiltert werden, ist es möglich, an dieser Stelle auch das in der vorliegenden Arbeit entwickelte Authentifizierungsschema zu erzwingen. Um dieses Authentifizierungsschema für einen RSVP-Router zu implementieren, reicht es, wenn die Entscheidungen über folgende RSVP-Nachrichten an den Policy-Server ausgelagert werden:

1. PATH-Nachrichten, die den Router verlassen;
2. RESV-Nachrichten, die im Rechner eintreffen.

Das Authentifizierungsschema wird folgendermassen erzwungen:

Alle ein- und ausgehenden IP-Pakete, die RSVP-Nachrichten beinhalten, werden durch IPTABLES per NETLINK_FIREWALL-Protokoll an einen Benutzerprozess umgeleitet. Der Benutzerprozess trifft die Entscheidung über die Verarbeitung für jedes RSVP-Paket, indem er die darin enthaltene RSVP-Nachricht per COPS-RSVP-Protokoll an den Policy-Server versendet, dessen Entscheidung umsetzt und per NETLINK_FIREWALL an den Kernel weiterleitet. So werden alle ausgehenden PATH-Nachrichten mit Anforderungen (Challenges) versehen und alle

31 Unter Linux wie auch unter Unix wird ein Benutzerprozess als Daemon-Prozess bezeichnet, wenn er im Hintergrund ohne direkte Interaktion mit dem Benutzer ausgeführt wird und bestimmte Dienste (wie zum Beispiel SMTP) zur Verfügung stellt.

eingehenden RESV-Nachrichten auf die Gültigkeit der Antworten (Responses) hin überprüft. Der Ablauf der Bearbeitung der Nachrichten wird in Abbildung 21 und 22 veranschaulicht.

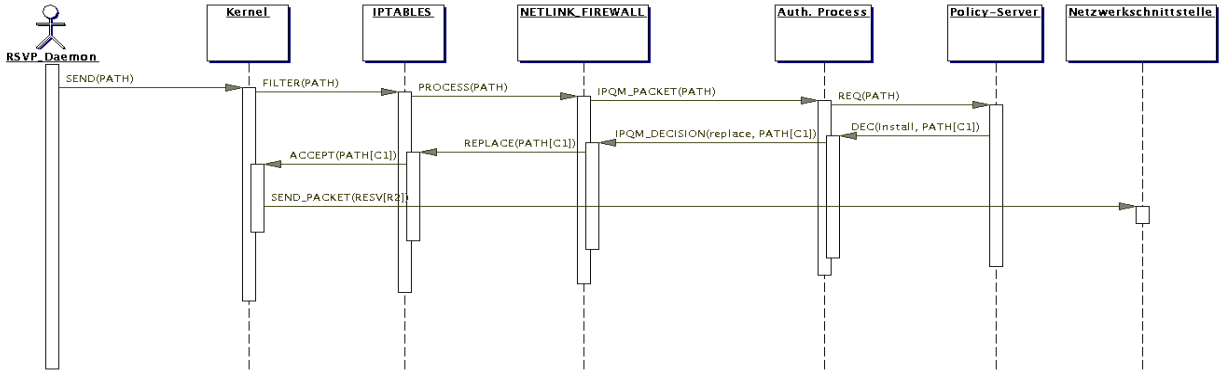


Abbildung 21: Der Ablauf der Generierung von Anforderungen (Challenges) für ausgehende PATH-Nachrichten durch den Einsatz von IPTABLES und NETLINK_FIREWALL.

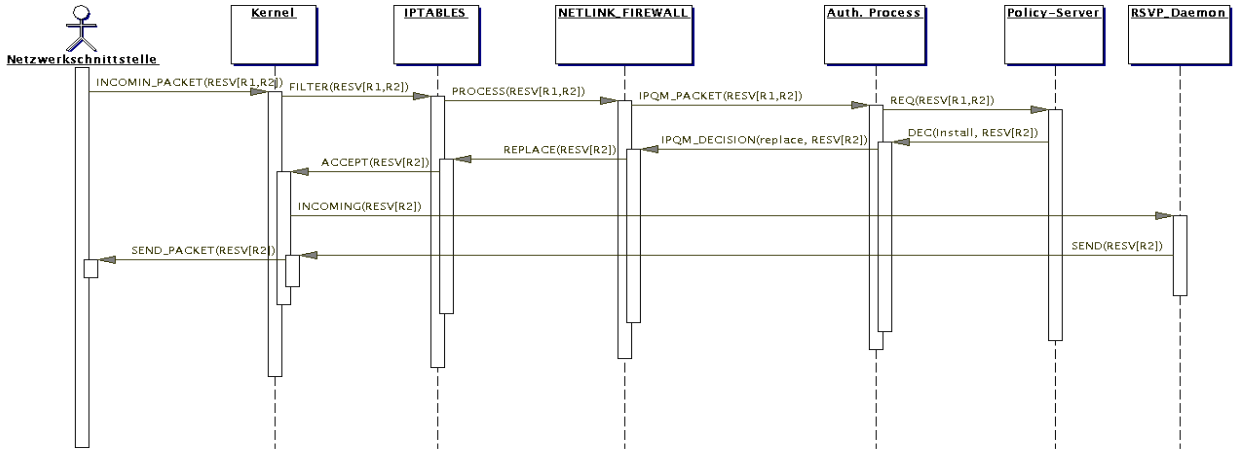


Abbildung 22: Der Ablauf der Authentifizierung von eingehenden RESV-Nachrichten durch den Einsatz von IPTABLES und NETLINK_FIREWALL.

- **RSVP-Empfänger**

Die RSVP-Empfänger sind unter Linux normalerweise ebenfalls als Benutzerprozesse implementiert. Dies ermöglicht es, die Implementierung der Authentifizierung durch das Filtern ein- und ausgehender Pakete in IPTABLES und ihre Bearbeitung durch NETLINK_FIREWALL in gleicher Weise wie bei RSVP-Routern durchzuführen. Der Ablauf der Bearbeitung der Pakete ist ähnlich wie bei RSVP-Routern. Der Unterschied besteht darin, dass bei RSVP-Empfängern die eingehenden PATH-Nachrichten und die ausgehenden RESV-Nachrichten für den Authentifizierungsprozess von Bedeutung sind und dass die Entscheidung über die Bearbeitung der Pakete durch den Authentifizierungsprozess selbst und nicht durch den Policy-Server getroffen wird.

Der Ablauf der Bearbeitung der Pakete wird in Abbildung 23 und 24 veranschaulicht.

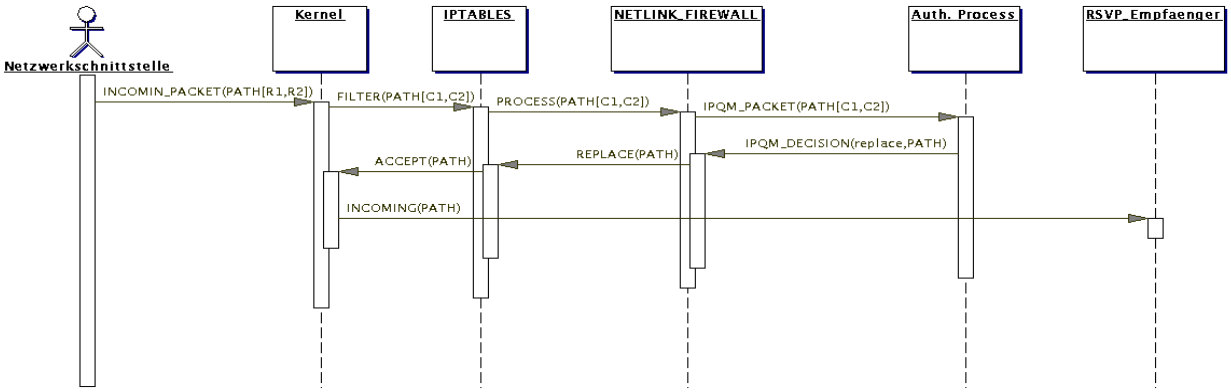


Abbildung 23: Der Ablauf der Bearbeitung der für einen RSVP-Empfänger eingehenden PATH-Nachrichten. Jede PATH-Nachricht wird zuerst durch den Authentifizierungsprozess auf Anforderungen (Challenges) untersucht; erst nach der Entfernung dieser Anfragen wird die Nachricht an den RSVP-Empfängerprozess weitergeleitet.

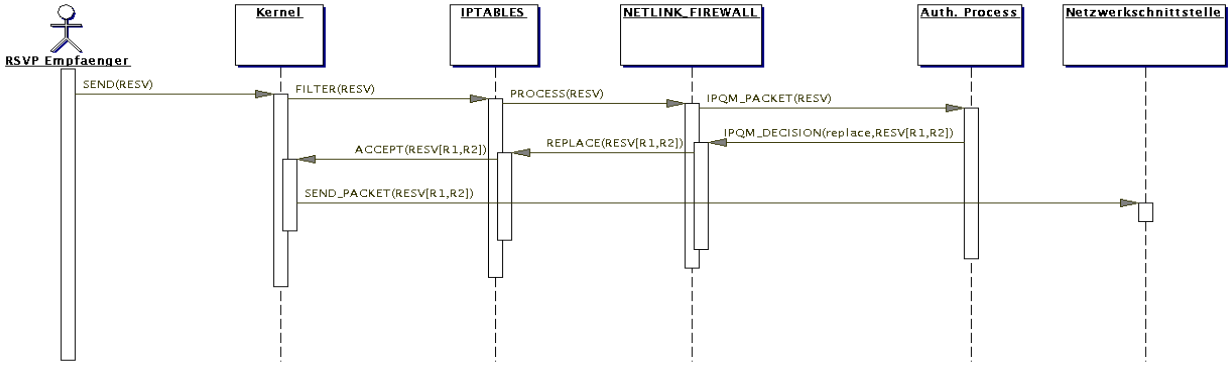


Abbildung 24: Der Ablauf der Bearbeitung von ausgehenden RESV-Nachrichten. Vor dem Versenden der RESV-Nachricht werden die Antworten (Responses) auf sämtliche Anforderungen (Challenges) aus den PATH-Nachrichten generiert und in die ausgehende Nachricht eingefügt.

5.4 NETLINK mit JAVA (JNETLINK)

Da im Rahmen der vorliegenden Arbeit bereits ein COPS-Client für JAVA entwickelt wurde, hat es sich angeboten, ihn für die Implementierung des Authentifizierungsschemas zu benutzen. Dabei stellt sich allerdings das Problem, dass JAVA keine Sockets aus der Domain PF_NETLINK unterstützt. Andererseits bietet JAVA die Möglichkeit, aus der Java-Virtual-Machine heraus den Maschinencode („native code“) aufzurufen. Dies wird durch das Java-Native-Interface (JNI) [35] ermöglicht. Um die NETLINK-Sockets benutzen zu können, wurde für die vorliegende Arbeit eine NETLINK-Bibliothek für JAVA (JNETLINK) entwickelt, die mittels JNI die Socketkommunikation implementiert und eine objektorientierte Schnittstelle für die Bearbeitung von NETLINK-Nachrichten anbietet.

5.4.1 Aufbau von JNETLINK

JNETLINK ist eine objektorientierte Abstraktion des NETLINK-Protokolls. JNETLINK besteht aus zwei Teilen:

1. einer Java-Bibliothek, die mittels JNI die NETLINK-Sockets benutzt und empfangene Daten in JAVA-Objekte sowie umgekehrt JAVA-Objekte, die ausgehende NETLINK-Nachrichten darstellen, in Daten umwandelt;
2. einer in C geschriebenen gemeinsam benutzbaren Bibliothek (Shared-Library), die für die NETLINK-Kommunikation die benötigten Funktionen implementiert.

Die Hauptkomponenten von JNETLINK werden im Folgenden beschrieben.

- **NETLINK-Connection**

Die zentrale Komponente von JNETLINK ist die NETLINK-Verbindung (NetlinkConnection). Die NETLINK-Verbindung ist definiert durch folgende Schnittstelle:

```
package com.crsn.jnetlink.core;

import java.io.IOException;

import com.crsn.jnetlink.core.message.*;

public interface NetlinkConnection {


    public static final int NETLINK_ROUTE=0;
    public static final int NETLINK_SKIP=1;
    public static final int NETLINK_USERSOCK=2;
    public static final int NETLINK_FIREWALL=3;
    public static final int NETLINK_TCPDIAG=4;
    public static final int NETLINK_NFLOG=5;
    public static final int NETLINK_ARPD=8;
    public static final int NETLINK_ROUTE6=11;
    public static final int NETLINK_IP6_FW=13;
    public static final int NETLINK_DNRTMSG=14;
    public static final int NETLINK_TAPBASE =16;

    public abstract void open() throws IOException;
    public abstract void close() throws IOException;;
    public abstract int sendMessage(NetlinkMessage msg) throws
IOException;
    public abstract NetlinkMessage readMessage() throws IOException;
    public abstract int getNetlinkFamily();
}
```

Die Aufgaben der NetlinkConnection sind:

- Das Öffnen und Schliessen der NETLINK-Verbindung (die Open-Methode).
- Das Senden von NETLINK-Nachrichten (die sendMessage()-Methode).
- Das Lesen der NETLINK-Nachrichten (die readMessage()-Methode).

Um eine Implementierung einer NETLINK-Connection zu erhalten, muss die Static-Methode „**createConnection()**“ aus der Klasse „**com.crsn.jnetlink.core.NetlinkConnectionFactory**“ mit folgenden Parametern aufgerufen werden:

- **netlinkFamily (int)**: Die Protokollnummer des NETLINK-Sockets (alle bisher bekannten Protokollnummern sind als Konstanten in der Schnittstelle „**com.crsn.jnetlink.core.NetlinkConnection**“ definiert).
- **messageFactory (NetlinkMessageFactory)**: Eine Implementierung der Schnittstelle „**com.crsn.jnetlink.core.message.NetlinkMessageFactory**“. Die Implementierung soll in der Lage sein, die Service-Templates für die NETLINK-Nachrichten für die entsprechende Protokollnummer zu generieren.

Die Schnittstelle „**com.crsn.jnetlink.core.message.NetlinkMessageFactory**“ definiert die Factory-Schnittstelle für die NETLINK-Nachrichten (siehe „Factory Method“[30], Seite 107) und hat folgende Signatur:

```
package com.crsn.jnetlink.core.message;

import java.nio.ByteBuffer;

public interface NetlinkMessageFactory {
    public NetlinkMessage create(ByteBuffer buffer);
}
```

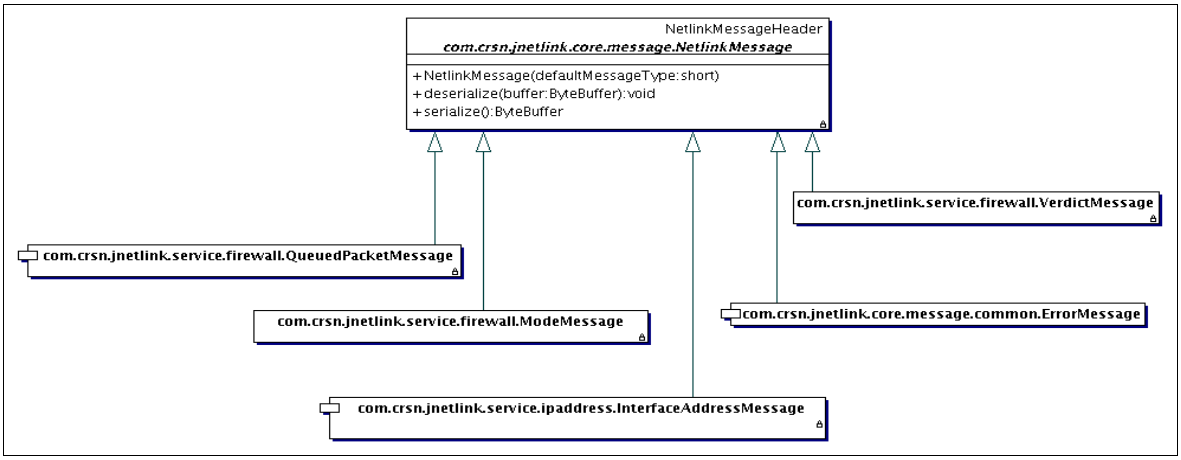
- **NetlinkMessage**

NetlinkMessage ist die abstrakte Oberklasse für alle Klassen von NETLINK-Nachrichten. Die Klasse wurde nach dem „Template Method“-Design-Pattern aufgebaut (vgl. [30], Seite 325). Die Klasse selbst übernimmt einen Teil (den Message-Header) der Serialisierung (also der Umwandlung einer Nachricht in ein Byte-Array) sowie der Deserialisierung (also der umgekehrten Umwandlung eines Byte-Arrays in eine Nachricht) und delegiert den Rest der Aufgabe an die abstrakte Template-Methode, deren Implementierung den Rest der Aufgabe erledigt (Service-Template und Service-Specific-Data). Die wichtigsten Methoden aus der NetlinkMessage-Klasse sind:

```
package com.crsn.jnetlink.core.message;

import java.io.UnsupportedEncodingException;
import java.net.*;
import java.nio.ByteBuffer;

public abstract class NetlinkMessage extends NetlinkMessageHeader {

    public NetlinkMessage(short defaultMessageType);

    public void deserialize(ByteBuffer buffer) {...}
    protected abstract void deserializeMessageContent(ByteBuffer
buffer);


    public ByteBuffer serialize() {...}
    protected abstract void serializeMessageContent(ByteBuffer
buffer);

    protected int getMessageLength() {...}
    protected abstract int messageContentLength();
```

```
}
```

Für jeden Typ von NETLINK-Nachrichten ist daher nötig, eine neue Klasse zu definieren. Zur Zeit sind folgende NetlinkMessage-Klassen definiert:



5.5 Implementierung des Authentifizierungsschemas mit JNETLINK

Nach der Implementierung von JNETLINK wurde es möglich, das NETLINK_FIREWALL-Protokoll innerhalb einer JAVA-Applikation zu benutzen und damit das Authentifizierungsschema in JAVA zu implementieren. Dabei wurden im Rahmen der vorliegenden Arbeit die folgenden zwei Applikationen entwickelt:

- **Border-Router**
- **Receiver-Authentication**

5.5.1 Border-Router

Die Border-Router-Applikation ist eine Applikation, die das Authentifizierungsschema für einen RSVP-Border-Router des ISP implementiert. Dabei wird beim Starten der Applikation eine Netfilter/IPTables-Regel definiert, die alle ein- und ausgehenden RSVP-Pakete an das QUEUE-Ziel umleitet. Die Applikation selbst empfängt per NETLINK_FIREWALL alle RSVP-Pakete und wendet in Abhängigkeit vom Typ der RSVP-Nachricht folgende Operationen an:

- Falls die Nachricht eine ausgehende PATH-Nachricht ist und sie die ISP-Domain verlässt, wird sie für die Authentifizierung per COPS-Request an den PDP versandt, und die entsprechenden POLICY_DATA-Objekte aus der Antwort werden in die PATH-Nachricht eingebunden. Anschliessend wird die ursprüngliche PATH-Nachricht durch eine PATH-Nachricht mit der generierten Anforderung (Challenge) ersetzt.

- Falls die Nachricht eine eingehende RESV-Nachricht ist, die von ausserhalb der ISP-Domain eintrifft, wird sie per COPS-Request an den PDP versandt. In Abhängigkeit davon, ob der PDP eine positive oder negative Antwort zurückgibt, wird sie entweder durch eine RESV-Nachricht ohne Response-Objekte für den ISP ersetzt und weitergeleitet oder aber einfach gelöscht.
- In allen anderen Fällen werden die Nachrichten unverändert weitergeleitet.

Die Border-Router-Applikation wurde ebenso wie der Java Policy Server als eine Avalon-Phoenix-Applikation etwickelt[32]. Dabei besteht die Border-Router-Applikation aus folgenden Komponenten (Blocks):

- **ServerAuth**: Die ServerAuth-Komponente emfängt die ein- und ausgehenden IP-Pakete, die RSVP-Nachrichten beinhalten, delegiert die Entscheidung über die Verarbeitung der Nachrichten an den COPSDecisionService (Block) und setzt die Entscheidung per NETLINK_FIREWALL um.
- **COPSDecisionService**: Die COPSDecisionService-Komponente sendet die RSVP-Nachrichten per COPS-RSVP-Protokoll (durch Benutzung von JCOPS-Bibliothek) an den Policy-Server und gibt die Entscheidung des Policy-Servers zurück.

Der Aufbau der Border-Router-Applikation kann Abbildung 25 entnommen werden.

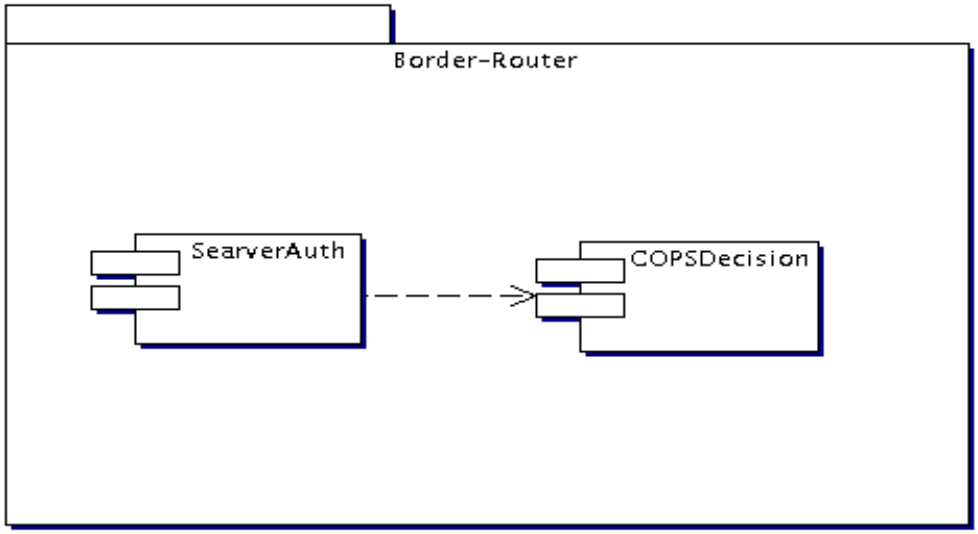


Abbildung 25: Die beiden Komponenten der Border-Router-Applikation und ihre Abhängigkeit.

5.5.2 Receiver-Authentication

Die Receiver-Authentication ist eine Applikation, die das clientseitige Authentifizierungsschema implementiert. Wie bei der Border-Router-Applikation wird eine Netfilter/IPTables-Regel definiert, die alle RSVP-Pakete an die QUEUE-Aktion umleitet. Beim Empfang eines RSVP-Pakets per NETLINK_FIREWALL wird in Abhängigkeit vom Typ der RSVP-Nachricht eine der folgenden Operationen angewandt:

- Falls ein PATH-Paket eintrifft, wird die Anforderung (Challenge) aus der Nachricht gespeichert. Dabei wird die RSVP-Session als Schlüssel benutzt.

32 Zu Avalon-Phoenix vgl. ausführlicher Kapitel 4.1, Seite 52 der vorliegenden Arbeit.

- Falls eine RESV-Nachricht den Rechner verlässt, werden Antworten (Reponses) für alle Anforderungen (Challenges) generiert, die in den PATH-Nachrichten empfangen wurden, und in die RESV-Nachricht eingebunden.
- Alle andere RSVP-Nachrichten werden unverändert akzeptiert.

Die Receiver-Authentication-Applikation besteht aus folgenden Komponenten (Blocks):

- **ClienAuth**: Die ClientAuth-Komponente empfängt und bearbeitet IP-Pakete mit RSVP-Nachrichten. Dabei benutzt sie den SHA1Encryption-Service, um für die Anforderungen (Challenges) aus den PATH-Nachrichten die Antworten (Responses) für die RESV-Nachrichten zu generieren.
- **SHA1Encryption**: Die SHA1Encryption-Komponente (Block) erzeugt nach dem Algorithmus, der in Kapitel 3 der vorliegenden Arbeit beschrieben ist, aus einer Anforderung (Challenge) eine Antwort (Response). Um dies zu erreichen, muss die Komponente den Schlüssel für die Entschlüsselung der Videodaten erhalten. Der Mechanismus zur Beschaffung dieses Schlüssels ist nicht in dieser Komponente definiert; statt dessen wird ein Service (KeyFactoryService) benutzt, um den Schlüssel zu erhalten.
- **KeyFactoryService**: KeyFactoryService wurde definiert, um den Mechanismus zur Beschaffung des Schlüssels für die Entschlüsselung der Videodaten, und damit zugleich des Schlüssels, der benötigt wird, um aus den Anforderungen (Challenges) die Antworten (Responses) zu generieren, austauschbar zu gestalten. Die Aufgabe dieses Services, der von der SHA1Encryption-Komponente benutzt wird, ist die Beschaffung des Schlüssels. Zur Zeit existiert nur eine Implementierung dieses Services, die den Schlüssel aus der Block-Konfiguration entnimmt. Sie wurde für die Funktionstests der Client-Authentication benutzt.

Einen Überblick über alle Komponenten der Client-Authentication-Applikation und ihre Abhängigkeiten gibt Abbildung 26.

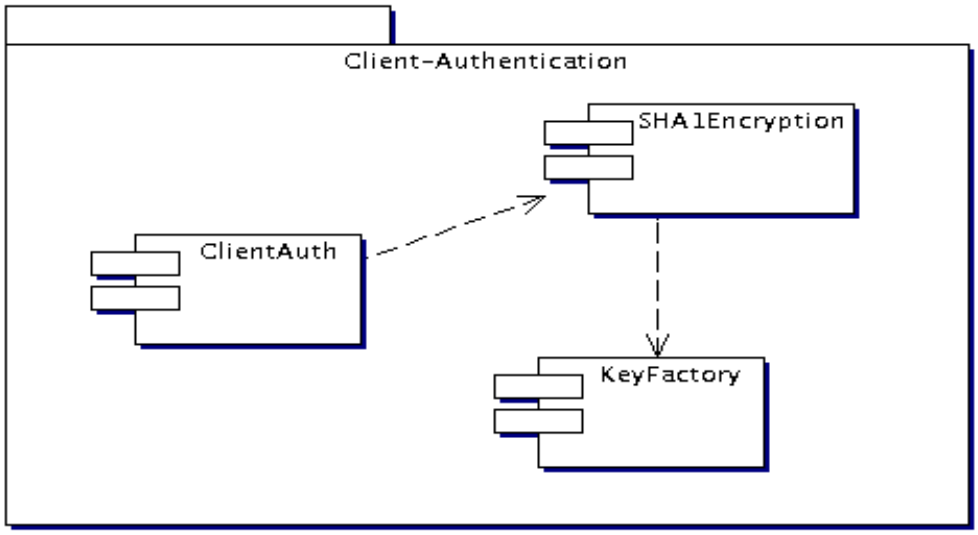


Abbildung 26: Komponentendiagramm der Client-Authentication-Applikation.

6 Zusammenfassung und Ausblick

6.1 Ergebnisse

Im Rahmen der vorliegenden Diplomarbeit wurde anhand eines realitätsnah konzipierten Szenarios für Multicast-Datenübertragung unter Verwendung des Protokolls für die Reservierung von Netzwerkressourcen (RSVP) ein **Authentifizierungsschema** entwickelt. Dieses Authentifizierungsschema unterscheidet sich von den bisher vorgeschlagenen dadurch, dass potentielle Denial-of-Service-(DoS)-Attacken selbst dann ausgeschlossen sind, wenn die an der Datenübertragung beteiligten Internet-Service-Provider (ISPs) selbst eine DoS-Attacke gegen andere ISPs beabsichtigten. Das hier entwickelte Authentifizierungsschema setzt daher keinerlei Kooperation zwischen den ISPs voraus. Darüber hinaus ist es skalierbar, und auch der Datenschutz der Empfänger wird durch dieses Authentifizierungsschema gewährleistet.

Weitere Ergebnisse dieser Diplomarbeit sind:

- **Java Policy Server**: Java Policy Server ist ein Policy-Server, der das entwickelte Authentifizierungsschema mittels COPS-RSVP-Protokoll implementiert. Da der Java Policy Server clusterfähig ist (Hot-Replication des Zustands durch Multicast), ist er sowohl hochperformant als auch hochverfügbar. Die Anzahl der Clients, die durch den Server bedient werden können, kann also durch das Hinzufügen weiterer Rechner zum Cluster erhöht werden, und falls eine Instanz im Cluster ausfällt, können die restlichen Instanzen die Clients ohne grosse Verzögerung übernehmen. Der Java Policy Server ist eine JAVA-Applikation und daher plattformunabhängig. Die Architektur des Policy-Servers ist komponentenbasiert, was den sehr einfachen Austausch und die ebenso einfache Veränderung der einzelnen Komponenten ermöglicht. Leistungsmessungen für die aufwendigste Aufgabe des Java Policy Servers, die Bearbeitung von PATH-Nachrichten, haben ergeben, dass er in der Lage ist, eine PATH-Nachricht in durchschnittlich 3.67 msec zu bearbeiten. Daraus kann die Folgerung abgeleitet werden, dass eine Instanz des Java Policy Servers 272 PATH-Nachrichten pro Sekunde bearbeiten kann.
- **JCOPS**: JCOPS ist eine vollständige und hochperformante Open-Source-Implementierung (unter LGPG-Lizenz) des COPS-Protokolls. JCOPS ermöglicht sowohl server- als auch clientseitige Implementierung beliebiger COPS-Client-Typen in JAVA. Da auch JCOPS ausschliesslich in JAVA implementiert wurde, liegt auch hier Plattformunabhängigkeit vor. Leistungsmessungen von JCOPS haben ergeben, dass durch eine COPS-Verbindung, in der sowohl der Server als auch der Client eine JCOPS-Implementierung ist, ohne COPS-Authentifizierung durchschnittlich 8209 Anfragen pro Sekunde bearbeitet werden können.
- **JNETLINK**: JNETLINK ist eine JAVA-Bibliothek für die Benutzung des NETLINK-Protokolls in JAVA-Programmen. Da JNETLINK eine C-Bibliothek für die Implementierung von NETLINK-Kommunikation durch JNI (Java Native Interface) benutzt, ist JNETLINK nicht plattformunabhängig, sondern nur für den Einsatz unter Linux bestimmt.
- **Transparente Implementierung der Authentifizierung für Border-Router und RSVP-Empfänger**: Dank der Eingriffsmöglichkeiten in die Bearbeitung von IP-Paketen, die sich unter Linux durch die Existenz von IPTABLES und NETLINK eröffnen, war es möglich, das Authentifizierungsschema zu implementieren, ohne die Implementierungen des RSVP-Routers oder RSVP-Empfängers zu verändern. Das Authentifizierungsschema wurde ebenfalls in JAVA implementiert.

- **Abweichung der NETLINK_FIREWALL-Implementierung in den Linux-Kernel-Versionen 2.4.x von der Protokollbeschreibung**: Ein unvorhergesehenes Nebenergebnis der vorliegenden Arbeit war die Feststellung, dass die NETLINK_FIREWALL-Implementierung in den Linux-Kernel-Versionen 2.4.x von der Protokollbeschreibung [33] abweicht. In der Protokollbeschreibung wurde das Format des Service-Templates für die MODE-Nachricht so definiert, dass sich dafür eine Länge von 8 Bytes ergibt. In der Datei 'net/ipv4/netfilter/ip_queue.c' hingegen wurde die Funktion 'ipq_receive_peer()', die die NETLINK-Nachrichten aus Benutzerprozessen bearbeitet, so implementiert, dass sie eine Mindestlänge von 12 Bytes haben muss.

6.2 Ausblick

Obwohl die Benutzung von RSVP für die Reservierung von Netzwerkressourcen derzeit nicht sehr weit verbreitet ist, gibt es Gründe für die Annahme, dass die Popularität von RSVP in näherer Zukunft steigen wird. Zu diesen Gründen gehört vor allem der stetig zunehmende Einsatz des Internets als Übertragungsmedium für die Verbreitung von Audio- und Videoinhalten. Da die Verbreitung kostenpflichtiger Inhalte ohne Reservierung von Netzwerkressourcen auch ohne Garantien bezüglich der Qualität des Empfangs erfolgen müsste und daher nur schwer vorstellbar ist, gewinnt der Einsatz von RSVP für solche Zwecke immer mehr an Wahrscheinlichkeit. Ein weiterer Grund liegt in der Zunahme von Echtzeitapplikationen, die das Internet für die Datenübertragung nutzen. Die Best-Effort-Datenübertragung ist mit solchen Applikationen per Definition nicht vereinbar, und daher müssen auch hier Netzwerkressourcen reserviert werden, um eine akzeptable Übertragungsqualität sicherzustellen.

Bei der Reservierung von Netzwerkressourcen, die ja naturgemäss stets begrenzt sind, stellt sich zugleich immer die Frage nach der Berechtigung zur Betätigung einer solchen Reservierung. Normalerweise ist die Bereitstellung von Ressourcen mit einer direkten oder indirekten Entschädigung verbunden: Wer Ressourcen reserviert, muss dafür bezahlen. Es liegt daher im Interesse des Anbieters, möglichst effizient zu überprüfen, ob die Bezahlung auch wirklich erfolgt ist, und sicherzustellen, dass diese Überprüfung nicht ohne Weiteres zu umgehen ist. Gleichzeitig sollte aber auch der Datenschutz für den Kunden gewährleistet werden. Es sollte also verhindert werden, dass Daten über die Empfänger gesammelt werden können, die dann beispielsweise zur Erstellung von Verbraucherprofilen für zielgruppenspezifische Werbesendungen missbraucht werden können.

Die vorliegende Diplomarbeit zeigt für das spezielle Beispiel von Datenübertragung per Multicast, wie bei der Authentifizierung der Reservierung von Ressourcen mit relativ geringem Aufwand eine Balance zwischen den Bedürfnissen der Sicherheit einerseits und dem Datenschutz andererseits erreicht werden kann. Sie kann als Grundlage für ein Authentifizierungsschema bei der Reservierung von Ressourcen für die kommerzielle Fernsehübertragung verwendet werden. Darüber hinaus steht zu hoffen, dass die im Rahmen dieser Diplomarbeit entwickelten JAVA-Bibliotheken JCOPS und JNETLINK künftig auch in anderen Projekten eingesetzt werden und mit dazu beitragen, dass die Programmiersprache JAVA stärker als bisher als Programmiersprache für die Implementierung von Netzwerkdiensten berücksichtigt wird.

7 Literaturverzeichnis

[1] Branden, R., Zhang, L., S. Berson, Herzog, S., Jamin, S.: Resource ReSerVation Protocol (RSVP). RFC 2205, September 1997.

[2] Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R., Sastry, A.: The COPS (Common Open Policy Service) Protocol. RFC 2748, January 2000.

[3] Postel, J.: Internet Protocol. RFC 791, September 1981.

[4] Deering, S., Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification. RFC 1883, December 1995.

[5] Deering, S.: Host Extensions for IP Multicasting. RFC 1112, August 1989.

[6] Fenner, W.: Internet Group Management Protocol, Version 2. RFC 2236, November 1997.

[7] Cain, B., Deering, S., Kouvelas, I., Fenner, B., Thyagarajan, A.: Internet Group Management Protocol, Version 3. RFC 3376, October 2002.

[8] Braun, T.: Multicast-Kommunikation im Internet. In: Elektrotechnik und Informationstechnik, Heft 6, 2000: 389-398.

[9] Postel, J.: Transmission Control Protocol. RFC 793, September 1981.

[10] Postel, J.: Internet Control Message Protocol. RFC 792, September 1981.

[11] Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997.

[12] Wroclawski, J.: The Use of RSVP with IETF Integrated Services. RFC 2210, September 1997.

[13] Herzog, S.: RSVP Extensions for Policy Control. RFC 2750, Januar 2001.

[14] The Internet Assigned Numbers Authority: Internet Assigned Numbers Authority , http://www.iana.org/.

[15] Herzog, S., Boyle, J., Cohen, R., Durham, D., Rajan, R., Sastry, A.: COPS Usage for RSVP. RFC 2749, January 2000.

[16] Balmer, R., Braun, T.: Zugangskontrolle für einen Videoverteildienst mit IP Multicast. 17. DFN-Arbeitstagung über Kommunikationsnetze, Düsseldorf, 10.-13. Juni 2003.

[17] Stattenberger, G.: Scalable Quality of Service Support for Mobile Users. Diss., Institut für Informatik und angewandte Mathematik, Universität Bern, 2002.

[18] Sun Microsystems: RPC: Remote Procedure Call Protocol specification: Version 2. RPC1057, June 1988.

[19] Zimmer, D. E.: Neuanglodeutsch. Über die Pidginisierung der Sprache. In: Deutsch und anders.. Reinbek: Rowohlt, 1997: 7-85.

[20] Zimmer, E. D.: Hundert Computerbegriffe in zehn europäischen Sprachen. In: Deutsch und anders. Reinbek: Rowohlt, 1997: 86-104.

[21] Eastlake, D. 3rd, Jones, P.: US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001.

[22] Dobbertin, H.: The Status of MD5 After a Recent Attack. In: Crypto Bytes, Heft 2, Summer 1996: 1-6.

[23] Sun Microsystems: Java Technology , http://java.sun.com.

[24] Sun Microsystems: Java HotSpot Technology , http://java.sun.com/products/hotspot/.

[25] Apache Group: Avalon-Phoenix , http://avalon.apache.org/phoenix/index.html.

[26] W3-Consortium: Extensible Markup Language (XML) , http://www.w3.org/XML/.

[27] Sun Microsystems, Inc.: JAR File Specification , http://java.sun.com/j2se/1.4.2/docs/guide/jar/.

[28] Qostali, A.: JCAPI Tool Kit , http://www.info.uqam.ca/~qostali/COPS/index.htm.

[29] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. A System of Patterns. Volume 1. Chichester etc.: John Wiley & Sons Ltd., 1996.

[30] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns Elements of Reusable Object-Oriented Software. Reading, Mass. etc.: Addison-Wesley, 1995.

[31] Free Software Foundation, Inc.: GNU Lesser General Public License , http://www.gnu.org/copyleft/lesser.html.

[32] Bela Ban: JGroups - A Toolkit for Reliable Multicast Communication , http://www.jgroups.org/.

[33] Salim, J., Khosravi, H., Kleen, A., Kuznetsov, A.: Linux Netlink as an IP Services Protocol. RFC 3549, July 2003.

[34] Kuznetsov A.: IPROUTE2 , ftp://ftp.inr.ac.ru/ip-routing.

[35] Sun Microsystems, Inc.: Java Native Interface , http://java.sun.com/j2se/1.4.2/docs/guide/jni/.