Bandwidth Broker for Differentiated Services

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

Vorgelegt von:

Emmanuel A. Granges

Leiter der Arbeit:
Prof. Dr. Torsten Braun

Betreuer der Arbeit:
Manuel Günter

Forschungsgruppe Rechnernetze und Verteilte Systeme (RVS)
Institut für Informatik und angewandte Mathematik

2000

Für Nadia, Janine und Alain

Inhaltsverzeichnis

Zusammenfassung

Die vorliegende Diplomarbeit hat zum Ziel, die in [VPN Architecture] vorgestellten Konzepte für die Realisierung von Service Brokern im Differentiated Services Umfeld weiterzuentwickeln, zu verfeinern und umzusetzen. Diese Konzepte gehen von einer Brokerhierarchie und einem Broker Signaling Protocol (BSP) aus, welche es ermöglichen sollen, Service Level Agreements (SLAs) elektronisch und weitgehend automatisch auszuhandeln. Service Broker arbeiten nach dem Client-Server Konzept, um Dienste und deren Parameter bilateral auszuhandeln.

Um eine rasche Verbreitung solcher Broker zu ermöglichen und um den Betreibern von Service Brokern möglichst viele Freiheiten zu lassen, sollen einerseits die verwendete Plattform wie auch die angebotenen Dienste frei wählbar sein. Diese Ziele sollen durch die Umsetzung in einer objektorientierten Programmiersprache und durch den modularen Aufbau erreicht werden. Die resultierende prototypische Implementierung eines speziellen Service Brokers realisiert die notwendige Funktionalität eines sogenannten External Bandwidth Brokers und ermöglicht somit, in einem Differentiated Services Umfeld das Bandbreitenmanagement weitgehend zu automatisieren. Durch die Automatisierung wird es möglich, von heute statischen SLAs zu dynamischen bzw. temporären SLAs überzugehen. Die Schwerpunkte der Umsetzung liegen im gesicherten (verschlüsselten) Informationsaustausch zwischen den Brokern, in der Allgemeinheit und Flexibilität der Realisierung und in der Portabilität der Implementierung.

Die Allgemeinheit der Realisierung wird einerseits dadurch erreicht, dass die Programmierung in einer weitverbreiteten und objektorientierten Sprache (Java) unter Verwendung einer portablen Graphik-Bibliothek (Swing) erfolgt. Andererseits widerspiegelt sich die Portabilität und Erweiterbarkeit im Graphischen User Interface (GUI), welches die Verarbeitung von Messages des sehr allgemein definierten BSP-Protokolls [CATI BSP] unterstützt. Das GUI baut sich dynamisch auf und ist damit in der Lage, heute noch nicht definierte Protokollobjekte korrekt darzustellen. Die Bearbeitung dieser Protokollobjekte kann direkt in diesem GUI erfolgen. Der Aufbau der BSP-Syntax ermöglicht nahezu beliebige Erweiterungen wie die verschlüsselte Kommunikation bzw. Signatur der SLAs. Die Verschlüsselung kann wahlweise mit PGP oder aber auch mit einem weniger rechenintensiven Verfahren, welches auf MD5 beruht, erfolgen.

Kapitel 1

Einführung

1.1 Computernetzwerke am Ende des 20. Jahrhunderts

Computernetzwerke findet man heute auf der ganzen Welt und in gewissem Sinne auch um die ganze Welt herum (Funk- und Satellitennetze). Aus der globalen Perspektive stechen einige grosse proprietäre weltumspannende Datennetz hervor, welche vorwiegend einem bestimmten Zweck dienen, wie z.B. dem militärischen Informationsaustausch, der Telefonie, dem Radio oder dem Fernsehen. Netzwerke ganz unterschiedlicher Grösse bilden in einem Verbund unterschiedlicher Netztopologien das heutige Internet. Diese Teilnetze unterscheiden sich nicht nur durch ihre Grösse sondern auch durch die Netzwerktechnologie, durch den Aufbau, durch die eingesetzten Kommunikationsprotokolle, durch den Verwendungszweck und durch ihren öffentlichen oder privaten Charakter. Ein weiteres grosses Netzwerk ist das Stromkabelnetz, welches meist auch recht abgelegene Regionen der Welt erreicht. Auch dieses Netzwerk ist, mit der entsprechenden Technologie versehen, als Datennetz verwendbar. Zu Beginn des nächsten Jahrtausends wird erwartet, dass eine solche Verwendung des Stromkabelnetzes eine weite Verbreitung finden wird.

Eine zweite Perspektive zeigt eine grosse Anzahl von Wide Area Networks (WANs), von Local Area Networks (LANs), von Personal Computern und weiteren Geräten, welche heute meist aus Sicherheitsgründen aber auch aus technischen Gründen immer noch nicht miteinander verbunden sind. Neue Technologien und Protokolle sollen hier die Vision der totalen Vernetzung wahrmachen.

1.1.1 Netztechnologien und Protokolle

1.1.1.1 Referenzmodelle

Das ISO-OSI Referenzmodell, welches in [Tan, Kap. 1.4] beschrieben wird, besteht im Wesentlichen aus den 7 Schichten, welche in Abbildung 1.1 dargestellt werden. Jede Schicht umschreibt durch ihre Bezeichnung eine Aufgabe der digitalen Datenübertragung. Dieses Schichtenmodell wird für das Internet im Wesentlichen übernommen, wobei die Schichten 5 und 6 im Internet-Referenzmodell (TCP/IP-Referenzmodell) fehlen und die Schichten 1 und 2 zu einer logischen Schicht verschmolzen sind. In dieser Arbeit wird grundsätzlich zwischen den Schichten oberhalb der Schicht 3, der Schicht 3 und den Schichten unterhalb der Schicht 3

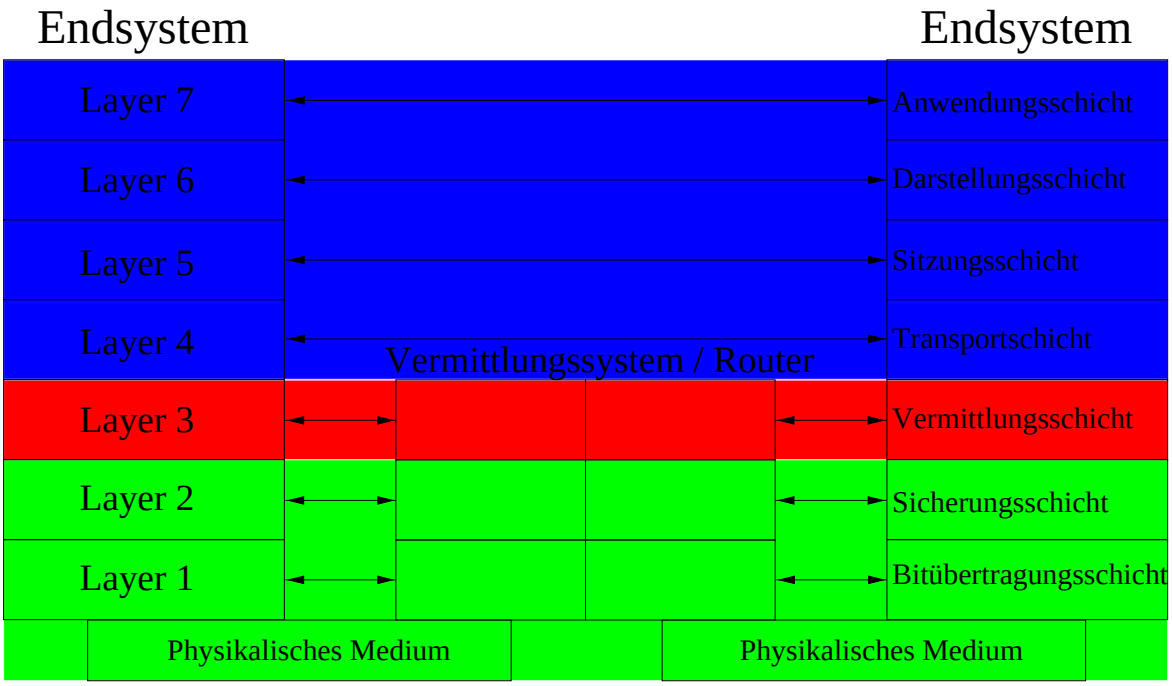


Abbildung 1.1: Das ISO-OSI Referenzmodell

unterschieden. In der Folge verwende auch ich für die Schichten den Fachbegriff 'Layer' und nehme dabei Bezug auf die Nummern des ISO-OSI Referenzmodells.

1.1.1.2 Netztechnologien auf Layer 1 und 2

Die Vielfalt an Netztechnologien ist sehr gross und nimmt ständig zu. Die digitale Übertragungstechnik hat gegenüber der analogen Technik viele neue Möglichkeiten eröffnet und zum rasanten und exponentiellen Wachstum des weltweiten Datenübertragungsvolumens beigetragen.

Nebst der klassischen Art, Computer über Medien wie z.B. Kupferkabel oder Glasfaserkabel zu verbinden, wird zunehmend auch die kabellose Verbindung z.B. via Radio- und Infrarotwellen genutzt. Die Charakteristiken der Datenströme in all diesen Netzen sind dementsprechend unterschiedlich.

Die verwendeten Protokolle dieser Layer erfüllen ganz unterschiedliche Ansprüche. So sind z.B. bei Asynchronous Transfer Mode (ATM) viele Funktionen vorhanden, welche gemäss ISO-OSI-Referenzmodell durch höhere Layer abgedeckt werden müssten. Diese Funktionalität auf Layer 1 oder 2 eröffnet aber in speziellen Fällen die Möglichkeit, Aufgaben höherer Layer direkter, effizienter und zuverlässiger umzusetzen.

1.1.1.3 Die Vermittlungsschicht (Layer 3)

Die Vermittlungsschicht hat durch die grosse Verbreitung des Vermittlungsprotokolls Internet Protocol (IP) eine besondere Bedeutung erhalten. IP dient sehr unterschiedlichen Technologien und einer Mehrheit von Netzanwendungen als universelle Schnittstelle. IP kann auf nahezu alle Protokolle des Layers 2 abgebildet werden, weil IP von dieser Schicht nur sehr einfache Dienste

verlangt. Diese Anspruchslosigkeit ist der Hauptgrund für die sehr häufige Verwendung und somit weltweite Verbreitung von IP. Dieser Vorteil ist aber auch der grosse Nachteil von IP, denn dadurch kann IP den Protokollen des Layers 4 keine zuverlässigen Dienste anbieten.

Der Status der universellen Schnittstelle hat somit einen wesentlichen Nachteil für die kommerzielle Nutzung. Dieser Nachteil fällt zudem mit der zunehmenden Auslastung des Internets immer mehr ins Gewicht. Durch geeignete Mechanismen wird nun versucht die Unterstützung von Dienstgüte auf Layer 3 in den Griff zu bekommen, um damit allen auf IP aufsetzenden Anwendungen hinreichend gute Dienste anzubieten, auch wenn Verbindungen ausgelastet oder sogar überlastet sind. Die Problematik der Dienstgütenunterstützung wird in Kapitel 1.2 näher erläutert.

1.1.1.4 Protokolle

Die eingesetzten Protokolle für verbindungsorientierte wie auch verbindungslose Dienste sind sehr vielfältig. Um verschiedene Netztechnologien miteinander zu verbinden und neue Dienste anzubieten, werden heute zudem immer wieder neue Protokolle entwickelt. Die Anzahl an vernetzten Elementen hat dazu geführt, dass einige Protokolle diesbezügliche Anpassungen erfahren mussten. So wird in den kommenden Jahren das IP-Protokoll Version 4 (IPv4) durch die Version 6 (IPv6) ersetzt werden müssen, weil die Anzahl an verschiedenen Kennungen in IPv4 künftig nicht mehr ausreichen wird, um allen vernetzten Geräten eine eindeutige Kennung zuzuordnen.

1.1.2 Netzaufbau

Die Netztopologien sind, bedingt durch die eingesetzte Technologie, sehr unterschiedlich. So sind heute meist ring- und sternförmige Netze oder Busnetze zu finden. Die logische Struktur der Netzwerke ist heute meist noch das Abbild dieser Topologien und somit statisch. Besonders wenn man die Möglichkeiten zur Bildung logischer Netzwerke betrachtet, indem einzelne verteilte Systeme oder Netzelemente durch geeignetes Management zusammengefasst werden, ist eine zunehmende Dynamik und Verflechtung zu erwarten. Diese Zusammenfassung kann auf unterschiedliche Weise in verschiedenen Layer erfolgen. Die in der mobilen Telekommunikation eingesetzten Netzwerke sind ein erstes gutes Beispiel für eine solche Dynamik und für den heterogenen Aufbau, da ein solches Netzwerk aus einem eher statischen wie auch aus einem dynamischen Teil besteht.

1.1.3 Qualitätsmerkmale

Die Qualitätsmerkmale heutiger Netze sind so unterschiedlich wie die eingesetzten Netztechnologien. Einige Technologien (z.B. Wählnetze) bieten eine implizite Qualitätsunterstützung an, bei anderen Technologien muss die Qualitätsunterstützung explizit durch geeignetes Management erfolgen bzw. simuliert werden. Bisher wurde Qualität meist dadurch erreicht, dass durch die geeignete Dimensionierung und sorgfältige Verwaltung bei normaler Netzlast eine gewisse Qualität erwartet werden konnte. Bei hoher oder unerwarteter Netzlast konnte hingegen keine Qualität erreicht und dementsprechend konnten keine Garantien abgegeben werden.

In einigen Fällen wurden die notwendigen Qualitätsmerkmale durch den Einsatz von teuren dienstgütenunterstützenden Technologien auf Layer 1 und 2 erreicht.

1.2 Dienstgüte in Computernetzwerken

Wie bereits erwähnt, besitzen Netzwerke unterschiedlicher Technologien über entsprechende implizite oder explizite Qualitätsgarantien. Je nach eingesetzter Netztechnologie ist so eine Dienstgütenunterstützung auf Layer 1 oder 2 gewährleistet. Asynchronous Transfer Mode (ATM) [Kyas98] ist ein gutes Beispiel für Dienstgütenunterstützung auf Layer 2. Dienstgütenunterstützung dieser Art ist jedoch sehr teuer und setzt entsprechende technologiegebundene Managementwerkzeuge voraus, welche meist keine Dienstgütenunterstützung beim Übergang zu Netzwerken anderer Technologien sicherstellen. Demzufolge wenden heute fast ausschliesslich Unternehmen mit eigenen grossen Netzwerken und mit speziell ausgebildetem Personal diese Form von Dienstgütenunterstützung an. In vielen Fällen lohnt sich der finanzielle Aufwand nicht, auf Layer 2 solche Technologien einzusetzen und zu verwalten.

Für die Unterstützung von Dienstgüte über die Grenzen proprietärer Netzwerke hinaus sind entsprechende Technologien in Entwicklung, welche auf das IP-Protokoll setzen und somit auf Layer 3 Dienstgüte garantieren können. Die Dienstgüte wird im Bezug auf Netzwerktechnologie häufig Quality of Service oder kurz QoS genannt.

1.2.1 Wozu Quality of Service?

QoS ist für sehr viele Anwendungen wie z.B. Videokonferenzen, Echtzeitanwendungen und die Telefonie äusserst wichtig und z.T. sogar zwingend erforderlich. Diese Anwendungen sind bisher auf verbindungsorientierte Datennetze mit impliziten Qualitätsmerkmalen aufgesetzt worden. Weil das Internet eine effizientere Nutzung und damit eine günstigere Bewirtschaftung ermöglicht, aber die erwähnten Anwendungen Qualitätsmerkmale benötigen, ist es unumgänglich, dass IP-Dienste mit garantierten Qualitätsmerkmalen angeboten werden.

Gegenüber verbindungsorientierten Datennetzen hat das Internet drei wichtige Vorteile:

1. Die Bandbreite eines physischen Links kann wesentlich effizienter genutzt werden, weil die im verbindungsorientierten Fall reservierte aber ungenutzte Bandbreite auch genutzt werden kann.

2. Die Kosten sind massiv tiefer, da die ungenutzte Bandbreite anderweitig verwendbar ist und deshalb nicht verrechnet werden muss.

3. Die sehr hohe Flexibilität und Technologieunabhängigkeit auf Layer 2.

Viele heutige Benutzer verbindungsorientierter Datennetze sind sich gewisser Dienstgütegarantien gewohnt, obschon diese nicht zwingend im vorhandenen Ausmass oder über die gesamte Dauer der Verbindung erforderlich sind. Hinter diesem Umstand verbirgt sich ein erhebliches Einsparungs- und Optimierungspotential, welches kommerzielle Anbieter aufdecken und nutzen wollen.

1.2.2 QoS Merkmale

Die benötigten QoS-Anforderungen hängen stark von der jeweiligen Anwendung ab. Es gibt Netzanwendungen, welche mit Paketverlusten umgehen können, indem die Verarbeitung beim Empfänger ab einem gewissen Prozentsatz an erhaltenen Daten korrekt erfolgen kann. Andere Anwendungen, wie z.B. ein Filetransfer, sind darauf angewiesen, dass alle IP-Pakete unverändert und in der richtigen Reihenfolge ankommen, um durch den Empfänger wieder aneinandergefügt werden zu können.

QoS soll differenzierte Dienste ermöglichen, deren Qualitätsmerkmale unabhängig voneinander sind. So sollen einerseits Dienste existieren, welche durch eine fest vorgegebene Bandbreite mit ev. variierender Verzögerung und möglichen Paketverlusten (d.h. unzuverlässig) charakterisiert sind und andererseits Dienste, welche eine statistisch garantierte Bandbreite mit maximaler Verzögerung und höchster Zuverlässigkeit anbieten.

1.2.2.1 Bandbreite / Durchsatz (Throughput)

Die maximale Bandbreite bzw. der Durchsatz ist im Wesentlichen von den vorhandenen Leitungen (Links) abhängig. Wenn eine grössere Bandbreite gewünscht wird, als ein einziger Link anbieten kann, können bei einem paketvermittelnden Netzwerk mehrere Links die erforderliche Bandbreite zur Verfügung stellen, unter Umständen auch über verschiedene Pfade. Auf Layer 2 sind je nach eingesetzter Technologie Beschränkungen (Einstellungen) möglich, so dass mehrere Kunden sich einen Link teilen können, ohne gegenseitig Störungen befürchten zu müssen.

Die Router in den Netzknoten müssen zudem in der Lage sein, den durchschnittlichen aber in gewissen Fällen auch maximalen Verkehr zwischen den angeschlossenen Links mit der gewünschten Performance zu verarbeiten.

1.2.2.2 Verzögerung (Mean Delay, Jitter)

Verzögerungen entstehen hauptsächlich durch Queueing in Netzelementen wie Router. Eine weitere Ursache für Verzögerungen sind grosse Übermittlungsdistanzen, wie bei der Übermittlung via Satelliten oder der Kommunikation zweier geografisch weit auseinandergelegenen Sender und Empfänger. Auch die eingesetzte Linktechnik spielt eine wichtige Rolle. In Bezug auf dieses Qualitätsmerkmal haben z.B. Glasfaser oder Kupferdraht ganz unterschiedliche Eigenschaften.

Die Verzögerungsschwankungen (Jitter), welche durch das Queueing in den Routern entstehen, müssen in einem bestimmten Rahmen bleiben, um einen kontinuierlichen Datenstrom zu ermöglichen. Die Anzahl der Netzelemente (Hops), welche ein IP-Paket durchlaufen muss, ist wesentlich, weil jeder Hop eine massgebliche Verzögerung verursacht. Bei bidirektionalen Echtzeitanwendungen müssen ausserdem die Endsysteme in der Lage sein, gewisse minimale Antwortzeiten zu garantieren.

1.2.2.3 Zuverlässigkeit (Error Rate)

Zuverlässigkeit bedeutet, dass die Pakete einerseits das Ziel erreichen und andererseits der Inhalt der Pakete nicht verändert wird. Zuverlässigkeit kann auch heissen, dass die Reihenfolge

der Pakete nicht vertauscht wird.

Das IP-Protokoll ist prinzipiell unzuverlässig. Deshalb ist es nicht zwingend, dass die Netztechnologie auf Layer 2 Zuverlässigkeit garantiert. Ist die vorhandene Netztechnologie jedoch sehr zuverlässig, so wird der darauf aufbauende IP-Dienst auch einigermassen zuverlässig sein. Je besser das Netzmanagement ist, desto besser wird der Dienst erbracht werden können.

1.2.2.4 Sicherheit

Verschlüsselung
Der Bedarf an Verbindungen mit privatem Charakter nimmt stark zu. Nebst VPN ist auch der private Internetverkehr (z.B. Internetbanking) in vielen Fällen auf Verschlüsselung angewiesen und in anderen Fällen zumindest wünschenswert. Diese Verschlüsselung findet heute meist auf Anwendungsebene statt.

Authentisierung
Insbesondere mit der Einführung von Internet Shopping, von Internet Banking und nicht zuletzt von QoS im Internet wird die Authentisierung zunehmend wichtiger. Benutzer (oder Anwendungen) müssen sich authentisieren, um bestimmte Dienste nutzen zu können und um den Dienstanbietern individuelle Abrechnungen zu ermöglichen.

1.2.3 Wie wird QoS durch Netzwerke unterstützt

QoS kann direkt durch die Netztechnologie (Layer 1 und 2) unterstützt werden, wie dies z.B. bei ATM der Fall ist. Andernfalls müssen Protokolle und Mechanismen auf höheren Schichten vorhanden sein, welche in der Lage sind, über unzuverlässige Technologien mittels Redundanz, Überwachung und Verwaltung von Ressourcen die entsprechende Dienstgüte zu erreichen.

Wenn ein Netzwerk für eine ganz bestimmte Aufgabe benutzt wird (Dedicated Network), genügt es, die Netzwerkressourcen sorgfältig zu dimensionieren. Andernfalls sind nebst der sorgfältigen Dimensionierung noch weitere Mechanismen wie Admission Control, Traffic Policing, Traffic Shaping und Queueing notwendig.

1.2.3.1 Internet

Das Internet integriert eine Vielzahl unterschiedlicher Netztechnologien. Sein Aufbau ist dementsprechend vielfältig. Den gemeinsamen Nenner bildet das Internet Protokoll (IP). Die Anforderungen, welche IP an diese Netztechnologien stellt, sind gering. Damit wird die Kommunikation zwischen sehr unterschiedlichen Technologien möglich, was IP zu einer sehr grossen Verbreitung verholfen hat. Das IP-Protokoll ist ein Protokoll, welches einen Best-Effort Dienst ohne Qualitätsgarantien bezüglich Zuverlässigkeit, Bandbreite und Verzögerung anbietet. Dieser Best-Effort Dienst ist verbindungslos, unzuverlässig und Punkt-zu-Punkt bzw. bei Multicast [RFC1301] Punkt-zu-Mehrpunkt.

Gewisse fehlende Qualitätsmerkmale werden heute nur punktuell und temporär erreicht. Einige Protokolle höherer Schichten haben bisher gewisse Schwächen von IP kompensiert, sind jedoch nicht in der Lage, bezüglich Bandbreite und Verzögerung, Garantien abzugeben. Ein sehr bekanntes Beispiel ist das Internet Transport Control Protocol TCP [RFC793], welches durch die Flusskontrolle in der Lage ist, Paketverluste festzustellen und zu korrigieren.

Obschon viele Anwendungen ohne ausreichend gute Qualitätsmerkmale des Netzwerkes kaum einsetzbar sind, fehlen heute noch universelle Protokolle und Mechanismen zur Qualitätssicherung im Internet. Aus diesem Grunde bestehen heute immer noch proprietäre Weitverkehrsnetze, welche durch eine geeignete Verwaltung und entsprechende Abschirmung gewisse Qualitätsmerkmale erreichen. Dies ist kostenintensiv und nur so zuverlässig wie die Menschen, welche die manuellen Einstellungen vornehmen.

Internet Service Provider (ISP) wollen Dienste auf ihren Netzwerken auf konkrete Kundenbedürfnisse zuschneiden und damit Dienste mit bestimmten Qualitätsmerkmalen zu entsprechenden Konditionen anbieten können. Da IP ein Best-Effort-Dienst ist, welcher von sich aus kein QoS anbietet, müssen diese Aufgaben durch Protokolle höherer Schichten oder spezielle Mechanismen erbracht werden. Die heutigen Protokolle sind jedoch nicht in der Lage, die geforderte Qualität im gewünschten Umfang, in der gewünschten Differenzierung und in der gewünschten Wirtschaftlichkeit und Handhabung zu gewährleisten. Deshalb wird QoS im Internet heute selten und nur uneinheitlich unterstützt.

1.2.3.2 Verbindungsorientierte Netzwerke

Unternehmen mit verbindungsorientierten Netzwerken können implizite Qualitätsmerkmale anbieten, werden ihre Netze aber künftig besser auslasten müssen, um konkurrenzfähig bleiben zu können. QoS-Protokolle und QoS-Mechanismen werden daher auch hier Einzug finden müssen, damit diese Unternehmen ihre Dienstleistungen differenzierter und zu günstigeren Konditionen anbieten können.

1.2.3.3 QoS in LANs / WANs

In LANs und WANs sind teilweise gewisse Qualitätsmerkmale durch die eingesetzte Technologie gegeben. So bietet z.B. Token Ring bezüglich Verzögerung gewisse Garantien. ATM ist auch ein gutes Beispiel für die Unterstützung von bestimmten Qualitätsmerkmalen auf Layer 2 in LANs und WANs.

Bei WANs muss häufig der sehr teure Weg über Miet- oder Wählleitungen eingeschlagen werden, um gewisse Funktionalitäten bereitzustellen und bestimmte Anwendungen zuverlässig betreiben zu können. Auch hier ist die durchschnittliche Auslastung meist sehr gering, so dass die allozierte und bezahlte Bandbreite zu einem grossen Teil ungenutzt bleibt.

Da heute schon viele LAN-Anwendungen Qualitätsgarantien erfordern, werden LANs entweder auf Netztechnologien aufgesetzt, welche gewisse Qualitätsmerkmale aufweisen, oder aber sie werden sehr grosszügig dimensioniert. Beide Varianten sind nicht optimal, da im ersten Fall viel höhere Anschaffungskosten anfallen und im zweiten Fall nebst mittleren Anschaffungskosten die Auslastung im Durchschnitt schlecht ist. Ausserdem ist bei grosszügiger Dimensionierung die „Dienstgüte“ relativ. UDP-Verkehr kann mitunter sehr aggressiv sein, d.h. dass die gesamte Bandbreite durch UDP-Verkehr belegt wird. Somit ist es wahrscheinlich, dass andere UDP-Verbindungen oder TCP-Verkehr gar nicht mehr möglich sind. Dies ist jedoch in vielen Fällen die günstigste Lösung, um viele Netzwerkprobleme temporär zu lösen.

Beispiel:
Angenommen ein 2 Mbps-Link sei vorhanden. Eine UDP-Verbindung bestehe bereits (IPTV,

IP-Multicast, ca. 1.5Mbps, MPEG-codierte Audio- und Videodaten). Zahlreiche weitere Voice over IP-Verbindungen (VoIP - ebenso UDP) bestünden zeitgleich. Werden damit die 2 Mbps bereits vollständig belegt, ist davon auszugehen, dass entweder gar keine FTP-Verbindung zu Stande kommt oder die FTP-Verbindung abbricht. Ausserdem ist es durchaus möglich, dass einige UDP-Verbindungen nicht mehr über die notwendige minimale Qualität verfügen und trotz adaptiver Protokolle wie das Real Time Protocol (RTP) [RFC1889] unbrauchbar werden. Dieses Szenario wird in der Praxis voraussichtlich sehr häufig auftreten, womit ein Bandbreitenmanagement unumgänglich werden wird.

In LANs wird QoS vereinzelt mittels Integrated Services (siehe Kapitel 1.2.5) realisiert oder mit Technologien der Layer 1 und 2 erfolgen, welche die neuen IEEE-Standards 802.1p und 802.1q implementieren.

1.2.3.4 QoS in VPN

Virtuelle private Netzwerke (Virtual Private Networks, VPN) sind logische Netze, die meist aus mehreren LANs bestehen, welche über ein öffentliches Weitverkehrsnetz wie das Internet, Mietleitungen (Leased Lines, Frame Relay) oder Wählnetze (ISDN) miteinander verbunden sind. Ausserdem sind VPNs meist statisch konfiguriert, obschon die Dynamik der Netzverkehrslast z.B. im Tagesverlauf bekannt ist und somit Kosteneinsparungen ohne weiteres möglich wären. Im Fall von Wählnetzen kann dieser Dynamik im Tagesverlauf insofern Rechnung getragen werden, dass z.B. in der Nacht die Verbindung abgebaut wird.

VPNs haben häufig einen privaten Charakter, d.h. einerseits, dass ein privater Standort mit einem anderen vertrauenswürdigen privaten Standort verbunden wird und andererseits, dass die Daten, welche in einem VPN über das Weitverkehrsnetz gehen, meist (firmen-)interne Daten sind, welche einen vertraulichen Charakter haben. Auch hier musste bisher aus Performance- und besonders aus Sicherheitsgründen häufig die teure Variante über Mietleitungen gewählt werden.

Da IP-VPNs zunehmend grössere Beliebtheit erlangen, ist es umso wichtiger, QoS auf Layer 3 anzubieten. Damit soll es möglich werden, mehrere Datenströme auf einem physikalischen Link unterschiedlich zu behandeln je nachdem, ob der Absender (oder die Anwendung) zu einem VPN gehört oder nicht.

Die heutige Technologie erlaubt VPN mit entsprechender QoS zu unterstützen. Dazu werden mittels RSVP verwaltete Tunnel-basierte Aggregate verwendet. Einerseits ist jedoch, wie in [IPnG] beschrieben, die Verwaltung solcher RSVP-Tunnels recht aufwendig und andererseits eine IP-in-IP-Einkapselung notwendig .

1.2.4 QoS auf Layer 3 (IP)

Die grosse Verbreitung des IP-Protokolls prädestiniert es, um QoS in den meisten Bereichen innert kurzer Frist mit relativ geringem Aufwand einführen zu können. Es liegt deshalb nahe, diesem Protokoll durch geeignete Mechanismen und geeignete übergeordnete Protokolle QoS-Eigenschaften zu verleihen. Einige Ansätze dazu werden in den folgenden Abschnitten näher erläutert.

1.2.5 Integrated Services (IntServ)

Der IntServ-Ansatz ist die Erweiterung der klassischen Internetarchitektur nach [RFC1633] und unterstützt QoS und Multicast. Die Erweiterung besteht aus folgenden Elementen:

1. Erweiterte Dienstmodelle, welche das Verhalten nach aussen beschreiben und durch Serviceklassen modelliert werden.

2. Flusskontrollmechanismen und Protokolle, um Reservierungen vorzunehmen, wie z.B. ST2(+) [RFC1190][RFC1819] und RSVP [RFC2205].

IntServ basiert auf der Alloziierung von Ressourcen pro Anwendungsdatenfluss (Flow), d.h. der Reservierung von Netz- und Systemressourcen wie Bandbreite, Puffer und CPU-Zeit für jeden Flow. Der Benutzer legt mittels Flussspezifikation (FlowSpec) und Filterspezifikation (FilterSpec) Reservations- (RSpec) und Trafficparameter (TSpec) fest, welche einer der folgenden drei Service Classes zugeordnet werden:

- Garanteed Service (GS) [RFC2211]
  GS bietet Bandbreitengarantie, mathematisch begrenzte Verzögerung und schliesst Verluste konformer Pakete aus.

- Controlled Load Service (CL) [RFC2212]
  CL bietet etwa dieselbe Dienstgüte wie der Best-Effort Dienst in einem unbelasteten Netzwerk. (Hier wird nur Tspec festgelegt.)

- Best-Effort

Wenn Pakete der Dienste CL oder GS nicht konform sind, werden diese wie Best-Effort Pakete behandelt. Nicht konform sind z.B. alle Pakete, welche ausserhalb der vereinbarten Bandbreite liegen

IntServ wird heute in kleineren und mittleren Netzen angewendet, eignet sich jedoch nicht für grosse IP-Netze (Backbone) mit sehr vielen Punkt-zu-Punkt Verbindungen, da der Verwaltungsaufwand (für jeden Flow) zu gross wird und die Speicherkapazitäten der Router nicht mehr ausreichen. Gute Resultate wurden jedoch in grossen IP-Netzen bei Multicastverbindungen erreicht. Die mangelnde Skalierbarkeit und Abrechnungsprobleme von IntServ führten hingegen dazu, dass für grosse Netzwerke von Internet Services Providern (ISPs) neue skalierbare Ansätze weiterverfolgt und entwickelt wurden.

Ein erster Schritt zur Behebung der Skalierungsproblematik ist die ToS-basierte Aggregation, welche auf dem Prinzip beruht, die IP-Pakete, welche zu RSVP-Datenflüssen gehören, im Backbone-Bereich auf ein bestimmtes ToS-Bitmuster abzubilden. Diese mit einem bestimmten ToS-Bitmuster markierten Pakete sollen gemäss den Vorschlägen für das DiffServ-Konzept (siehe Kapitel 1.2.7) gegenüber anders oder gar nicht markierten IP-Paketen bevorzugt behandelt werden.

1.2.6 Anwendungsadaption

Ein weiterer Ansatz zur Handhabung der Dienstgüte in Netzwerken geht davon aus, dass keine durchgängige Ressourcenreservierung zwischen Endsystemen im Internet möglich ist. Deshalb wird beispielsweise versucht, mit geeigneten Protokollen wie z.B. RTP den Anwendungen zu ermöglichen, je nach zur Verfügung stehender Bandbreite die Kompressionsrate der Daten zu variieren und damit in stark belasteten Netzen einen kontinuierlichen Datenstrom aufrechtzuerhalten. Mit diesem Ansatz kann jedoch keine Dienstgütegarantie erreicht werden.

Eine detaillierte Übersicht zu IntServ und zur Anwendungsadaption ist beispielsweise in [IPnG, Seite 116ff] zu finden.

1.2.7 Differentiated Services (DiffServ)

DiffServ [RFC2474][RFC2475][NiBl98] ist ein gelungener Ansatz, welcher Dienstgütenunterstützung im Backbone-Bereich ermöglichen soll. Bei DiffServ wurde ein Verfahren gewählt, bei welchem die Skalierungsprobleme von IntServ prinzipiell nicht auftreten können. Durch Aggregation und Bildung von Dienstklassen müssen in den Netzknoten eines Backbones statt alle Anwendungsdatenflüsse (bis zu mehreren Millionen Flows) höchstens 8 Klassen voneinander unterschieden und bezüglich Dienstgüte unterschiedlich behandelt werden. Dies wird erreicht, indem verlangt wird, dass jedes IP-Paket eine entsprechende Markierung aufweist, welche von DiffServ-Netzknoten berücksichtigt wird. Nebst der deutlich geringeren Anzahl von Klassen müssen für die Dienstgütenunterstützung in den Netzknoten des Backbones keine weiteren Parameter wie Quell- oder Zieladresse oder zusätzliche Merkmale berücksichtigt werden.

Die Markierung der Pakete kann im Kundennetz oder im Border Router des Providers (siehe Abbildung 1.2) erfolgen, womit die Backbone Router stark entlastet werden können. Sind die Netzelemente in einem Backbone einmal konfiguriert, kann der Kunde selbst bestimmen, welche Anwendung bzw. welche Anwendungsdatenflüsse wie markiert werden und demzufolge von welcher Dienstqualität sie profitieren sollen. Insbesondere können mehrere Anwendungen gleichzeitig oder zeitlich aufeinanderfolgende Anwendungsdatenflüsse ohne weitere Signalisierung im Backbone vom entsprechenden Dienst profitieren. Kurzlebige Anwendungsdatenflüsse, die typischerweise bei HTTP [RFC2616] vorkommen, können somit durch eine entsprechende Markierung auch von Dienstgüte profitieren.

Um sicherzustellen, dass Flows eines Aggregates (mit derselben Markierung) sich gegenseitig nicht behindern, ist eine sorgfältige Verwaltung der Netzressourcen mit Admission Control bzw. Shapingmechanismen und entsprechenden Queueing- und Dequeueingmechanismen notwendig. Heute unterstützen viele Netzelemente bereits diese notwendigen Mechanismen zumindest in Beta-Releases der Betriebssysteme.

Zusammenfassend kumulieren sich folgende drei Vorteile von DiffServ gegenüber IntServ im Backbone:

1. Es muss nicht mehr jeder Flow einzeln berücksichtigt werden sondern nur noch 8 Klassen.

2. Es ist keine durchgehende Signalisierung per Flow mehr nötig sondern nur noch per Aggregat.

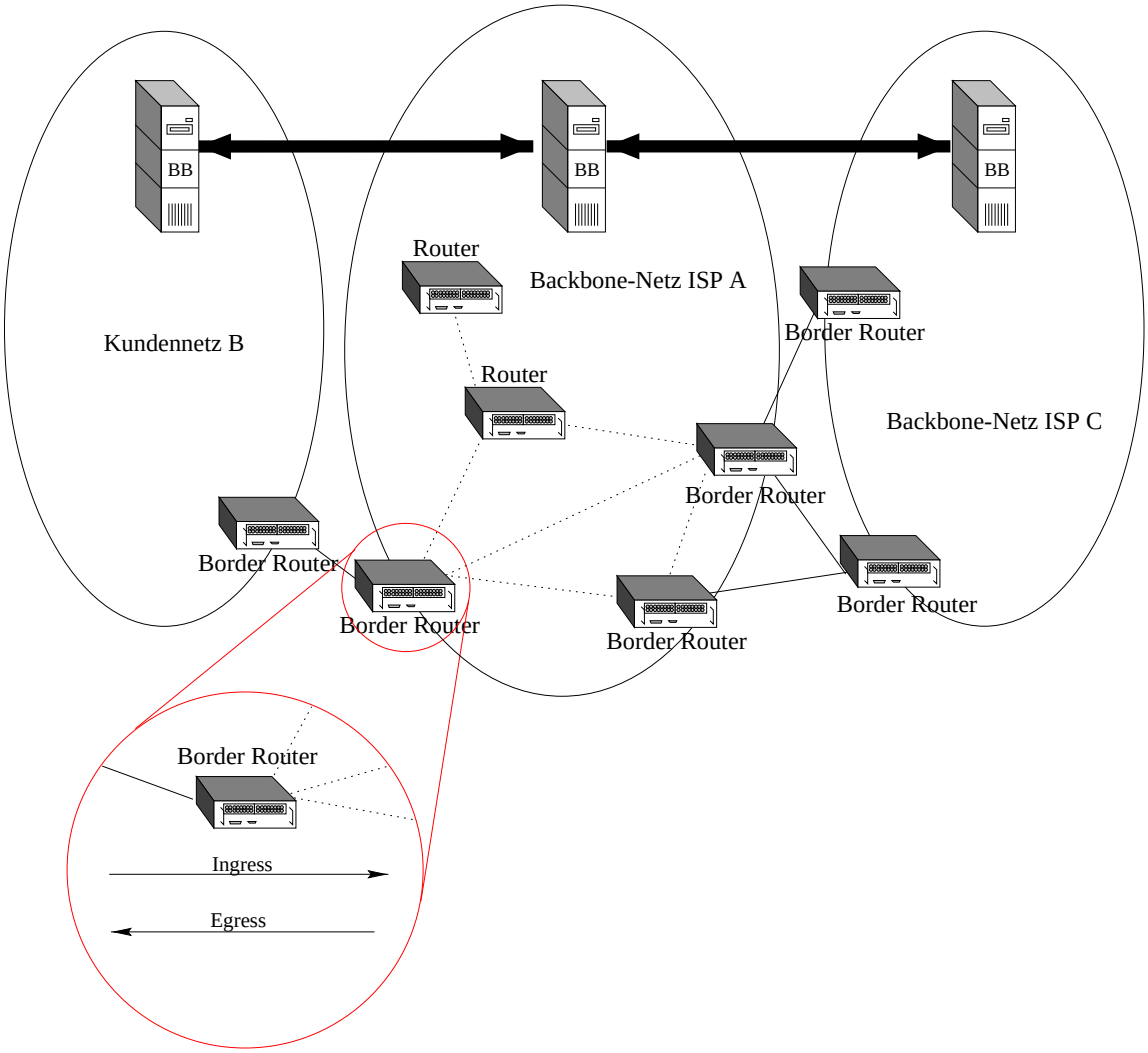


Abbildung 1.2: Komponenten

3. Die Unterscheidung anhand von Merkmalen wie Quell- oder Ziel-Adresse, Quell- oder Ziel-Port oder zusätzlicher Merkmale von Protokollen höherer Schichten erfolgt nicht mehr in jedem Backbone-Netzelement sondern meist einmalig beim Verlassen des Kundennetzes oder beim Eintritt ins Providernetz.

1.2.7.1 Paketmarkierung

Der IP-Header enthält ein sogenanntes ToS-Feld (Type of Service), welches heute häufig DS-Feld genannt wird (DS steht für Differentiated Services). Das DS-Feld ermöglicht nun die geforderte Markierung sowohl in IPv4 wie auch in IPv6 [RFC2474].

Die Markierung der Pakete kann durch die Anwendung selbst oder durch ein Netzelement des Kunden erfolgen. Es ist auch denkbar, die Markierung durch den ersten Router (Ingress Border Router) im Providernetz durchführen zu lassen. Es sind heute schon Produkte auf dem Markt erhältlich, welche den Kunden erlauben, anhand diverser Merkmale Klassen zu bilden und so die IP-Pakete gemäss ihrer Klassifizierung zu markieren. Eine aktuelle Übersicht ist in [ShaperTest] zu finden.

Auf einem Pfad kann die Markierung der Pakete verändert werden, so dass jeder Provider entlang des Pfades prinzipiell eigene Markierungen in seinem Netzwerk verwenden kann. Es gilt lediglich sicherzustellen, dass die Pakete beim Verlassen des Netzwerkes die Markierung aufweisen, welche dem im SLA bestimmten Dienst entsprechen.

1.2.7.2 Dienste

Bei DiffServ werden prinzipiell drei Dienstklassen unterschieden: Premium Service, Assured Services und Best Effort. Diese Dienstklassen werden mittels entsprechender Markierungen der zugehörigen IP-Pakete unterschieden.

Premium Service
Premium Service ist ein Dienst, welcher eine Mietleitung simulieren soll und deshalb höchste Zuverlässigkeit garantieren muss. Deshalb ist diese Klasse durch eine feste Bandbreite charakterisiert. Werden mehr IP-Pakete gesendet als vorgängig durch das entsprechende SLA vereinbart, werden diese weggeworfen (gedropt). Es wird nur ein minimales Queueing durchgeführt, um kleine Lastschwankungen auszugleichen. Premium Service soll dem Garanteed Service von IntServ entsprechen.

Assured Services
Assured Services sind das Analogon zum Control Load Service (IntServ) und liefern nur statistische Garantien einer bestimmten Bandbreite. Hier wird jedoch im Gegensatz zum Premium Service bei Überlastung reklassifiziert und nicht sofort gedropt. In jeder der vier Assured Service Klassen kommen drei Wegwerfwahrscheinlichkeiten (Drop Probabilities) vor, um möglichst ausgeglichen wegwerfen zu können. Diese vier Klassen unterscheiden sich untereinander in der Höhe der statistischen Garantie bezüglich Bandbreite, nicht aber in der Fähigkeit der Reklassifizierung. Ein weiteres Merkmal dieser Dienste ist die Zulassung von Bursts. Damit werden einerseits über kurze Zeit mehr IP-Pakete (sogar ohne Reklassifizierung) zugelassen als im SLA vereinbart und andererseits ermöglicht die Reklassifizierung grössere Bursts.

Best Effort
Best Effort ist der heute bekannte Dienst im Internet, welcher nach dem Motto, „jeder macht das Möglichste“, alle IP-Pakete gleich behandelt.

1.2.7.3 Verwaltung der Netzressourcen bei DiffServ

Policing
Im Ingress Border Router (siehe Abbildung 1.2) ist ein Policing notwendig, um zu überprüfen, ob sich der Dienstnutzer (Customer) an das vereinbarte SLA hält und nicht eine grössere Anzahl entsprechend markierter IP-Pakete bzw. Bytes sendet als das SLA vorsieht.

Shaping
Findet mehr Verkehr statt als im SLA vereinbart, werden insbesondere bei Premium Service überzählige Pakete weggeworfen sobald die Puffer (Queues) voll sind oder aber sobald ein bestimmter Füllgrad erreicht wird. Diese Arbeit erledigt der Shaper, welcher je nach Dienst über grössere oder kleinere Queues und Token Buckets verfügt.

(De)Queueing
Pakete unterschiedlicher Priorität werden in DiffServ-Netzelementen in verschiedene Queues

geleitet. Diese Queues können mit verschiedenen Verfahren geleert werden, so dass eine faire, tolerante und relative Bandbreitenzuweisung zu jeder dieser Queue möglich wird.

Die Gegenüberstellung verschiedener Shaping- und (De)Queueing-Verfahren wurde in der Diplomarbeit von A. Dobreff [DiffServ Simulationen] vorgestellt. Diese DiffServ-Simulationen im IP-ATM Umfeld zeigen die prinzipielle Funktionsweise dieser Verfahren und decken Stärken und Schwächen einiger solcher Verfahren auf.

1.2.7.4 Aufgaben der einzelnen Komponenten

Im DiffServ-Ansatz werden grundsätzlich folgende Komponenten unterschieden:

- Border Router der benachbarten Netzwerke.
- Im Backbone werden die Border Router von weiteren Routern innerhalb des Netzes unterschieden.
- Als weitere Komponenten sind die im CATI-Ansatz vorhandenen Broker als weitere Komponenten zu unterscheiden.

Border-Router im benachbarten Netz
Aus Sicht des DiffServ-Ansatzes ist es unwesentlich, ob Komponenten im benachbarten Netz oder Border Router die notwendige Markierung vornehmen bzw. Markierungen überprüfen. Ist der Border Router des benachbarten Netzes jedoch nicht in der Lage, eine Markierung vorzunehmen oder entsprechende Kontrollen durchzuführen, übernimmt der Border Router des Providers im WAN diese Arbeit. Folglich sind vier Szenarien denkbar:

1. Der Border Router des benachbarten Netzes unterstützt weder IntServ noch DiffServ:
   Der Border Router des WANs übernimmt alle QoS-Aufgaben.
2. Der Border Router des benachbarten Netzes unterstützt IntServ jedoch nicht DiffServ:
   Ein RSVP-DiffServ-Gateway (RDG) [RDG] im WAN übersetzt IntServ-Klassen in DiffServ-Klassen und nimmt entsprechende Aggregationen vor. Markierung und Kontrollen erfolgen beim Provider gemäss eines traditionellen meist papierförmigen SLAs.
3. Der Border Router des benachbarten Netzes unterstützt IntServ und DiffServ:
   Ein RDG im benachbarten Netz übersetzt IntServ-Klassen in DiffServ-Klassen und nimmt entsprechende Aggregationen vor. Damit wird bereits der Link zwischen Customer und Provider mittels DiffServ verwaltet. In diesem Fall muss der Border Router im WAN nur noch die Kontrolle bezüglich Einhaltung des SLAs durchführen. Ist im benachbarten Netz kein entsprechender Service Broker vorhanden, muss auch hier auf traditionelle SLAs zurückgegriffen werden.
4. Der Border Router des benachbarten Netzes unterstützt IntServ und DiffServ und entsprechende Service Broker sind vorhanden:
   Gleiche Aufgaben wie unter 3. zuzüglich Rapportierungen an die Broker. Dies sollte künftig der Normalfall sein.

Diese vier Szenarien werden in Kapitel 3.3 am Beispiel des Bandwidth Brokers näher erläutert.

Router im WAN
Die Aufgaben der Router im WAN unterscheiden sich je nachdem, ob es sich um Ingress Router oder Egress Router handelt. Meistens ist ein Border Router sowohl Ingress wie auch Egress Router.

- Aufgaben des Ingress Routers:
  In jedem Fall überprüft ein Ingress Router die im entsprechenden Dienst vereinbarten Parameter und behandelt die IP-Pakete bereits gemäss ihrer Markierung (Policing). Je nachdem ob der zugehörige Border Router des benachbarten Netzes DiffServ-fähig ist oder nicht, kommen noch weitere Aufgaben dazu (siehe oben, Szenarien 3 und 4).
- Aufgaben des Egress Routers:
  Wie beim Ingress Router werden die IP-Pakete gemäss ihrer Markierung behandelt und eventuell nochmals die vereinbarten Parameter überprüft. Falls im Ingress Router kein Shaping stattgefunden hat, muss ein Shaping an dieser Stelle erfolgen. Diese Überprüfung soll Service Brokern zusätzliche Kontrollen ermöglichen.
- Weitere Netzelemente im WAN:
  Die Netzelemente im WAN müssen DiffServ unterstützen und IP-Pakete entsprechend ihrer Markierung behandeln. Wenn nun ein Provider eine Netztechnologie einsetzt, welche QoS auf Layer 2 unterstützt, besteht die Möglichkeit, diese Layer 2-Funktionalität zu verwenden. Eine Umsetzung von IP QoS-Parameter auf entsprechende ATM-Parameter ist in [DiffServ Simulationen] ersichtlich. Die Umsetzung von DiffServ-Parametern auf QoS-Parameter des Layers 2 übernimmt gemäss Broker Hierarchie (siehe Kapitel 2.1) ein Internal Service Broker (ISB).

Service Broker
Die Aufgaben der verschiedenen Service Broker (siehe Kapitel 2.1) wird in Kapitel 3 für den Spezialfall des Bandwidth Brokers näher erläutert. Generell werden Service Broker zur Koordination und zur Kontrolle von Konfigurationen aller zugeordneten Netzelemente verwendet, um abstrakte Dienste anzubieten.

1.2.8 IntServ-DiffServ-Integration

Die IntServ-DiffServ-Integration geschieht durch Zuordnung der IntServ-Klassen zu DiffServ-Klassen und entsprechender DiffServ Markierung der Pakete. Die Signalisation per Flow muss aggregiert werden und in entsprechende DiffServ-Signalisierung umgewandelt werden. Die Diplomarbeit von Roland Balmer [RDG] befasst sich ausführlich mit diesem Thema und stellt einen Prototypen eines RSVP-DiffServ-Gateway (RDG) vor, welcher im Rahmen des bereits erwähnten CATI-Projektes angewendet wird und dessen Konzept in [IntServ - DiffServ] beschrieben wird.

1.2.9 Weitere Entwicklung von Differentiated Services

Nebst dem Ansatz, die Ressourcen in einem Netzwerk mittels Brokern zu verwalten, werden unabhängig davon Konzepte zur dynamischen Dienstkonfiguration via Protokolle wie RSVP

oder COPS weiterverfolgt. Die Unterstützung von Multicast durch DiffServ ist ebenso ein Thema wie die konsequente Verwendung der QoS-Fähigkeiten der Schichten unterhalb der IP-Schicht.

1.2.10 Weshalb Brokering?

ISPs wollen mit dem Transport von Inhalten auf ihren Netzwerken Geld verdienen können. Ein Link zwischen zwei Providern ist zudem mehrheitlich in der einen Richtung wesentlich mehr belastet als in der Gegenrichtung. Heute sind SLAs eher statischer Natur und bedingen den Einsatz von Netzwerkspezialisten, welche häufig Netzwerkkonfigurationen von Hand vornehmen und ein breites Wissen über Netzwerkprotokolle besitzen müssen. Die dazu verwendeten Protokolle und Mechanismen sind ausserdem meist nicht dazu geeignet, dynamische Konfigurationen vorzunehmen und auch nicht in der Lage, eine detaillierte Abrechnung zu ermöglichen. Die Kontrolle auf Management-Ebene erfordert Abstraktionsmechanismen, welche Brokering voraussetzen. Die Tunnel-basierte Aggregation mittels RSVP [IPnG, Seite 139] ist ein Beispiel für eine solche unvollständige und deshalb unbefriedigende Lösung des Dienstgüteproblems im Internet aus Managementsicht.

In Zukunft ist es unumgänglich, den kommerziellen Aspekt mehr in den Vordergrund zu rücken, um Managern zu ermöglichen, die vorhandenen Netzressourcen als Handelsgut zu betrachten. Das in Kapitel 2.1 vorgestellte Service Broker Modell soll durch die vier Abstraktionsstufen im Composite Service Server Layer ein Abstraktionsniveau erreichen, welches Managern ermöglicht, ohne Netzwerkspezialistenwissen das Management des Netzwerkes weitgehend autonom vorzunehmen, wobei entsprechende Automatismen einen grossen Teil der Koordinationsarbeit übernehmen.

Kapitel 2

Modelle und Konzepte

2.1 Service Broker Modell für DiffServ

Das allgemeine Service Broker Modell basiert auf der in Abbildung 2.1 ersichtlichen Broker Hierachie, welche in [VPN Architecture] vorgeschlagen wurde.

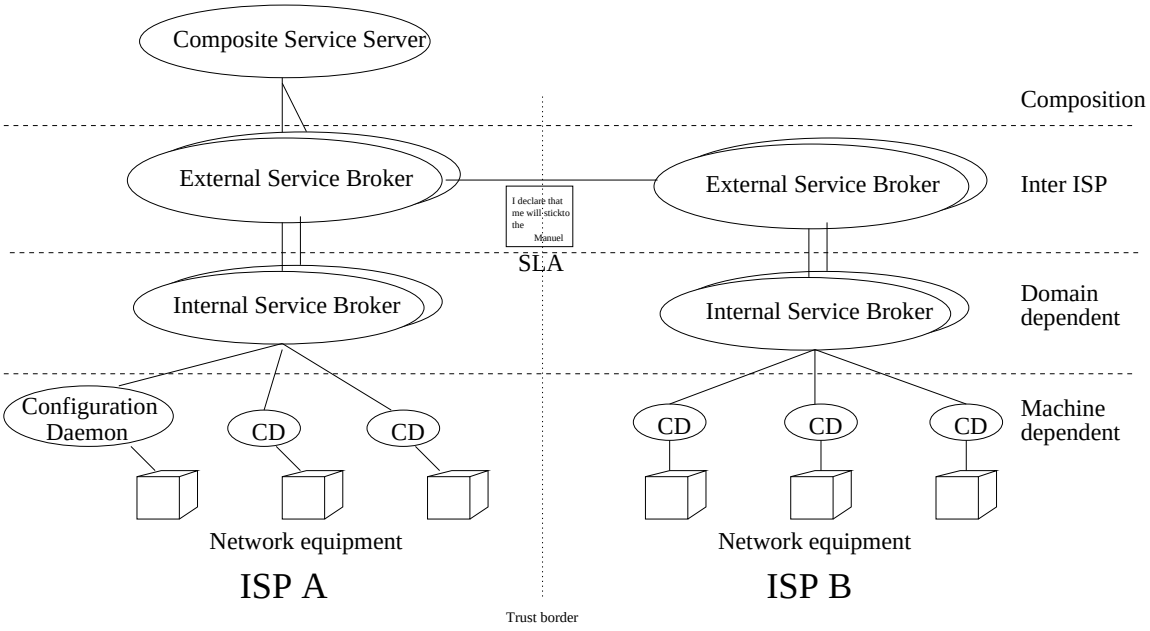


Abbildung 2.1: Broker Hierarchie

Die Broker Hierarchie setzt sich aus Broker Instanzen zusammen, welche folgenden vier Schichten zugeteilt werden können:

1. Composition Layer
   Instanzen dieser Schicht werden Composite Service Server (CSS) genannt und haben die Aufgabe, kombinierte Dienste von Instanzen der Schicht 2 anzubieten. CSS nutzen dazu allgemein definierte Schnittstellen der Instanzen der Schicht 2.

2. Business Layer
   Instanzen der Schicht 2 werden External Service Broker (ESB) genannt, da ihre Hauptaufgabe darin besteht, domainübergreifende Dienste anzubieten. Über eine allgemein

definierte Schnittstelle (siehe Kapitel 2.3) sind sie in der Lage, mit ESBs anderer Domains zu kommunizieren. Dazu können ESBs ebenso über eine allgemein definierte Schnittstelle von Instanzen der Schicht 3 der eigenen Domain angesprochen werden.

3. Domain Dependent Layer
   Instanzen der Schicht 3 werden Internal Service Broker (ISB) genannt, weil diese ausschliesslich Aufgaben erfüllen, welche innerhalb der Domain liegen, in welcher sie sich befinden. So bietet jeder ISB den ESBs der eigenen Domain Dienste an, welche unter Zuhilfenahme von Instanzen der Schicht 4 erfüllt werden können.

4. Machine Dependent Layer
   Instanzen der Schicht 4 werden Configuration Daemon (CD) genannt. Ihre Aufgabe besteht darin, eine Netzkomponente zu abstrahieren und somit den ISBs eine einheitliche Schnittstelle anzubieten. Hier sollen also herstellerspezifische Befehle oder Konfigurationsweisen in allgemeine Konzepte umgewandelt werden. Die Überwachung der Netzelemente gehört ebenso zur Aufgabe von CDs, welche einen Ausfall den betroffenen ISBs mitteilen oder eine diesbezügliche Anfrage eines ISBs beantworten können.

Beispiel:
Ein CSS hat die Aufgabe, Dienste von ESBs zu kombinieren und damit z.B. einen VPN-Service mit QoS-Garantie anzubieten.

Ein beteiligter ESB ist der External VPN Broker (EVB), wovon auf jeder Seite eines VPN-Tunnels eine Instanz vorhanden ist. Diese Instanzen kommunizieren verschlüsselt domainübergreifend durch andere Domains hindurch. Der zweite beteiligte ESB ist der External Bandwidth Broker (EBB), welcher zwei Aufgaben wahrnimmt: Erstens kommuniziert er mit direkt benachbarten EBBs, welche wiederum mit ihren benachbarten EBBs den gesamten Pfad eines VPN-Tunnels abzudecken versuchen. Die zweite Aufgabe besteht darin, die benötigten Dienste von ISB so zu nutzen, dass alle Netzkomponenten der eigenen Domain so konfiguriert werden, dass der gewünschte Dienst erbracht werden kann. Es ist denkbar, in diesem Beispiel einen weiteren ESB, einen External Delay Broker (EDB) zu nutzen, um nebst der Bandbreite auch die Verzögerungen verwalten zu können, welche bei Echtzeitanwendungen wie z.B. der Telefonie massgebend sind.

Die beteiligten ISB haben einerseits die Aufgabe, den ESBs Parameter von Netzkomponenten mitzuteilen und andererseits den CDs Konfigurationsanforderungen weiterzuleiten. Im vorliegenden Beispiel müssten dem EBB die effektiv vorhandenen Bandbreiten mitgeteilt werden und dem EVB die Netzelemente, welche in der Lage sind, Tunnel-Endpunkte zu realisieren. Die Konfigurationsanforderungen werden dann, nach der Aushandlung von SLAs durch die ESB, an die zuständigen ISB gestellt, welche die Konfigurationen über die CDs vornehmen.

2.2 Two Phase Commitment

Die Kommunikation zwischen den verschiedenen Instanzen des Service Broker Modells erfolgt in mehreren Schritten. Insbesondere kommen SLAs zustande, indem mindestens zwei Broker-Instanzen signierte Meldungen austauschen, die für den betroffenen Dienst die notwendigen Parameter enthalten.

Der Austausch von Parametern geschieht im Wesentlichen in zwei (drei) Schritten:

I. Mittels QUERYs werden sowohl Struktur wie auch Inhalt von Objects (siehe Kapitel 2.3) ausgehandelt.

II. Mittels REQUESTs werden dann entsprechende Dienste aufgesetzt oder die Ressourcen für diese Dienste reserviert, ohne dabei die Nutzung der Dienste zuzulassen. Requests sind bereits kostenpflichtig, da der Dienstanbieter Ressourcen reserviert bzw. freihält. Dies ist der 1. Schritt des „two phase commitments“. Er ermöglicht ISPs, vorsorglich Ressourcen zu anderen ISPs zu reservieren und erwartete Anfragen seiner Kunden deshalb schnell beantworten zu können, ohne bei jeder Anfrage zuerst selbst Request-Anfragen an benachbarte Provider stellen zu müssen.

   Beispiel:
   Nach einem entsprechenden REQUEST kann bei einem VPN-Dienst ein Tunnel bereits aufgesetzt werden, die Pakete jedoch noch nicht durch den Tunnel hindurch geleitet werden. Bei DiffServ-Diensten können die Shaping- und Queueing-Mechanismen bereits eingestellt, die Markierungen der Pakete des entsprechenden Kunden (der entsprechenden Quelle) jedoch bis zum entsprechenden COMMIT gelöscht oder nicht beachtet werden.

III. Mittels COMMITs werden die bereits teilweise aufgesetzten Dienste zur Nutzung freigeschaltet. Der Dienstanbieter beginnt hiermit, den Netzverkehr zuzulassen und die Einhaltung des SLAs entsprechend zu überwachen. Bei DiffServ können ab diesem Zeitpunkt die Markierungen der Pakete, welche von einer bestimmten Quelladresse oder von einem bestimmten Port stammen, entsprechend ihrer Markierung behandelt werden. Wenn die Pakete noch nicht markiert sind, kann eine Markierung erfolgen. Der ISP kann somit sowohl eine rein zeitabhängige Pauschale oder aber eine nutzungsabhängige Verrechnung oder eine Kombination davon vorsehen. Mit diesem Schritt wird das „two phase commitment“ abgeschlossen und ein SLA voll wirksam

Schritt I. ist nicht in jedem Fall erforderlich, da unter Umständen die gewünschte Struktur wie auch der Inhalt der Objects - z.B. der Preis eines Dienstes - bereits bekannt ist. Ebenso ist Schritt II. nicht zwingend, weil der 2. Teil der Phase (Schritt III.) in mehrere COMMITs aufgesplittert werden kann. Nach einem ersten REQUEST können mehrere COMMITs folgen, welche jeweils nur einen Teil der im aufgesetzten Dienst reservierten Ressourcen freischalten.

2.2.1 Dynamische Beziehung zwischen SLAs

2.2.1.1 Dynamische Beziehung entlang eines Pfades

Anhand des folgenden Beispiels soll deutlich werden, dass im DiffServ-Umfeld die statische Beziehung zwischen Requests (bzw. Commits) entlang eines Pfades nicht notwendig ist und dass sich die dynamische Koordination von Requests (bzw. von Commits) mit der Paketmarkierung sehr gut ergänzt. In Abbildung 2.2 sind zwei Links in der Domain von ISP A rot hervorgehoben. Diese zwei Links sollen SLAs desselben Dienstes mit zwei verschiedenen Kunden darstellen, welche somit IP-Pakete mit derselben Markierung aufweisen und das Netzwerk von ISP A durch denselben Link verlassen. Auf dem Inter-Domain-Link zu ISP C ist nur noch ein SLA mit mindestens der kumulierten Bandbreite notwendig. Nebst der Aggregation von Datenflüssen kann damit eine Aggregation von Signalisierungsverkehr bei jedem ISP erfolgen.

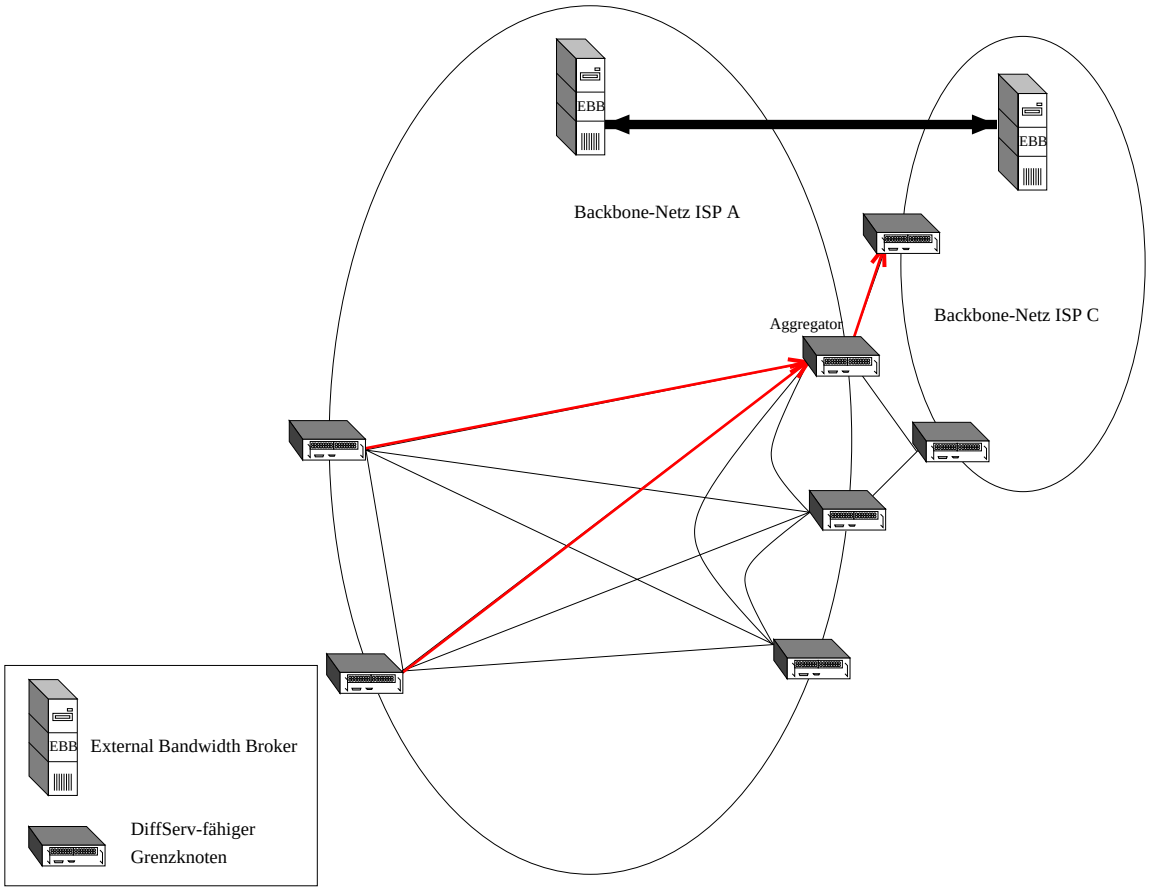


Abbildung 2.2: Aggregation von Datenflüssen und von Signalisierungsverkehr.

Ein Provider kann auf all seinen Inter-Domain-Links bereits Request-SLAs aufsetzen, ohne entsprechende Kundenanfragen zu haben und diese sukzessive mit Kundenanfragen 'füllen'. Damit kann ein ISP vorsorgen, indem er z.B. bei einer 90%-igen Auslastung seiner Request-SLAs durch Kunden-Request-SLAs, die eigenen Request-SLAs aufstockt. Mit Commit-SLAs kann ein ISP ähnlich umgehen, wobei die Zusammenhänge zwischen eigenen Requests, eigenen Commits, Kunden-Requests, Kunden-Commits oder die aktuelle bzw. erwartete Netzlast berücksichtigt werden müssen.

2.2.1.2 Dynamische Beziehung zwischen Requests und Commits

Die Dimensionierung von vorsorglichen Request-SLAs kann ein ISP davon abhängig machen, welche Menge an Kunden-Requests bereits durch entsprechende Kunden-Commits abgedeckt wurden. Statt Kunden-Commits kann auch der effektive Verkehr oder eine Kombination von Kunden-Commits und effektivem Verkehr betrachtet werden. Durch diese dynamische Beziehung zwischen Requests und Commits haben ISPs viel Spielraum in der Gestaltung der im Zusammenhang mit DiffServ-Diensten angebotenen Services.

2.3 Broker Signaling Protocol (BSP)

Das Konzept eines Broker Signaling Protokolls (BSP) wird in [CATI BSP] beschrieben. Im Wesentlichen wird in dieser Arbeit dieses Konzept verwendet. Dabei werden Verfeinerungen vorgenommen und neue Aspekte (neue BSP-Elemente) eingeführt.

2.3.1 Wichtige Definitionen

Meldungen, welche zwischen Brokern ausgetauscht werden, werden *Messages* genannt.

Messages sind *Objects* mit einem eindeutigen Identifier.

Je zwei Broker sind an einem Message-Austausch beteiligt, wobei der Dienstanbieter *Provider* und der Dienstnutzer *Customer* genannt wird.

2.3.2 Aufgabe

Das BSP-Protokoll soll einer Broker-Instanz (Customer) ermöglichen, ein SLA mit einer anderen Broker-Instanz (Provider) auszuhandeln, ohne die genaue Struktur der Messages zu kennen, welche für einen bestimmten Dienst notwendig sind. Die Bekanntgabe der Struktur erfolgt durch den Provider aufgrund einer allgemeinen Anfrage. Ist die Struktur bekannt, kann vom Provider eine Auswahl von zulässigen Werten der in dieser Struktur vorhandenen Parameter erfragt werden. Damit kann der Customer aus einer Auswahl von Parametern die gewünschte Kombination zusammenstellen und eine weitere Anfrage an den Provider stellen.

2.3.3 Hierarchie

In der Abbildung 2.3 sind alle BSP-Elemente in der Objekthierarchie des OO-Design ersichtlich:

- Pfeile bedeuten Vererbung (isA).
- Die Zahl in der Wolke gibt die Anzahl der möglichen Instanzen an (0 bei abstrakten Klassen, n bedeutet beliebig viele).
- Die weiteren Verbindungen (hasA) haben auf einer Seite eine Zahl, welche die Anzahl Instanzen der Klasse angibt.

Beispielsweise kann eine Ordered Collection n Objects haben. Eine Ordered Collection ist eine Collection, welche eine Entity ist.

2.3.4 Protokollelemente und deren BSP-Syntax

Alle Elemente des Protokolls haben eine binäre Darstellung (BSP-Syntax), welche grundsätzlich folgendem Schema entspricht:

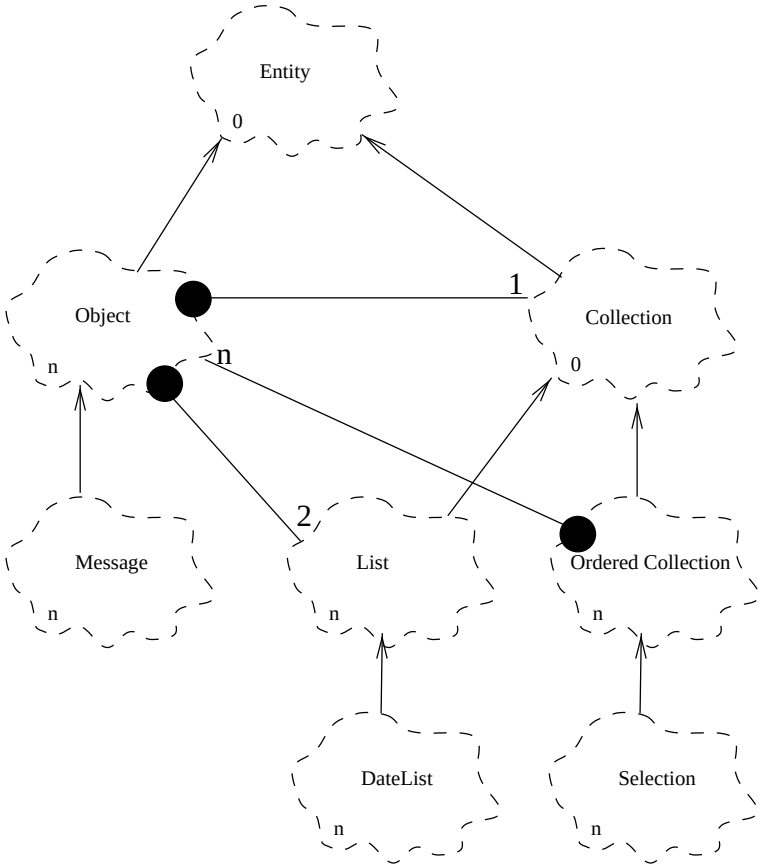


Abbildung 2.3: Protokoll Entitätenhierarchie

$$\{XY_1 \cdots Y_n\}$$

Dabei definiert X die Klasse und entspricht einem eindeutigen Kürzel gemäss folgender Liste:

- O für **O**bject
- M für **M**essage
- L für **L**ist
- C für Ordered **C**ollection
- S für **S**election
- D für **D**ateList

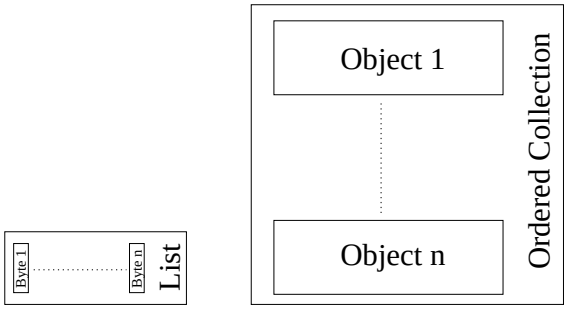
Wofür $Y_1 \cdots Y_n$ steht, wird in den folgenden Abschnitten erläutert. Dabei ist zu beachten, dass aus Kompatibilitätsgründen bzw. wegen des internationalen Charakters eines Protokolls genormte Darstellungen zur Anwendung kommen.

2.3.4.1 Abstrakte BSP-Elemente

Entity
Eine Entity ist die Mutter aller Elemente der BSP-Hierarchie und bestimmt die allgemeine Darstellung wie bereits erwähnt:

$$\{XY_1 \cdots Y_n\}$$

Collection
Eine Collection ist ein abstraktes BSP-Element. Wie in Abbildung 2.4 ersichtlich, besteht eine Collection entweder aus einer List oder einer Ordered Collection. Eine List ist eine Liste



(a) Bytes-Collection (b) Objects-Collection

Abbildung 2.4: Struktur einer Collection

von Bytes, während eine Ordered Collection eine Liste von Objects ist. Mit diesem Konstrukt entsteht eine Art Baum, deren Knoten Objects und deren Blätter Lists sind.

2.3.4.2 Konkrete BSP-Elemente

Objects und Messages
BSP-Repräsentation:

{OF{LH9}{LAName}{LAObject}}

Abbildung 2.5: Struktur eines Objects

Im Falle von Objects und Messages ist n=4 für folgende vier Felder:

1. rot: Status gemäss Auflistung in Kapitel 2.3.6.
2. grün: Eine List als Identifier (bei Messages eindeutig für jede Broker-Instanz).
3. blau: Eine List als Name gemäss Kapitel 2.3.5.

4. gelb: Eine Collection.

Eine Message ist ein Object mit einer Identifikationsnummer, welche durch den Erzeuger der Message eindeutig vergeben wird. Der Identifier ermöglicht entweder zusammen mit dem Object Customer oder zusammen mit dem Object Provider (bei Replies), welches in der Collection jeder Message vorkommen muss, eine weltweit eindeutige Identifizierung jeder Message. Die Verwendung des Identifiers in Objects unterscheidet sich somit von der Definition in [CATI BSP], welche auch für Objects eine eindeutige Identifikationsnummer vorsieht. Die Vergabe einer eindeutigen Identifikationsnummer für jedes Object macht nur dann Sinn, wenn die Vergabe von Identifikationsnummern weltweit koordiniert wird, da nicht immer festgestellt werden kann, welcher Broker welches Object erzeugt hat. Eine solche weltweite Vergabe von Identifikationsnummern erscheint mir nicht sinnvoll, weil bei Objects keine offensichtliche Notwendigkeit besteht, jedes Object identifizieren zu müssen.

List und DateList
BSP-Repräsentation:

$$\{LY_1Y_2\} \ bzw. \ \{DY_1Y_2\}$$

Bei List und DateList gibt Y_1 (ein ASCII-Byte) den Typ bzw. die Codierung gemäss folgender Liste an:

- A für **A**SCII (Text)
- H für **H**exadezimal (integer, long)
- E für **E**xponential (real, double)
- I für **I**P (Stringrepräsentation, siehe Beispiele)

Y_2 ist der Inhalt der List und reicht bis zur nächsten schliessenden geschweiften Klammer und wird entsprechend der Codierung interpretiert.

Beispiele:

ASCII	{LAText, vor geschweiften Klammern muss ein \\ stehen: \{}
Hexadezimal	{LH3f46}
Exponential	{LE3e45}
IP	{LI123.123.123.123/255.255.255.0}

Das Datum ist eine eigene Klasse (DateList), da die Codierung aus Kompatibilitätsgründen in Millisekunden seit dem 1.1.1970 0 Uhr 00 GMT erfolgt. Die Millisekunden werden als Datentyp long und somit Hexadezimal gespeichert.
BSP-Repräsentation:

{DHdd3619d4f8}

Diese von der Definition in [CATI BSP] abweichende Definition ist notwendig, um Sprachunabhängigkeit bzw. Internationalisierung zu erreichen.

Ordered Collection und Selection
BSP-Repräsentation:

$$\{CY_1 \ldots Y_n\} \ bzw. \ \{SY_1 \cdots Y_n\}$$

Die Y_i stehen für eine Liste von n Objects.

Beispiele:

Ordered Collection	{C {OP{LH0}{LAamount}{LH8}} {OF{LH0}{LAunit}{LAsFr}} }
Selection	{S {OF{LH0}{LAca}{LASwiss Key}} {OF{LH1}{LAca}{LAUniKey}} }

Ein Object mit einer Collection könnte folgendermassen aussehen:

{OP{LH0}{LAbandwidth}{C
{OP{LH0}{LAamount}{LH8}}
{OF{LH0}{LAunit}{LAsFr}}
}}

Die Definition einer Selection weicht von der Definition in [CATI BSP] ab. Ursprünglich wurde eine Selection als Object-Zustand definiert. Bei Objects, welche als Collection eine List haben, macht der Zustand 'selection' wenig Sinn. Im Unterschied zu allen anderen Zuständen (siehe Kapitel 2.3.6) ist 'selection' an eine Struktur der Collection des Objects und eine Interpretation dieser Struktur gebunden. Dies wurde verdeutlicht indem eine Selection als eigenständiges BSP-Element definiert wurde. Da die Struktur einer Selection beinahe identisch ist mit der Struktur einer Ordered Collection, wurde die Selection auch davon abgeleitet.

2.3.5 Normierte BSP-Namen

Die folgende Auflistung normierter BSP-Namen enthält alle Konstanten der Klasse Protocol-Constants.java und deren BSP-Syntax.

2.3.5.1 Messages

- QUERY {LAquery}
  Eine Query ist eine einfache Anfrage. Der Status einer query ist immer „please complete“.
- REQUEST {LArequest}
  Ein Request ist eine Anforderung von Ressourcen, welche Kosten nach sich zieht. Der Status von Requests ist immer „fixed“.

- COMMIT {LAcommit}
  Ein Commit setzt ein entsprechendes Request voraus und dient zur Freischaltung des entsprechenden Dienstes. Der Status eines Commits ist immer „fixed“.

- CANCEL {LAcancel}
  Cancels werden verwendet, um aktive Request-SLAs oder Commit-SLAs aufzuheben. Auch hier wird der Status „fixed“ verlangt.

- REPLY {LAreply}
  Ein Reply ist eine Antwort auf eine erfolgreich bearbeitete Query-, Request-, Commit- oder Cancel-Message.

- SIGNAL {LAsignal}
  Eine Signal-Message ist dazu vorgesehen, bei der ISB-ESB Kommunikation dem ISB zu ermöglichen, Signal-Meldungen abzusetzen, ohne eine Antwort darauf zu erwarten.

- ENCRYPTED {LAencrypted}
  Diese Message enthält eine verschlüsselte Message eines anderen Typs in der Collection und dient der Identifizierung, ob eine verschlüsselte oder unverschlüsselte Message empfangen wird. Status: „fixed“.

- ERROR {LAerror}
  Error-Messages werden verwendet, wenn eine Anfrage nicht erfolgreich bearbeitet werden konnte, sei dies wegen fehlerhaften Strukturen oder Übermittlungsfehlern.

2.3.5.2 Objects

- ERROR_MESSAGE {LAerror_message}
  Ein Error_message-Object ist eine textuelle Beschreibung der Ursache für eine Error-Message.

- CUSTOMER {LAcustomer}
  Dieses Object beschreibt den Dienstnutzer und muss normiert sein. Denkbar sind analoge Beschreibungen wie bei URL, z.B. esb.domain.com oder sb.domain.com (sb=Service Broker); eine entsprechende Unterstützung durch DNS vorausgesetzt. Der Customer muss nicht in jedem Fall identisch sein mit dem Encryption-Owner. Status: „fixed“.

- PROVIDER {LAprovider}
  Analog zu Customer. Hier ist in speziellen Fällen denkbar, einen Status „please complete“ zuzulassen. Üblicher Status: „fixed“.

- SERVICE {LAservice}
  Dieses Object beschreibt den gewünschten Service in einer zu normierenden Codierung.

- INGRESS {LAingress}
  Dieses Object beschreibt den Eingangspunkt von IP-Paketen in einem Netzwerk und bestimmt somit die Richtung eines Dienstes.

- EGRESS {LAegress}
  Dieses Object beschreibt den Austrittspunkt von IP-Paketen aus einem Netzwerk und bestimmt somit die Richtung eines Dienstes.

- DESCRIPTION {LAdescription}
  Dieses Object beschreibt den Service durch entsprechende Parameter.
- PRICE {LAprice}
  Dieses Object beschreibt den Preis eines Dienstes.
- PAYMENT {LApayment}
  Das Payment-Object gibt die Art der Zahlung an: z.B. mit Billing.
- START {LAstart}
  Das Start-Object gibt Datum und Zeit der Aufschaltung eines Dienstes an.
- END {LAend}
  Das End-Object gibt Datum und Zeit der Abschaltung eines Dienstes an.
- CA {LAca}
  Falls eine Message nicht verschlüsselt wird, wird hiermit die Certification Authority angegeben, welche die Signatur überprüfen kann.
- SIGNATURE {LAsignature}
  Digitale Unterschrift.
- ENDPOINT {LAendpoint}
  Dieses Object gibt den Endpunkt eines Dienstes an, welcher nicht zwingend im Netzwerk des Providers liegen muss.
- REFERENCE {LAreference}
  Dieses Object enthält die Referenz (Identifier einer Request) und kommt in jeder Commit-Message vor.
- VALUE_AMOUNT {LAamount}
  In diesem Object steht der Betrag eines strukturierten Objects.
- VALUE_UNIT {LAunit}
  In diesem Object steht die Einheit eines strukturierten Objects. Dieses Object wird meist im Zusammenhang mit Amount verwendet.
- SERVICE_ADDRESS_ADDRESS {LAaddress}
  Dieses spezielle Object soll es ermöglichen, eine IP-Adresse in spezieller Notation (mit Wildcards) darzustellen.
- SERVICE_ADDRESS_MASK {LAmask}
  Object zur Darstellung der Maske einer IP-Adresse. Siehe SERVICE_ADDRESS_ADDRESS.
- CANCEL_REASON {LAreason}
  Textuelle Begründung in einer Cancel-Message.
- KEY {LAkey}
  Schlüsselwert zur Berechnung der Signatur.
- ENCRYPTION {LAencryption}
  Die Collection dieses Objects enthält die Binärdaten einer verschlüsselten Message.

- ENCRYPTION_TYPE {LAencryptionType}
  Dieses Object gibt an, um welchen Verschlüsselungstyp es sich handelt. Derzeit werden PGP und MD5 unterstützt.

- ENCRYPTION_OWNER {LAencryptionOwner}
  Gibt in einer Encrypted-Message die Identität des Autors bekannt.

2.3.5.3 Normierte Lists

- EF long
  Beschreibt einen Service.

- KEY_ID long
  Identifier eines Key-Objects.

- DATE_TIME_FORMAT
  Java Date-Format.

2.3.5.4 Interne Objects

BSP-Objects werden ohne Umsetzung in eine andere Datenstruktur in allen Komponenten des Brokers konsequent weiterverwendet. Die Gültigkeit bzw. Normierung dieser BSP-Namen beschränkt sich somit auf einen Broker und werden deshalb intern genannt.

- TIMESTAMP
  Empfangs- oder Sendezeit einer Message.

- SOCKET
  Java-Socket Informationen.

- CLIENT_NAME
  DNS-Name des Verbindungspartners.

- CLIENT_IP
  IP-Adresse des Verbindungspartners.

- CLIENT_PORT
  Port-Nummer des Verbindungspartners.

- NAME
  Eigener lokaler DNS-Name.

- IP
  Eigene IP-Adresse.

- PORT
  Lokaler Port einer Verbindung.

2.3.6 Object-Zustände

In der folgenden Auflistung werden die Begriffe „Anfrage“ und „Antwort“ verwendet.

Anfrage	Antwort
Query	Reply
Request	Error
Commit	
Cancel	

Der Status eines Objects bestimmt die Interpretation der Collection. Dabei kann eine Collection überprüft, ignoriert, ergänzt oder ersetzt werden.

Es sind folgende Object-Zustände (Stati) definiert:

- FIXED
  Fixed bedeutet, dass sowohl Struktur des Objects als auch Inhalt der Object-Collection nicht mehr verändert werden sollen.
- PLEASE_COMPLETE
  Der Status „please complete“ in einer Anfrage bedeutet, dass der Sender wünscht, dass die Struktur oder der Inhalt eines Objects durch den Empfänger ersetzt, ergänzt bzw. vervollständigt wird. Fügt ein Empfänger ein Object in eine Collection neu ein, wird nur eine mögliche Struktur nicht aber eine Auswahl von Strukturen vorgeschlagen. Der Sender hat dann die Möglichkeit, nach Erhalt der Antwort auf die erste Anfrage, durch die erneute Angabe des Status „please complete“ mögliche Strukturvarianten bzw. Inhalte zu erfahren (siehe Beispiele in Kapitel 3.3.3).
- ERRONEOUS
  Dieser Status in einer Antwort gibt an, dass die Struktur oder der Inhalt fehlerhaft ist oder durch den Empfänger nicht verarbeitet werden kann.
- UNKNOWN
  Dieser Status in einer Antwort gibt an, dass die Struktur dem Empfänger unbekannt ist oder durch den Empfänger nicht verarbeitet werden kann.
- INFORMATIONAL
  Dieser Status kann für Hinweise verwendet werden.
- OMITTED
  Gibt den Hinweis, dass das Object überflüssig und somit nicht beachtet wird.

Kapitel 3

Bandwidth Broker

Im Rahmen des CATI-Projektes[1] wird in [VPN Architecture] und [QoS and VPN] das Konzept eines Bandwidth Brokers (BB) vorgestellt. Dieses Konzept baut auf Vorschlägen des Internet Drafts 'A Two-Tier Resource Management Model for Differentiated Services Networks' [Two-Tier] und auf das RFC 2638 'A Two-bit Differentiated Services Architecture for the Internet' [Two-bit] auf. Bandwidth Broker sollen es ermöglichen, automatisches Bandbreitenmanagement im DiffServ-Umfeld zu realisieren. DiffServ basiert auf der Aggregation von IP-Flows in Service-Klassen. Durch eine entsprechende Markierung im IP-Header wird jedes IP-Paket einer Service-Klasse zugeordnet. Somit können alle Netzelemente IP-Pakete entsprechend ihrer Klassierung unterschiedlich behandeln. Durch geeignetes Management wird für jeden Link bestimmt, welche Bandbreite für welche Klasse zur Verfügung stehen soll. Dieses Management soll mittels BBs auf dynamische Weise erfolgen, indem für jeden Link zwischen zwei ISPs entsprechende elektronische SLAs für jede Service-Klasse ausgehandelt werden können. An Ein- und Austrittspunkte eines Netzwerkes wird mittels Policing und Shaping die Einhaltung dieser SLAs überprüft. Innerhalb des eigenen Netzes kann der ISP die DiffServ-Markierungen der IP-Pakete verwenden, um auch im eigenen Netz DiffServ-Bandbreitenmanagement zu betreiben.

Das Skalierungsproblem von IntServ im Backbone wird einerseits dadurch vermieden, dass bei DiffServ der Verwaltungsaufwand in den Netzelementen nicht für jeden Flow sondern nur für eine kleine Anzahl von Klassen anfällt, und andererseits dadurch, dass keine durchgängige Ressourcenreservierung per Flow erfolgt, sondern in jeder Domain ein BB für die Ressourcenreservierung per Aggregat (Klasse) zuständig ist.

3.1 Aufgaben eines Bandwidth Brokers

Ein BB hat die Aufgabe, die Ressourcen in seiner Domain und die Links zu benachbarten Domains zu verwalten, indem er das Verhalten der Router steuert. Innerhalb der eigenen Domain verfügt der BB über die Freiheit, QoS auf eine proprietäre Weise zu gewährleisten, so kann er QoS-Mechanismen des Layers 2 zu Hilfe nehmen oder aber innerhalb des eigenen Netzwerkes QoS mittels DiffServ unterstützen. Die Verwaltung der Links zu benachbarten Domains basiert auf einer DiffServ-Markierung der IP-Pakete und auf der Koordination des

[1] T. Braun, B. Stiller et al.: Charging and Accounting Technologies for the Internet; http://www.tik.ee.ethz.ch/~cati/; 1999.

Verhaltens der miteinander verbundenen Border Router der eigenen und der dieser benachbarten Domain. Die notwendige Koordination wird durch bilaterale Vereinbarungen (Service Level Agreement, SLA) realisiert. Diese SLAs werden mittels eines in Kapitel 2.3 vorgestellten und speziell dafür entwickelten Broker Signaling Protocols (BSP) ausgehandelt.

Da bei kostenloser Ressourcenreservierung mit unfairem Verhalten einiger Beteiligter zu rechnen ist und zudem die höhere Dienstgüte zu entsprechenden Konditionen verkauft werden kann, wird im Bandwidth Broker ein zuverlässiger Mechanismus bereitgestellt, um die Ressourcenreservierung oder Ressourcenverwendung zu verrechnen. Damit wird ein unfaires Verhalten verunmöglicht und der DiffServ-Anbieter vor finanziellen Nachteilen effizient geschützt. Um das Verhalten eines Brokers steuern zu können, ist der Zugriff auf eine Policy Datenbank notwendig, welche festlegt, wer zu welchen Konditionen welche Dienste nutzen darf.

Mit Admission Control und der Verrechnungsmöglichkeit steht oder fällt das DiffServ-Konzept. Wenn alle Benutzer ihre IP-Pakete ohne weitere Konsequenzen mit hoher Priorität markieren können, werden sie dies auch tun, mit der Konsequenz, dass alle Pakete mit höchster Priorität markiert sind und deshalb dieselbe Behandlung erfahren. Was geschieht, wenn alle Pakete gleichberechtigt und somit gleichbehandelt werden, sehen wir heute im World Wide Web, welches deshalb häufig auch World Wide Wait genannt wird.

3.1.1 Service Level Agreement (SLA)

SLAs sind elektronisch ausgehandelte Verträge zwischen mehreren Brokerinstanzen. Der Diensterbringer wird darin Provider, der Dienstnutzer Customer genannt.

In DiffServ-SLAs sind im Wesentlichen folgende Parameter zu finden:

1. Die Vertragspartner
2. Die Dienstbeschreibung bestehend aus einer standardisierten Dienstklasse, der Richtung des Datenflusses, der Bandbreite (Nennwert) und dem Verhalten der Netzwerkelemente.
3. Die Zahlungsmodalitäten

Um bilaterale SLAs verwalten zu können, muss der EBB die Möglichkeit haben, festzustellen mit welchen Domains welche SLAs vereinbart werden können. Dazu ist eine Policy Datenbank notwendig, welche diese Informationen liefert.

3.1.2 Ressourcenverwaltung

3.1.2.1 Bandbreite

Wenn in diesem Kapitel von Ressourcen die Rede ist, beschränkt sich dies auf die Bandbreite, d.h. die zuverlässige Weiterleitung der IP-Pakete, deren aufsummierte Grösse einen bestimmten Grenzwert pro Zeiteinheit nicht überschreitet. Weitere Parameter wie Verzögerung (Delay) oder Verzögerungsschwankungen (Jitter) werden durch andere Broker verwaltet.

Wie IP-Pakete behandelt werden, wenn die vereinbarte Bandbreite überschritten wird, hängt vom jeweiligen Dienst ab. Premium Service verwendet den Begriff Bandbreite klassisch, so

dass bei einer Überschreitung der vereinbarten Bandbreite eine Verstopfung (Congestion) entsteht und überzählige IP-Pakete weggeworfen (gedropt) werden. Statistische Bandbreitengarantien, wie bei Assured Services definiert, bedeuten, dass die aufsummierten Grössen der weitergeleiteten IP-Pakete pro Zeiteinheit durchschnittlich dem durch den Dienst festgelegten Prozentsatz der vereinbarten Bandbreite entsprechen. Wenn beispielsweise ein Assured Service mit 80% von 2 Mbps vereinbart wurde, muss die nutzbare Bandbreite im Durchschnitt mindestens 1.6 Mbps betragen. Dabei kann es durchaus vorkommen, dass die Priorität einiger IP-Pakete herabgesetzt wird oder aber im Extremfall dass IP-Pakete weggeworfen (gedropt) werden. Es kann jedoch durchaus vorkommen, dass während einer kurzen Zeit sogar mehr IP-Pakete weitergeleitet werden, als der Durchschnittswert im SLA vorgibt. Die Sicherstellung der statistischen Zusicherung erfolgt mit speziellen Mechanismen, welche eine möglichst faire Behandlung aller beteiligten Flows erlauben.

3.1.2.2 Inter-Domain und Intra-Domain Links

Unter dem Begriff 'Domain' wird, wo nicht anders vermerkt, eine DiffServ-Domain ('administrative domain') verstanden, welche sich in gewissen Spezialfällen von einer klassischen DNS-Domain unterscheidet (siehe Kapitel 3.4.2).

In jeder DiffServ-Domain ist ein BB für die Ressourcen der eigenen Domain und die Koordination der Ressourcen mit benachbarten Domains verantwortlich.

3.1.2.3 Ressourcenverwaltung in der eigenen Domain (Intra-Domain)

Das DiffServ-Konzept im Rahmen von CATI versucht die Topologie des Netzwerkes sowie die effektiv verwendete Hardware in mehreren Stufen zu abstrahieren, um auf der höchsten Ebene abstrakte Services anzubieten. Dazu werden folgende vier Layer (siehe Kapitel 2.1) [DiffServ Layer] verwendet:

1. Composition Layer
2. Business Layer
3. Domain Dependent Layer
4. Machine Dependent Layer

Der Bandwidth Broker besteht aus einem sogenannten External Service Broker (ESB) des Business Layers und aus einem Internal Service Broker (ISB) des Domain Dependent Layers. Der ESB wird im folgenden External Bandwidth Broker **(EBB)**, der ISB wird entsprechend Internal Bandwidth Broker **(IBB)** genannt (siehe auch Abbildung 3.3, Seite 38).

Die Sicht des EBBs auf das eigene Netz beschränkt sich auf die Border Router und die virtuellen Verbindungen dazwischen (siehe Abbildung 3.1). Sowohl diese Border Router wie auch die erwähnten virtuellen Verbindungen werden nur indirekt via IBB verwaltet:

- Die Border Router müssen konkret vorhanden sein, werden jedoch in abstrakter Weise vom EBB über den IBB angesprochen. Es ist denkbar, in der eigenen Domain z.B. ATM zu verwenden und somit dem IBB zu ermöglichen, innerhalb der Domain auf Layer 2 Traffic Engineering mit QoS-Unterstützung zu betreiben.

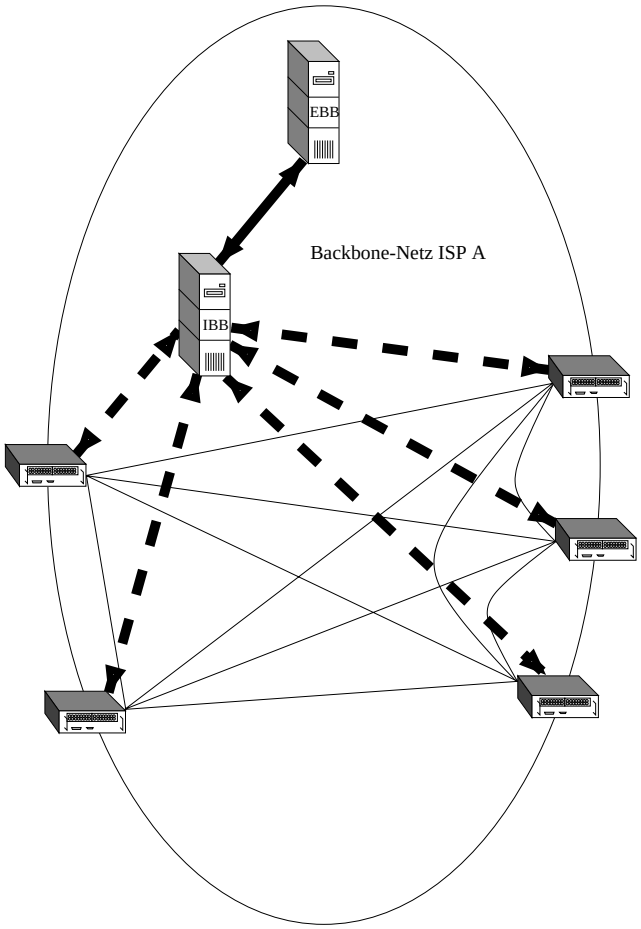


Abbildung 3.1: EBB-Sicht der eigenen Domain

- Der EBB geht davon aus, dass die Border Router logisch vermascht (Full Meshed) untereinander verbunden sind. Diese virtuellen Verbindungen müssen nicht physisch vorhanden sein, da der IBB die Autonomie besitzt, innerhalb der Domain das Routing (Switching bei ATM) selbst zu verwalten. So kann es sein, dass die Border Router physisch in einem Ring oder einem Stern angeordnet und nicht Full Meshed miteinander verbunden sind. Verbreitete Topologien sind in [Tan, Kap. 1.2.3] zu finden. Ausserdem können zwischen den Border Routern beliebig viele weitere Router oder im Falle von ATM Switches vorhanden sein.

Die Aufgaben des IBBs sind die subsidiäre Ressourcenverwaltung innerhalb der eigenen Domain und die Bereitstellung einiger Dienste für den EBB. Aus Sicht des EBBs muss der IBB folgende Aufgaben erfüllen:

- Die Verwaltung der Bandbreite aller potentiell möglichen Links zwischen allen Border Routern der Domain in Zusammenarbeit mit dem EBB. Diese Zusammenarbeit besteht darin, dass der EBB Verbindungsanforderungen mit entsprechenden QoS-Parametern zwischen bestimmten Border Routern an den IBB stellt, welche der IBB zu erfüllen versucht. Der IBB besitzt die Autonomie, z.B. das Routing innerhalb der Domain zu verwalten. Damit können sich die Charakteristiken der virtuellen Links ändern. Diese Änderungen dürfen jedoch nur mit Zustimmung des EBBs erfolgen.

- Das Weiterleiten abstrakter Konfigurationsrequests (Service Class, Rate, Maximum Burst) des EBBs an bestimmte Border Router.
- Das Weiterleiten von statistischen Rohdaten des Configuration Daemons (CD) jedes Border Routers an den EBB.
- Die Berechnung von alternativen Pfaden zu Handen des EBBs (optional).

Der IBB muss mit dem EBB ähnliche Informationen austauschen wie die EBBs verschiedener Domains untereinander. Deshalb erscheint es sinnvoll, das EBB-EBB-Protokoll (BSP) auch als IBB-EBB-Protokoll zu verwenden. Da das BSP-Protokoll ein objektorientiertes erweiterbares Protokoll ist, eignet es sich deshalb ebenso gut für diesen Zweck.

Der IBB und die interne Kommunikation zwischen IBBs und EBBs sind nicht Gegenstand dieser Diplomarbeit.

3.1.2.4 Verwaltung der Ressourcen zu benachbarten Domains (Inter-Domain)

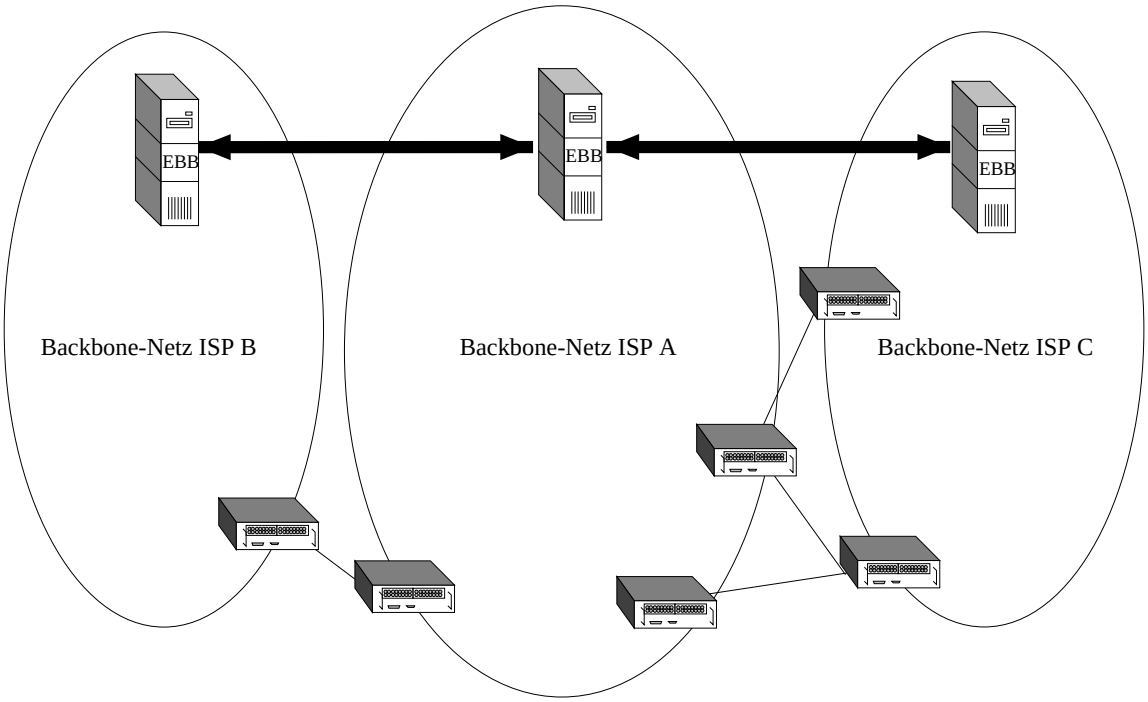


Abbildung 3.2: EBB-Sicht (Inter-Domain)

Die Verwaltung der Ressourcen zu einer benachbarten Domain besteht darin, die Konfiguration der Border Router beidseits jedes Links zwischen der eigenen und dieser benachbarten Domain zu koordinieren. Der sendende Border Router (Egress Router) muss so konfiguriert werden, dass nur die vereinbarte Paket- bzw. Bytemenge mit der entsprechenden Markierung dem Border Router (Ingress Router) der benachbarten Domain gesendet wird. Dieser Ingress Router muss die Einhaltung der vereinbarten und entsprechend markierten Paketmenge überprüfen und gegebenenfalls geeignete Massnahmen ergreifen. Diese Massnahmen sind grundsätzlich durch die Markierung bzw. den Dienst bestimmt, können aber auch kundenfreundlicher durchgesetzt werden.

Ein Beispiel für kundenfreundliche Massnahmen:
Ein Provider bietet einen klassischen DiffServ-Dienst an, regelt aber im SLA, dass bei Überschreitung der vereinbarten Bandbreite IP-Pakete zwar weitergeleitet werden, aber ein entsprechender Zuschlag erhoben wird. Dadurch hat der Kunde den Vorteil, z.B. bei einer kurzfristigen Überschreitung kein neues SLA aushandeln zu müssen und trotzdem von der entsprechenden Dienstgüte profitieren zu können.

Da ein EBB die indirekte Konfiguration der Border Router nur im eigenen Netz vornehmen kann, müssen mit dem EBB der benachbarten Domain bilaterale Vereinbarungen (SLAs) zur Koordination der Konfigurationen getroffen werden.

Die grundsätzlichen Inter-Domain Aufgaben lassen sich wie folgt unterscheiden:

1. Das Aushandeln von SLAs.
2. Die Umsetzung der SLAs in der eigenen Domain.
3. Die Umsetzung der SLAs auf den Inter-Domain Links.

3.1.2.5 Verwaltung von SLAs

Zur Verwaltung von SLAs gehören folgende Teilaufgaben:

- Verwalten von SLAs in einer Datenbank.
- Aushandeln von Parametern mit den EBBs benachbarter Domains für neue wie auch für bestehende SLAs.

SLAs werden gemäss dem in Kapitel 2.2 erläuterten Anfrage-Antwort-Verfahren ausgehandelt.

3.1.2.6 Umsetzung von SLAs durch Steuerung des Verhaltens der Border Router

Wenn ein SLA ausgehandelt ist, müssen die beteiligten EBBs dafür sorgen, dass die entsprechenden Konfigurationen der Border Router im eigenen Netz zum entsprechenden Zeitpunkt durchgeführt werden. Jeder Border Router wird dabei als abstrakter DiffServ-fähiger Netzknoten betrachtet, welcher in der Lage ist, die entsprechenden DiffServ-Dienste einzurichten und zu kontrollieren.

Die erfolgreiche Einrichtung eines Dienstes muss dem EBB via IBB mitgeteilt werden. Der EBB wird dann periodisch die statistischen Rohdaten vom Border Router anfordern. Der Border Router muss dazu in der Lage sein, auf Anfrage des EBBs den Status aller aktuellen Dienste zu rapportieren, um dem EBB die Übersicht über den Netzzustand zu ermöglichen. Solche Statusmeldungen sind die Menge verarbeiteter Bytes für jeden DiffServ Service, die Auslastung der Links, die Auslastung des Prozessors, die Dropstatistiken und eventuell weitere mit den QoS-Mechanismen zusammenhängende Werte. Zwischen dem Router und dem Configuration Daemon kann der Informationsaustausch via Simple Network Management Protocol (SNMP) [RFC1157] erfolgen. Der EBB erwartet vom IBB jedoch BSP-Objects, welche aus SNMP-Informationen zusammengestellt werden.

3.2 Komponenten eines Bandwidth Brokers

Ein Bandwidth Broker besteht im Wesentlichen aus folgenden Komponenten:

- Einen External Bandwidth Broker (EBB)
- Einen Internal Bandwidth Broker (IBB)
- Eine SLA Datenbank
- Eine Policy Datenbank

3.3 Mögliche Szenarien

In den folgenden Szenarien soll die Funktionsweise eines Bandwidth Brokers und der Umgang mit der Struktur und dem Inhalte der BSP-Objects anhand von einigen Beispielen erläutert werden.

3.3.1 Initialisierung eines Bandwidth Brokers

Die Initialisierung des Bandwidth Brokers erfolgt mit folgenden Parametern:

1. Dere Name der DiffServ-Domain, die verwaltet werden soll, und die Koordination mit dem DNS-Server, damit der BB in jeder Domain unter einer einheitlichen Kennung angesprochen werden kann. Die Angabe der vertrauenswürdigen Certification Authorities und die Generierung eigener Verschlüsselungspasswörter.
2. Die Registrierung aller vorhandenen ISBs.
3. Die Definition der zur Domain gehörenden Border Router und die Festlegung der maximalen Bandbreite der Links zwischen diesen Border Router in Zusammenarbeit mit dem IBB. Diese Parameter können zu Laufzeit dynamisch verändert werden.
4. Die Bandbreite jedes Links und die IP-Adresse jedes Ports[2]zu den Border Routern der benachbarten Domain mit Angabe der IP-Adresse und des Ports des benachbarten Border Routers. Diese Parameter sind eher statisch, können aber ebenso zu Laufzeit verändert werden.

3.3.2 Initiierung eines DiffServ-Dienstes

Die Initiierung eines DiffServ-Dienstes erfolgt entweder durch den Netzwerkadministrator oder automatisch anhand von Anforderungen spezieller Anwendungen, welche z.B. mittels RSVP und RDG DiffServ-Dienste beanspruchen.

[2]Der Begriff 'Port' wird in diesem Abschnitt im Sinn von physischer Anschluss verwendet.

3.3.2.1 Initiierung durch den Netzwerkadministrator

Der Netzwerkadministrator kann anhand von Statistiken über die Auslastung einzelner Links und über beanspruchte Dienste in Erwartung eines ähnlichen Verkehrsprofils entsprechende SLAs initiieren. Wenn neue Anwendungen in Betrieb genommen werden, deren Netzwerkanforderungen bekannt sind, kann dieses Wissen ebenso in die Gestaltung der SLAs einfliessen. Diese SLAs können manuell mittels des GUIs des Bandwidth Brokers eingegeben werden.

Auslaufende SLAs können durch den Netzwerkadministrator erkannt und erneuert werden. Diese Arbeit bzw. Intelligenz kann aber typischerweise in den Broker integriert werden, indem ein Regelwerk geschaffen wird, welches auslaufende SLAs unter gewissen Voraussetzungen automatisch erneuert oder noch weitergehende Anpassungen in der Gestaltung von SLAs erlaubt. Dies wird mit der Zeit zur vollständigen Automatisierung des Bandbreitenmanagements führen.

3.3.2.2 Initiierung via RSVP und RDG

Falls im lokalen Bereich bereits Automatismen für das Bandbreitenmanagement mittels IntServ vorhanden sind, kann eine Umsetzung der IntServ-Signalisierung (RSVP) mittels eines RDGs in eine DiffServ-Signalisierung mit entsprechender Aggregation erfolgen. Diese DiffServ-Signalisierung muss im RDG, im IBB oder im EBB in konkrete SLAs umgesetzt werden.

3.3.3 DiffServ Transitverkehr

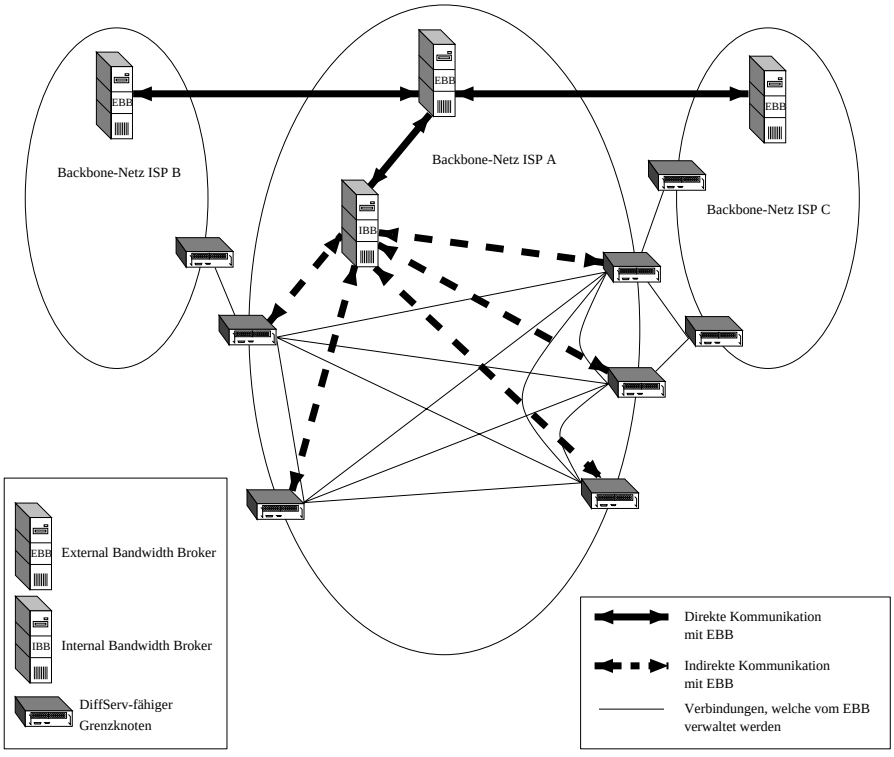


Abbildung 3.3: Transit

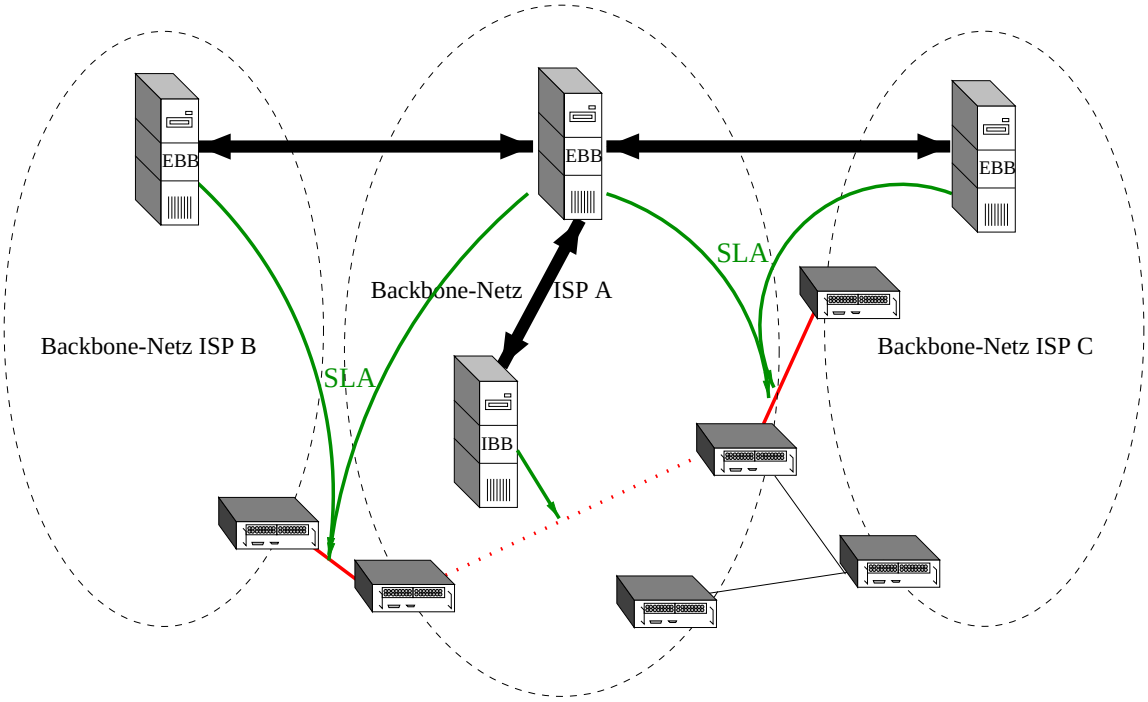


Abbildung 3.4: SLA und zugehörige Links

In diesem Beispiel werden SLAs für den Transitverkehr vom ISP B über den ISP A zum ISP C benötigt. Jedes SLA kommt mittels einer Anfrage und einer Antwort darauf zustande. Insgesamt sind mindestens vier SLAs notwendig (siehe Abbildung 3.4):

1. ein Request zwischen B und A
2. ein Request zwischen A und C
3. ein Commit zwischen B und A
4. ein Commit zwischen A und C

Prinzipiell kann sowohl der BB des ISPs C (BBC) wie auch der BBB oder sogar der BBA den Message-Austausch zur Aushandlung von SLAs initiieren und somit für die Kosten aufkommen[3], selbst wenn es sich um den Verkehr in Richtung B - A - C handelt.

Das einfachste Szenario zwischen genau zwei Brokern wird hier nicht speziell erläutert, da es im Wesentlichen dem hier gezeigten Szenario entspricht, wenn der Endpunkt statt im Netz von ISP B im Netz von ISP A liegen würde. Dann findet ein Austausch von BSP-Messages gemäss der rechten Hälfte (BBA - BBC) des Signalisierungsverkehrs in Abbildung 3.5 statt.

Im folgenden konkreten Fall soll der BBC die SLAs für den Transitverkehr in Richtung B - A - C initiieren. Dazu sind die in Abbildung 3.5 ersichtlichen Schritte notwendig, welche anschliessend einzeln erläutert werden. Detaillierte Angaben über die ausgetauschten Messages sind in den Tabellen auf den Seiten 40 bis 45 zu finden.

[3]Der Zusammenhang zwischen Auslösen von SLA und Kostenübernahme ist wie in 3.3.6 vorgeschlagen nicht zwingend.

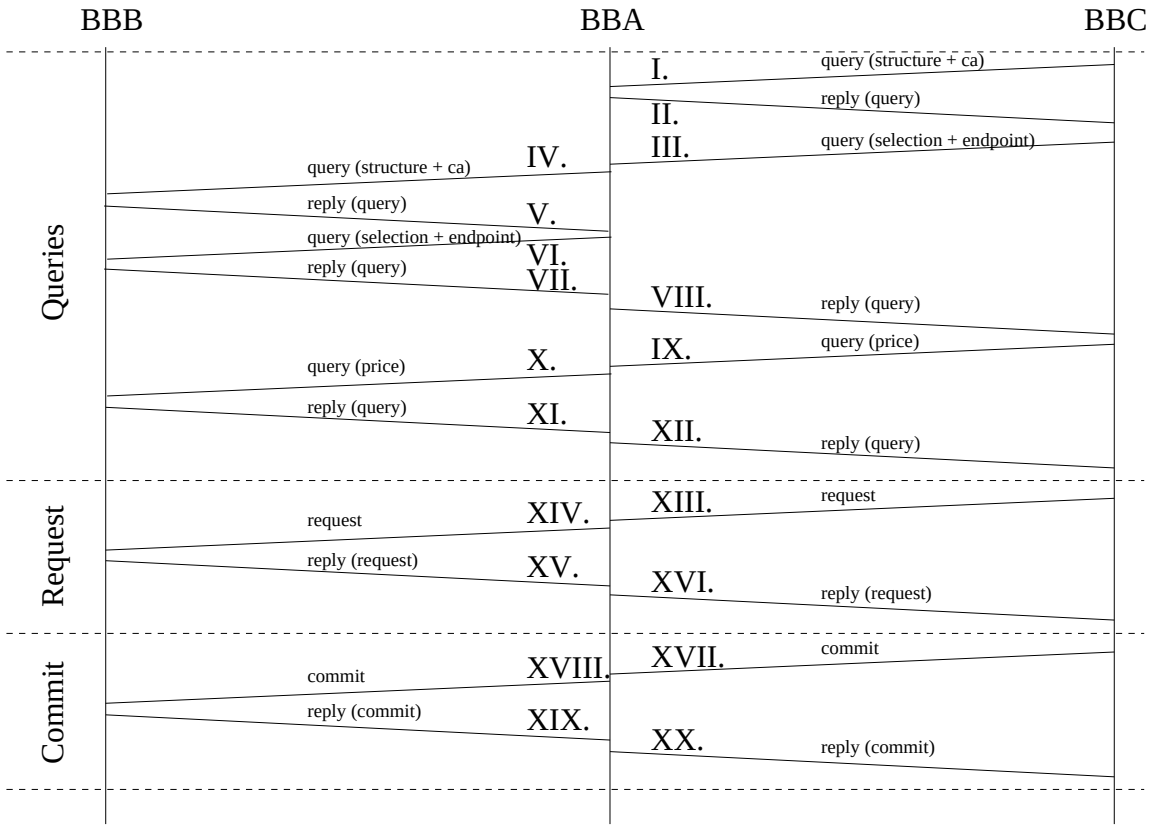


Abbildung 3.5: Signalisierungsverkehr für Transit-SLAs

I. Der BBC stellt eine allgemeine Query an den BBA. In dieser allgemeinen Query müssen nur die Felder Customer und Provider entsprechend ausgefüllt sein und der Status muss, wie bei allen Queries, „please complete“ sein. Hier wird zusätzlich erfragt, welche Certification Authorities der Provider anerkennt (siehe Tab. 3.1).

{MP{LH12d4}{LAquery}{C {OF{LH0}{LAcustomer}{LABBC}} {OF{LH0}{LAprovider}{LABBA}} {OP{LH0}{LAca}{LA}} }}	Von BBC vergebener eindeutiger Identifier. Dienstnutzer Dienstanbieter Certification Authority

Tabelle 3.1: BSP-Message zu Schritt I.

II. Der BBA antwortet mit einer Liste von angebotenen Diensten und der dazu notwendigen Struktur.
Die Objects der Struktur sind mit je einem zulässigen Inhalt versehen (siehe Tab. 3.2).

III. Der BBC teilt nun mit, dass der BBB der Enpoint ist und legt sich fest, welche Certification Authority verwendet werden soll (siehe Tab. 3.3).

IV. Falls der BBA noch kein Request-SLA mit dem BBB hat und auch die vom BBB gewünschten Strukturen nicht kennt, setzt der BBA eine allgemeine Query analog I. an den BBB ab (siehe Tab. 3.4).

V. Der BBB antwortet analog II.

{MP{LH12d4}{LAreply}{C	Gleicher Identifier wie bei Query
{OF{LH0}{LAcustomer}{LABBC}}	unverändert gegenüber Query
{OF{LH0}{LAprovider}{LABBA}}	unverändert gegenüber Query
{OP{LH0}{LAca}{S	wie gewünscht eine Auswahl (Selection)
{OP{LH0}{LAca}{LASwiss Key}}	
{OP{LH0}{LAca}{LAUniKey}}	
}}	
{OF{LH0}{LAservice}{LHef}}	In diesem Fall wird nur ein Dienst. angeboten.
{OP{LH0}{LAdescription}{C	Beschreibung des Dienstes
{OP{LH0}{LAbandwidth}{C	
{OP{LH0}{LAamount}{LA0}}	
{OF{LH0}{LAunit}{LAbps}}	Die Einheit ist hier „fixed“. Der BBA kann nur mit dieser Einheit umgehen.
}}	
}}	
{OF{LH0}{LApayment}{LABilling}}	Bezahlung mittels Rechnung („fixed“).
{OP{LH0}{LAstart}{DHdd30f216f8}}	Vorgeschlagene Startzeit des Dienstes.
{OP{LH0}{LAend}{DHdd361872fb}}	Vorgeschlagene Endzeit des Dienstes.
{OP{LH0}{LAingress}{C	Wenn der Dienst in Richtung A -> C gewünscht wird, kann dieses Netzelement der Domain von BBA verwendet werden.
{OP{LH0}{LAaddress}{LA130.0.2.1}}	
{OP{LH0}{LAmask}{LA255.255.255.255}}	
}}	
{OP{LH0}{LAegress}{C	Wenn der Dienst in Richtung C -> A gewünscht wird, kann dieses Netzelement der Domain von BBA verwendet werden.
{OP{LH0}{LAaddress}{LA130.0.2.1}}	
{OP{LH0}{LAmask}{LA255.255.255.255}}	
}}	
{OP{LH0}{LAendpoint}{LABBA}}	Endpoint wäre BBA, wenn es sich nicht um Transitverkehr handeln würde.
{OF{LH0}{LAsignature}{LAò§*[1]Ö¾xêj_ë}}	Unterschrift von BBA
}}	

Tabelle 3.2: BSP-Message zu Schritt II.

VI. Der BBA sendet eine Query analog III. an den BBB und setzt dabei als Endpoint den BBB ein.

VII. Der BBB antwortet definitiv, da als Endpoint BBB angegeben wurde.

VIII. Der BBA weiss nun, dass der gewünschte Endpoint erreicht werden kann und sendet eine entsprechende Antwort an den BBC. Dabei werden alle Objects der Struktur, welche in der Query vom BBC mit „please complete“ versehen wurden, mit allen möglichen Alternativen ergänzt (Selection).

IX. Nun kann der BBC aus den Alternativen auswählen und damit eine vollständige Query bilden. (Diese Query kann der BBC zwecks Kontrolle an den BBA senden und dann mit der entsprechenden Antwort fortfahren). Mit dieser vollständigen Query kann der BBC durch Hinzufügen einer Price-Struktur (mit Status „please complete“) eine abschliessende Query bilden und damit eine verbindliche Preisanfrage für einen vollständig definierten Dienst an den BBA richten (siehe Tab. 3.5).

X. Der BBA muss nun, um den Preis berechnen zu können, auch eine Price-Query an den BBB senden. In dieser Query werden ebenso die Parameter ausgewählt (z.B. Ingress und Egress Router) wie dies der BBC in IX. tun musste.

XI. Der BBB berechnet den Preis für den Dienst zwischen B und A und sendet eine signierte Reply-Message an den BBA. Hier ist erstmals die Unterschrift von entscheidender Bedeutung, da der BBB sich gegenüber dem BBA verpflichtet, auf Verlangen den gewünschten Dienst zu den entsprechenden Konditionen anzubieten.

XII. Der BBA kann nun seinerseits die Summe der Preise der Verbindungen B - A und A - C berechnen und eventuell auf dem Preis der Verbindung B - A einen Zuschlag berechnen. Der resultierende Preis wird nun in die Price-Struktur der letzten Anfrage vom BBC eingefügt und als signierte Reply zurückgesendet

XIII. Nun ist der BBC in der Lage, in Kenntnis des Preises einen SLA-Request zu initiieren (siehe Tab. 3.6).

XIV. Der BBA muss nun, bevor eine entsprechende Bestätigung an den BBC gesendet werden kann, selbst ein SLA-Request an den BBB senden.

XV. Mit der Bestätigung vom BBB an den BBA ist das 1. SLA (der vier eingangs genannten SLAs) zustandegekommen.

XVI. Nach Erhalt des Request-Reply bestätigt der BBA den SLA-Request dem BBC. Damit kommt das 2. SLA zustande.

XVII. - XX. Analog der Schritte XIII. - XVI. werden die Commit-SLAs erzeugt, wodurch dann jeder BB die Netzelemente in seiner Domain auf den gewünschten Zeitpunkt hin entsprechend konfiguriert.

In diesem einfachen Beispiel wird darauf verzichtet, für die Requests und Commits separate Kosten zu verlangen. Ebenso erfolgt keine gebrauchsabhängige Verrechnung. Der im Price-Object genannte Preis deckt alle Kosten für die Verwendung des Dienstes während der gewünschten Zeit ab.

Wenn nun vor Beginn der Erbringung des Dienstes oder während der Zeitspanne der Erbringung des Dienstes eine Cancel-Message für das betroffene SLA eintrifft, wird der Dienst nicht aufgesetzt bzw. abgebrochen. Dabei können zusätzliche Kosten anfallen, welche z.B. im Price-Query-Reply (Antwort auf eine Price-Query) in einem Object mit Status „informational“ erwähnt worden sind.

Ausblick:
Das „two phase commitment“ ermöglicht Providern, in dieser Weise mit allen Nachbarn SLA-Requests aufzusetzen, ohne bereits entsprechende Kundenrequests zu haben. Damit kann ein Provider künftige Kundenrequests schneller beantworten und somit kurzfristigere und entsprechend teurere SLAs anbieten und sich damit von der Konkurrenz durch einen besseren Service absetzen.

{MP{LH12d6}{LAquery}{C	Neuer Identifier
{OF{LH0}{LAcustomer}{LABBC}}	unverändert
{OF{LH0}{LAprovider}{LABBA}}	unverändert
{OF{LH0}{LAca}{LASwiss Key}}	CA ausgewählt
{OF{LH0}{LAservice}{LHef}}	unverändert
{OP{LH0}{LAdescription}{C	unverändert
{OP{LH0}{LAbandwidth}{C	
{OP{LH0}{LAamount}{LA0}}	
{OF{LH0}{LAunit}{LAbps}}	
}}	
}}	
{OF{LH0}{LApayment}{LABilling}}	unverändert
{OP{LH0}{LAstart}{DHdd30f216f8}}	unverändert
{OP{LH0}{LAend}{DHdd361872fb}}	unverändert
{OP{LH0}{LAingress}{C	unverändert
{OP{LH0}{LAaddress}{LA130.0.2.1}}	
{OP{LH0}{LAmask}{LA255.255.255.255}}	
}}	
{OP{LH0}{LAegress}{C	unverändert
{OP{LH0}{LAaddress}{LA130.0.2.1}}	
{OP{LH0}{LAmask}{LA255.255.255.255}}	
}}	
{OF{LH0}{LAendpoint}{LABBB}}	Als Endpoint wird hier BBB gewählt.
{OF{LH0}{LAsignature}{LAÒk■Ë Jæq}}	Unterschrift von BBC
}}	

Tabelle 3.3: BSP-Message zu Schritt III.

{MP{LHf55}{LAquery}{C	Von BBA vergebener eindeutiger Identifier.
{OF{LH0}{LAcustomer}{LABBA}}	Dienstnutzer
{OF{LH0}{LAprovider}{LABBB}}	Dienstanbieter
{OP{LH0}{LAca}{LA}}	Certification Authority
}}	

Tabelle 3.4: BSP-Message zu Schritt IV.

{MP{LH12d8}{LAquery}{C	Neuer Identifier
{OF{LH0}{LAcustomer}{LABBC}}	unverändert
{OF{LH0}{LAprovider}{LABBA}}	unverändert
{OF{LH0}{LAca}{LASwiss Key}}	unverändert
{OF{LH0}{LAservice}{LHef}}	Status -> „fixed“
{OF{LH0}{LAdescription}{C	Status -> „fixed“
{OF{LH0}{LAbandwidth}{C	Status -> „fixed“
{OF{LH0}{LAamount}{LA1000}}	Wert einsetzen. Status -> „fixed“
{OF{LH0}{LAunit}{LAbps}}	unverändert
}}	
}}	
{OF{LH0}{LApayment}{LABilling}}	unverändert
{OF{LH0}{LAstart}{DHdd30f216f8}}	Status -> „fixed“
{OF{LH0}{LAend}{DHdd361872fb}}	Status -> „fixed“
{OF{LH0}{LAingress}{C	Der Dienst wird in Richtung A -> C gewünscht und somit dieser Vorschlag
{OF{LH0}{LAaddress}{LA130.0.2.1}}	als „fixed“ markiert.
{OF{LH0}{LAmask}{LA255.255.255.255}}	
}}	
{OF{LH0}{LAegress}{C	Es besteht ein Link zwischen 130.0.2.1
{OF{LH0}{LAaddress}{LA130.9.9.9}}	und dem BBC-Netzelement 130.9.9.9.
{OF{LH0}{LAmask}{LA255.255.255.255}}	Der Dienst soll nun auf diesem Link
}}	erbracht werden.
{OF{LH0}{LAendpoint}{LABBA}}	
{OP{LH0}{LAprice}{C	Preisanfrage: nur diese Struktur
{OP{LH0}{LAamount}{LE0.0}}	hat den Status „please complete“.
{OF{LH0}{LAunit}{LAsFr}}	
}}	
{OF{LH0}{LAsignature}{LAGW¬Jïí°&\\K$*}}	Unterschrift von BBC
}}	

Tabelle 3.5: BSP-Message zu Schritt IX.

{MP{LH12da}{LArequest}{C	Neuer Identifier und neuer Typ
{OF{LH0}{LAcustomer}{LABBC}}	unverändert
{OF{LH0}{LAprovider}{LABBA}}	unverändert
{OF{LH0}{LAca}{LASwiss Key}}	unverändert
{OF{LH0}{LAservice}{LHef}}	unverändert
{OF{LH0}{LAdescription}{C	unverändert
{OF{LH0}{LAbandwidth}{C	
{OF{LH0}{LAamount}{LA1000}}	
{OF{LH0}{LAunit}{LAbps}}	
}}	
}}	
{OF{LH0}{LApayment}{LABilling}}	unverändert
{OF{LH0}{LAstart}{DHdd30f216f8}}	unverändert
{OF{LH0}{LAend}{DHdd361872fb}}	unverändert
{OF{LH0}{LAingress}{C	unverändert
{OF{LH0}{LAaddress}{LA130.0.2.1}}	
{OF{LH0}{LAmask}{LA255.255.255.255}}	
}}	
{OF{LH0}{LAegress}{C	unverändert
{OF{LH0}{LAaddress}{LA130.9.9.9}}	
{OF{LH0}{LAmask}{LA255.255.255.255}}	
}}	
{OF{LH0}{LAendpoint}{LABBA}}	unverändert
{OF{LH0}{LAprice}{C	
{OF{LH0}{LAamount}{LE1900.0}}	
{OF{LH0}{LAunit}{LAsFr}}	
}}	
{OF{LH0}{LAsignature}{LAGW¬Jïe#°&ˆO$*}}	Unterschrift von BBC
}}	

Tabelle 3.6: BSP-Message zu Schritt XIII.

3.3.4 IntServ-DiffServ-Gateway eines Stub Netzwerkes

Falls das Stub-Netzwerk[4] DiffServ-fähig ist und über einen BB verfügt, besteht kein Unterschied zum Szenario von Kapitel 3.3.3 (siehe Abbildung 3.6). Fehlt hingegen der BB, so besteht

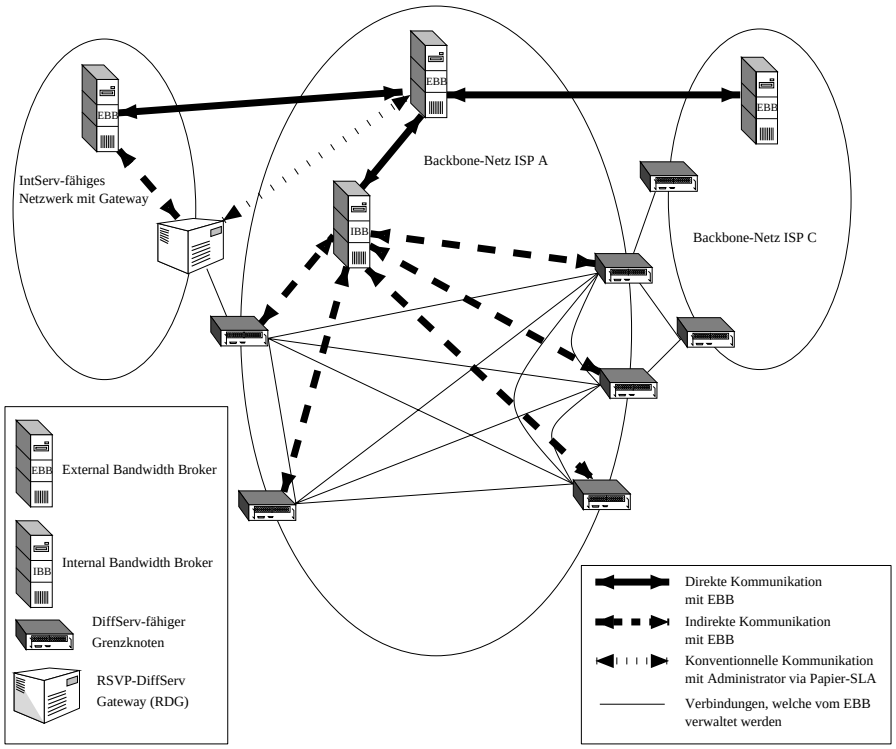


Abbildung 3.6: IntServ-DiffServ-Szenario 1

die Möglichkeit ein traditionelles papierförmiges SLA auszuhandeln und das Ergebnis via EBB in entsprechende Konfigurationen für den betroffenen Border Router der eigenen Domain umzusetzen. Auf der Seite des Stub-Netzes muss eine manuell koordinierte Konfiguration des RDGs erfolgen.

3.3.5 IntServ-DiffServ-Gateway innerhalb des eigenen Netzes

Wenn der RDG innerhalb der eigenen Domain steht, werden die Dienstanforderungen genau gleich behandelt, wie wenn eine Dienstanforderung eines Benutzers innerhalb der eigenen Domain erfolgen würde. Es ist denkbar, falls der RDG eine dynamische Erzeugung von SLAs aufgrund von RSVP-Nachrichten unterstützt, eine DiffServ-Domain zu bilden, welche nur aus dem RDG besteht und diese als benachbarte Domain mit eigenem BB zu betrachten.

3.3.6 Ausblick

Ein weiteres Szenario ist denkbar, bei welchem ein Provider z.B. aufgrund von eigenen Netzverkehrsanalysen einem Kunden ein Angebot machen würde, indem mit einem neuen Message-Typ (z.B. OFFER) der Abschluss von alternativen SLAs empfohlen wird, um die Qualität der vom Kunden verwendeten Netzwerkdienste künftig sicherzustellen.

[4]Netzwerk, welches nur einen Übergang zu weiteren Netzwerken hat (Stub = Stumpf)

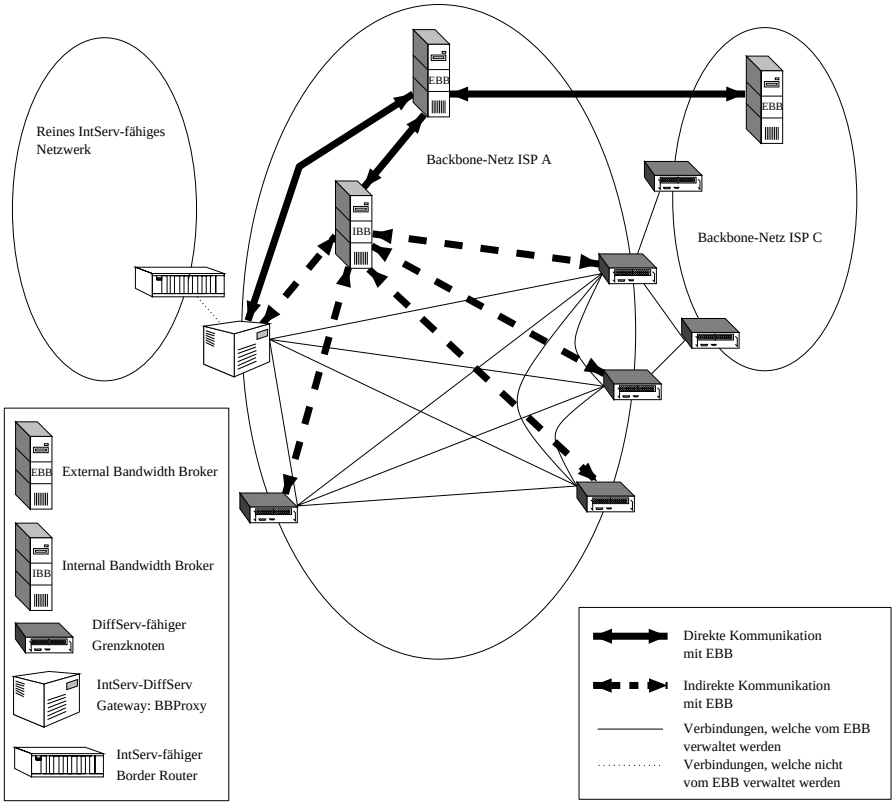


Abbildung 3.7: IntServ-DiffServ-Szenario 2

3.4 Verschlüsselung und Authentisierung von Messages

Die Implementierung des BSP-Protokolls in dieser Arbeit bietet zwei Verschlüsselungs- und Authentisierungsmöglichkeiten an. Einerseits ist eine Verschlüsselung und Signatur mittels eines Verfahrens möglich, welches auf Message Digest 5 (MD5) beruht, andererseits steht die weit verbreitete aber sehr rechenintensive Verschlüsselung und Signatur mittels Pretty Good Privacy (PGP) zur Verfügung. In PGP sind verschiedene Schlüsselstärken möglich, wobei im Rahmen dieser Diplomarbeit nur die Verschlüsselung mit 1024 Bits unterstützt wird.

Da in der Praxis der grösste Teil der Rechenzeit eines Brokers dazu verwendet wird, um zu ver- und entschlüsseln, werden in den folgenden Abschnitten Vorgehensweisen aufgeführt, welche darlegen, wo wieviel Sicherheit notwendig ist, und damit wo mit welchem Verfahren ver- und entschlüsselt werden soll.

3.4.1 Inter-Domain Kommunikation

3.4.1.1 Kommunikation durch fremde Domains hindurch

Beispielsweise bei VPNs kommunizieren die Broker der Domains der IP-Tunnel-Endpunkte direkt miteinander. Dabei transitieren BSP-Meldungen möglicherweise durch fremde Domains hindurch. Deshalb wird empfohlen, den gesamten BSP-Signalisierungsverkehr direkt mittels PGP zu verschlüsseln. Hier ist es denkbar mit der Schlüsselstärke von PGP zu spielen und z.B. Queries nur mit 512 Bits zu verschlüsseln, da der Schaden von gefälschten Queries doch deutlich geringer ist, als dies bei Requests und besonders bei Commits der Fall sein dürfte. Der

Vorteil des schwächeren Schlüssels ist die höhere Geschwindigkeit der Ver- und Entschlüsselung und somit der Verarbeitung von BSP-Messages.

Was die Authentisierung angeht, wird generell empfohlen, alle Messages zu signieren. Das verwendete Verfahren, welches ebenso auf MD5 beruht, setzt voraus, dass beide beteiligten Instanzen dasselbe Passwort kennen. Dieses Passwort kann einmalig oder in regelmässigen Abständen mittels spezieller PGP-Messages ausgetauscht werden. In folgenden drei Fällen wird ausserdem empfohlen, ausschliesslich auf die PGP-Signatur zu vertrauen:

1. Antworten auf Preisanfragen (Price-Query-Reply)
2. Bei allen Requests und Antworten auf Requests (Request-Reply)
3. Bei allen Commits und Antworten auf Commits (Commit-Reply)

3.4.1.2 Direkt benachbarte Domains

Die Kommunikation zwischen direkt benachbarten Domains ist aus Sicht der Verschlüsselung weniger kritisch, da die Möglichkeiten, BSP-Messages abzufangen und damit auszuspionieren, deutlich seltener sind. Somit kann bei einfachen Queries auf eine Verschlüsselung verzichtet werden. Dies ist insbesondere von Bedeutung, wenn man bedenkt, dass doch ein erheblicher Teil der Broker-Kommunikation aus Queries zwischen zwei Brokern benachbarter Domains besteht.

Das Einsparungspotential an Rechenzeit für Ver- und Entschlüsselung ist ebenso beträchtlich, wenn statt PGP MD5 eingesetzt wird. In diesem konkreten Fall wird empfohlen, eine schnelle Ver- und Entschlüsselung anzuwenden, indem einmalig oder in regelmässigen zeitlichen Abständen mittels spezieller PGP-Messages MD5-Passwörter ausgetauscht werden.

Was die Signatur angeht, gilt die bereits erfolgte Empfehlung, bei folgenden Message-Typen ausschliesslich auf eine starke PGP-Signatur zu vertrauen:

1. Bei allen Requests und Antworten auf Requests (Request-Reply)
2. Bei allen Commits und Antworten auf Commits (Commit-Reply)

3.4.2 Intra-Domain Kommunikation

Die Kommunikation zwischen ISBs und ESBs innerhalb einer Domain kann in gewissen Fällen, insbesondere bei relativ kleinen und geografisch auf kleine Gebiete beschränkte Domains unverschlüsselt erfolgen. Da die interne Kommunikation (Intra-Domain) von der Inter-Domain-Kommunikation durch die Wahl der Portnummer getrennt ist, ist hinreichende Sicherheit gewährt, wenn die eigene Firewall generell keine Kommunikation zwischen Rechnern ausserhalb der Domain und Rechnern innerhalb der Domain auf dem internen Port (Slave-Port 8699) zulässt.

Bei grossen Domains, welche sich z.B. über mehrere Kontinente erstrecken, wird dringend empfohlen, in einer DNS-Domain mehrere DiffServ-Domains zu bilden und damit dem internen Charakter des Slave-Ports 8699 Rechnung zu tragen.

3.5 Architektur eines ESBs

Gemäss [VPN Architecture] besteht ein ESB im Wesentlichen aus den 7 Komponenten, welche in der Abbildung 3.8 zu sehen sind. Der EBB ist ein solcher ESB, welcher alle 7 Komponenten implementiert.

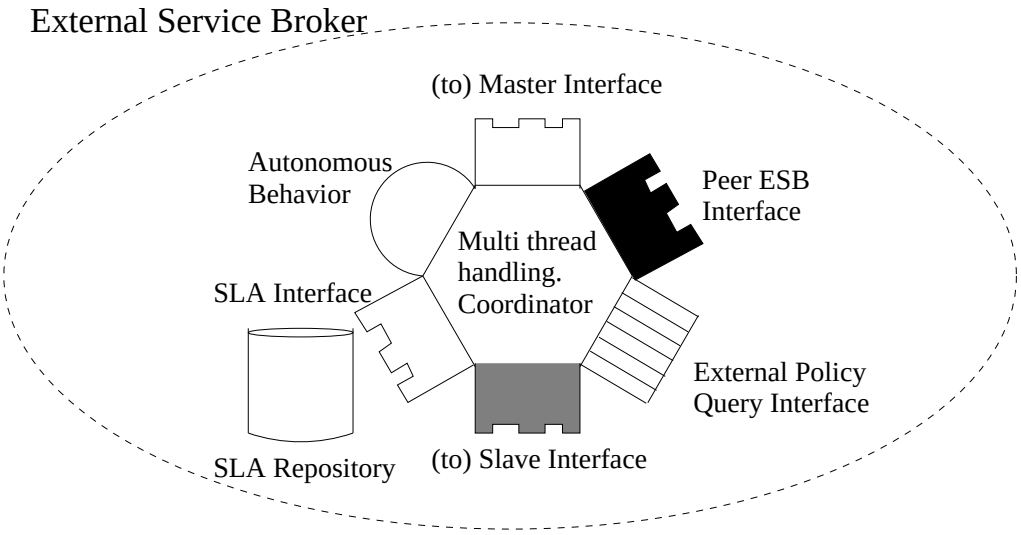


Abbildung 3.8: Komponenten eines ESB

In den folgenden Unterabschnitten werden die Aufgaben der einzelnen Komponenten im Fall eines Bandwidth Brokers erläutert

3.5.1 Multithread Handling Coordinator

Um gleichzeitig mehrere Anfragen bearbeiten zu können und um die eigenen Aufgaben zu koordinieren ist der Multithread Handling Coordinator vorgesehen. Er übernimmt im Wesentlichen Überprüfungs- und Koordinationsaufgaben wie z.B.:

- Führen einer Liste abgesendeter BSP-Messages und Ablehnung von Replies, welche keiner Message dieser Liste entsprechen.
- Verwalten von pendenten Anfragen, z.B. falls der Endpoint einer Anfrage in einer Domain liegt, mit welcher noch kein Anschluss-SLA besteht.
- Ablehnung nicht konformer BSP-Messages, z.B. bei fehlender oder falscher Signatur.

3.5.2 Peer ESB Interface

Das Peer ESB Interface ist zuständig für die sichere Kommunikation mit anderen ESB. Die Kommunikation erfolgt über das in Kapitel 2.3 eingeführte BSP-Protokoll, welches verschiedene Parameter aushandeln kann. Zu diesen Parametern gehören insbesondere auch die Authentisierung der Gegenstelle.

Das ESB-Interface entschlüsselt BSP-Messages und gibt den anderen Komponenten des Brokers bekannt, woher (Port) und mit welcher Verschlüsselungsstärke jede Meldung angekommen ist. Ausserdem teilt das ESB-Interface bei Bedarf mit, wie der mittels PGP authentisierte Absender jeder BSP-Message heisst.

Für jede Übermittlung eines BSP-Messages wird ein entsprechender Server gestartet, welcher versucht, den Empfänger zu erreichen. Bricht die Verbindung ab oder lehnt der Empfänger die Verbindung temporär ab, wird insgesamt fünf Mal ein Verbindungsaufbau versucht. Zwischen diesen Versuchen wird ein inkrementelles Delay eingefügt. Damit wird eine fehlertolerante Kommunikation erreicht.

3.5.3 Master Interface

Der BB ist über das Master Interface durch den Netzadministrator steuerbar. Das Master Interface besteht im Wesentlichen aus einem Konfigurationsmodul und einem grafischen Benutzerinterface (GUI) und optional aus einem Command Line Interface.

Die Bearbeitung der Messages, welche nicht vollautomatisch verarbeitet werden können, erfolgt mittels eines separaten GUIs. Dieses GUI baut sich dynamisch auf und ist damit in der Lage, heute noch unbekannte Message-Typen mit neuen Strukturen korrekt darzustellen.

3.5.4 Slave Interface

Das Slave Interface wird zur Kommunikation mit den Komponenten des eigenen Netzes und zur Steuerung des ISBs (IBB) verwendet. Die Sicherheitsanforderungen an dieses Interface sind geringer als beim Peer ESB Interface.

3.5.5 SLA Interface

Das SLA Interface implementiert die Schnittstelle zur SLA-Datenbank, aus welcher alle SLA-Zustände entnommen werden können. In diesem Interface wird die Koordination der Zustände der SLAs und das effektive Bandbreitenmanagement vorgenommen.

3.5.6 External Policy Query Interface

Diese Schnittstelle ermöglicht das Verhalten gegenüber anderen ESBs und das Pricing zu bestimmen. Im Fall des BBs liefert eine Policy-Datenbank die Verhaltensregeln gegenüber anderen BBs und alle Preisinformationen für DiffServ-Dienste. Dieses Interface wird indirekt über ein eigenes Managementprogramm verwaltet, welches die Policies in die Datenbank einträgt. Dieses Managementprogramm gehört nicht zum BB.

3.5.7 Autonomous Behavior

Die Autonomie des BBs kann sehr rudimentär ausfallen oder aber eine gewisse Eigenständigkeit ermöglichen, um z.B. SLAs, welche kurz vor Ablauf sind, rechtzeitig aufzufrischen oder

bestehende SLAs abhängig vom Netzzustand neu zu verhandeln. Des weiteren sind andere Automatismen denkbar, welche Optimierungsvorschläge oder sogar Änderungen in der Preisstruktur empfehlen könnten.

Kapitel 4

Implementierung

Die Implementierung des External Service Brokers ist in Java realisiert, um die Portabilität zu gewährleisten. Die Java GUI-Bibliothek [Swing] ermöglicht zudem eine weitgehend betriebssystemunabhängige Programmierung des GUIs, wobei auf allen Betriebssystemen die Darstellung mit grafischen Elementen des Betriebssystems selbst erfolgt und somit mit Elementen, welche dem jeweiligen Benutzer bekannt sind. Die Vorteile der dynamischen Darstellung und direkten Bearbeitungsmöglichkeit von BSP-Messages im GUI werden damit auf allen Systemen vorhanden sein.

Ich verzichte in dieser Dokumentation, alle ca. 70 Klassen detailliert zu dokumentieren und verweise an dieser Stelle auf den Source-Code. Die wichtigsten Klassen und die wesentlichen Zusammenhänge werden in den folgenden Abschnitten erläutert. Die Implementierung umfasst ca. 10'000 Code-Zeilen und ca. 600 Funktionen. An dieser Stelle möchte ich mich bei Manuel Günter ganz herzlich für seine Unterstützung bedanken und festhalten, dass ca. 3'000 Code-Zeilen ursprünglich von Ihm stammen. Dies trifft insbesondere auf die Verschlüsselungs- und Authentisierungsfunktionalität wie auf die Implementierung des Parsers zu.

Die Klassen sind auf folgende Packages aufgeteilt:

- BSP
  Alle Klassen zur Bearbeitung und Darstellung von Protokoll-Elementen des BSP-Protokolls sind in diesem Package zusammengefasst. Die Definitionen der Object-Zustände und die Codierung der BSP-Elemente sind in ByteConstants.java zu finden.

- CRYPTO
  Die Funktionalität zur Verschlüsselung und Authentisierung ist in den Klassen des CRYPTO-Packages konzentriert.

- PARSING
  In diesem Package sind die Klassen zur Analyse und Bearbeitung der BSP-Elemente von Messages zu finden. In der Klasse ProtocolConstants.java dieses Packages sind alle Messages- und Objects-Definitionen aufgelistet.

- EBB
  Das External Bandwidth Broker Package umfasst alle Klassen und Komponenten des BB-Prototyps.

4.1 Komponentenhierarchie

In Abbildung 4.1 ist die Komponentenhierarchie in grafischer Weise dargestellt. Diese Darstellung soll einen kleinen Einblick in die wesentlichen Zusammenhänge der 16 wichtigsten Klassen der Implementierung geben. Dabei ist insbesondere zu beachten, dass Funktionen, welche von

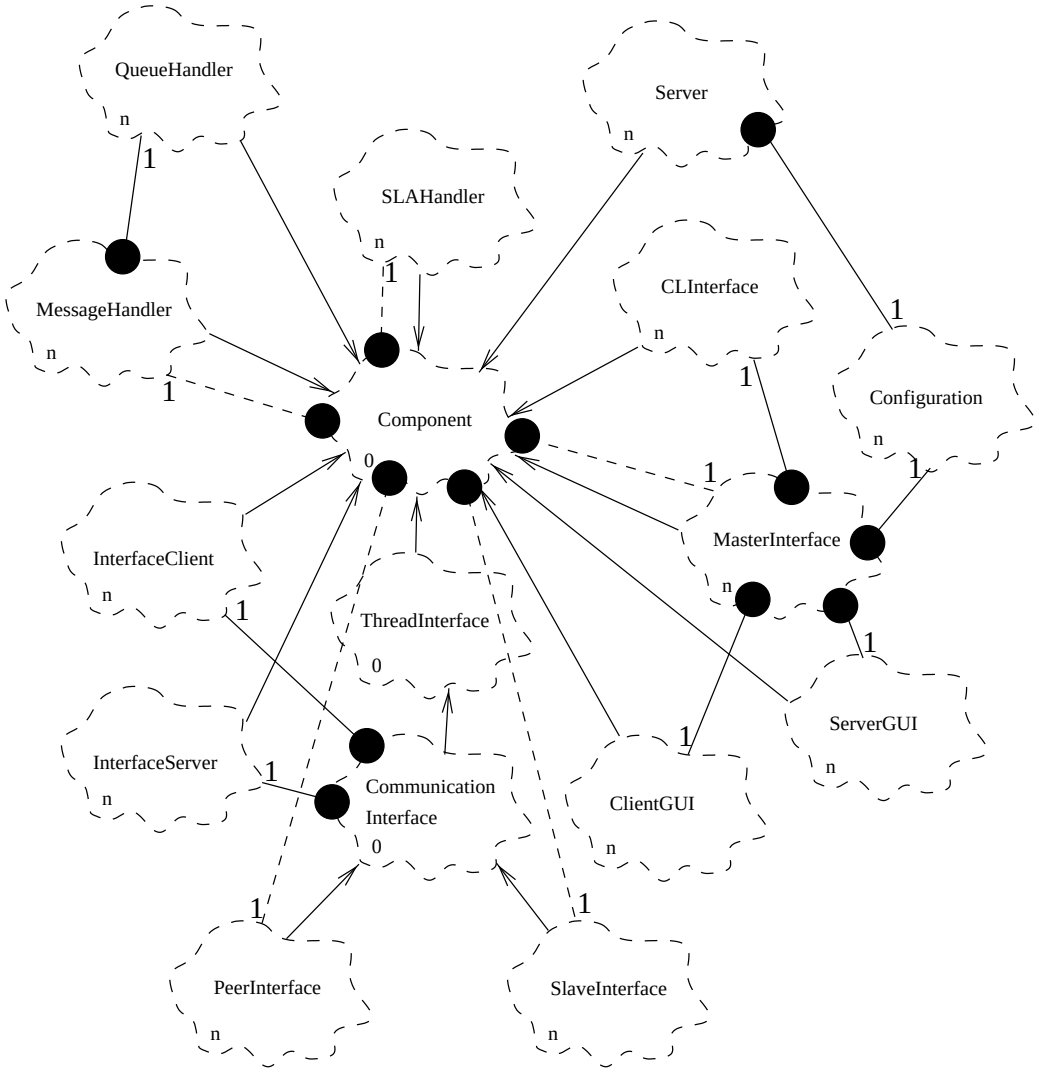


Abbildung 4.1: Komponentenhierarchie (Package EBB)

mehreren Komponenten verwendet werden, in der übergeordneten Klasse implementiert sind bzw. eine übergeordnete Klasse gebildet wurde. Damit ist der Code strukturiert und nicht mehrfach vorhanden, was die Codepflege und nicht zuletzt das Debugging vereinfacht.

4.1.1 Grundfunktionalität: ebb.Component

Alle Komponenten werden von der abstrakten Klasse ebb.Component abgeleitet, welche die Grundfunktionalität jeder Komponente bereitstellen soll. Zu dieser Grundfunktionalität gehören mehrere Writer, welche unterschiedliche Log-Meldungen entsprechend formatiert sowohl in ein Online-Log (siehe Abbildung 4.12) wie auch in ein Logfile schreiben. Alle Meldungen in das

Logfile werden zusätzlich mit dem Zeitstempel des Systems versehen. Zur Grundfunktionalität jeder Komponente gehört auch die Fähigkeit, BSP-Objects zu erzeugen. Diese Funktionalität wird durch den Object Creator bereitgestellt.

4.1.1.1 Writer

Folgende Writer erlauben es, Meldungen je nach Wichtigkeit und Dringlichkeit auf unterschiedliche Weise zu logen:

1. Alert-Meldungen werden erzeugt, wenn eine sicherheitsrelevante Regel verletzt wird oder eine anderweitige Dringlichkeit vorliegt. Die Verletzung einer solchen Regel ist beispielsweise das Vortäuschen eines falschen Absenders. Der Broker erlaubt es, die Anzahl aktiver ESB- und ISB-Server-Threads zu beschränken. Wird diese Beschränkung erreicht bzw. müssen wegen dieser Beschränkung Anfragen abgelehnt werden, wird eine Alert-Meldung erzeugt. Eine Erweiterung dieses Writers wäre z.B. die Meldung von Alerts via e-mail oder Pager an den Systemadministrator. Beim Logen werden Alert-Meldungen mit 'A -> ' eingeleitet.

2. Error-Meldungen werden erzeugt, wenn Messages fehlerhafte Objects enthalten oder andere Fehler vorliegen, welche eine Weiterverarbeitung dieser Messages verunmöglichen. Wenn der Broker eine Anfrage mit einer Error-Message beantworten muss, wird ebenso eine Error-Meldung erzeugt. Bei diesem Writer sind ähnliche Erweiterungen denkbar wie beim Alert-Writer. Beim Logen werden Error-Meldungen mit 'E -> ' eingeleitet.

3. Warnhinweise, welche eine geringere Wichtigkeit aufweisen als Error-Meldungen, sich aber von routinemässigen Meldungen unterscheiden sollen, können mit dem Warn-Writer gelogt werden. Beim Logen werden Warn-Meldungen mit 'W -> ' eingeleitet.

4. Alle anderen Meldungen werden mit dem Log-Writer bearbeitet. Diese Meldungen erlauben eine Überwachung des Brokers und sind zu Nachweiszwecken verwendbar. Beispielsweise kann anhand solcher Meldungen der sonst unwichtige Zusammenhang zwischen dem Identifier einer verschlüsselten Message und dem Identifier der zugehörigen unverschlüsselten Message hergestellt werden.

4.1.1.2 Object Creator

Eine wichtige Komponente, welche die Handhabung von BSP-Messages bei der Programmierung vereinfachen soll, ist der Object Creator. Der Object Creator bietet Funktionen an, welche korrekte BSP-Messages anhand einer vorgegebenen Liste von Parametern erzeugen. So ist es beispielsweise möglich, Fehlermeldungen mit einem Funktionsaufruf zu erzeugen und dabei die fehlerverursachende Message auf einfache Weise in die Error-Message zu integrieren.

4.1.1.3 Drehscheibe der Kommunikation

Über diese Component-Klasse können zudem die fünf folgenden Komponenten des Brokers untereinander angesprochen werden: Master Interface, Peer Interface, Slave Interface, Message Handler und SLA Handler. Eine abstrakte Methode überprüft dabei, ob die entsprechenden Komponenten bereits initialisiert sind. Diese Klasse stellt eine Art Drehscheibe für die Kommunikation zwischen den abgeleiteten Komponenten dar.

4.2 Message Handling und Kommunikation

Die Kommunikation mit anderen Brokern geschieht über das Communication Interface, welches im Wesentlichen verschlüsselte BSP-Messages empfangen und absenden kann.

Ein erster Ansatz hat vorgesehen, eine Verbindung für jedes Anfrage-Antwort-Paar aufrechtzuerhalten, bis die Antwort empfangen worden ist. Im Laufe der Implementierung und entsprechender Tests hat sich herausgestellt, dass dieser verbindungsorientierte Ansatz problematisch ist, weil dabei Verbindungen aufrechterhalten werden müssen, obschon über sehr lange Zeit weder Daten gesendet noch empfangen worden sind. Dies wirkt sich besonders dann negativ aus, wenn ein Broker vor der Beantwortung einer Anfrage mit einem oder sogar mehreren benachbarten BBs Abklärungen vornehmen muss, wie dies beim Forwarding (siehe Kapitel 4.2.2) notwendig ist. Dieser verbindungsorientierte Ansatz ist in der Folge nicht mehr berücksichtigt worden, obschon im folgenden Beispiel nur ein solcher Ansatz möglich ist.

Beispiel:
Wenn ein Kunde eines ISPs über keinen BB verfügt, wäre es denkbar ein HTML-Formular mit Java-Applets zur Verfügung zu stellen, welches die Kommunikation mit dem BB des ISPs erlaubt. Der Benutzer eines solchen Formulars ist aber nur solange für Antworten des BBs erreichbar, wie die Verbindung aufrechterhalten wird oder aber die Kommunikation über ein CGI-Skript abgewickelt wird. Dieses CGI-Skript würde diese Verbindungsparameter verwalten. Dieses Szenario wäre ebenso für die Kommunikation von Benutzern im eigenen Netz mit dem internen BB denkbar.

Ein solcher Ansatz, für jede Übermittlung einer Message eine Verbindung aufzubauen und danach unmittelbar wieder abzubauen, hat den Vorteil, Übermittlungsfehler besser handhaben zu können, indem z.B. bei unerwartetem Verbindungsabbruch einfach eine neue Verbindung aufgebaut und die Message nochmals gesendet wird. Die Koordination zwischen Anfrage und Antwort bedeutet aber einen nicht unwesentlich grösseren Aufwand beim Message Handling.

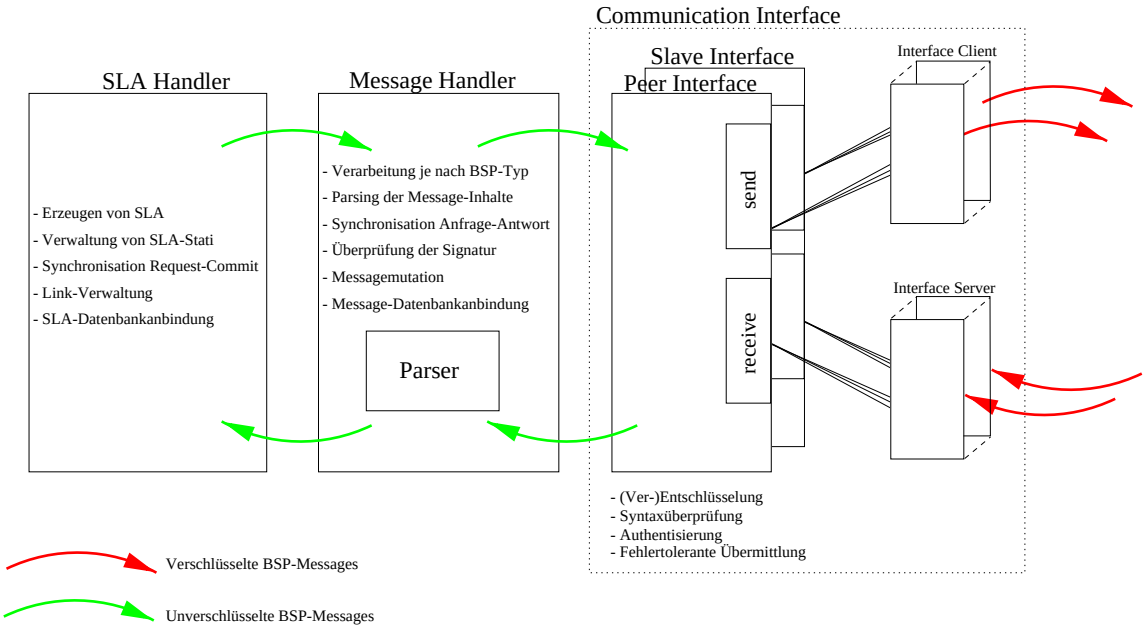


Abbildung 4.2: Message Handling

In Abbildung 4.2 sind die wesentlichen Abläufe bei der Verarbeitung von Messages darge-

stellt. Eine ankommende Message wird als erstes im Communication Interface entschlüsselt und in eine interne Darstellung gebracht. Danach wird die Message dem Message Handler übergeben, welcher die eigentliche Bearbeitung übernimmt und falls nötig eine Antwort generiert. Requests, Commits und Antworten darauf werden dem SLA Handler übergeben, welcher Buch führt über das Zustandekommen eines SLAs. Ein SLA besteht im Wesentlichen aus einer authentisierten Anfrage und der zugehörigen authentisierten Antwort. Die Koordination zwischen Request-SLAs und Commit-SLAs wird ebenso durch den SLA Handler sichergestellt. In den folgenden Abschnitten wird jede Komponente einzeln erläutert.

4.2.1 ebb.CommunicationInterface

In dieser Komponente ist die Funktionalität realisiert, welche für die Kommunikation mit anderen Brokern notwendig ist. Jede Verbindung läuft in einem eigenen Thread. Ankommende Messages werden durch InterfaceServer-Threads, abgehende Messages durch InterfaceClient-Threads parallel bearbeitet.

4.2.1.1 Ankommende Messages empfangen

Messages werden durch Java ServerSockets auf dem entsprechenden Port entgegengenommen, indem bei jeder Verbindungsanfrage ein eigener Socket erzeugt wird, welcher dem ebenso neu erzeugten InterfaceServer-Thread übergeben wird. Als interner Port (Slave Port) wurde der Port mit der Nummer 8699 und als externer Port (Peer Port) der Port mit der Nummer 8698 definiert.

Die Verarbeitung nach dem Message-Empfang erfolgt in folgenden Schritten:

1. Der Protokoll-Parser überprüft die Syntax der Message mittels Methoden bzw. Konstruktoren der Klassen im BSP-Package. Der Parser wandelt die Codierung bestimmter Objects bzw. Lists bereits in alternative Codierungen der internen Darstellungen um.
2. Zur Decodierung einer verschlüsselten Message sind entsprechende Crypto-Objekte für PGP und für MD5 vorhanden.
3. Die Authentisierung wird vorgenommen und der ermittelte Absender zur weiteren Verarbeitung weitergeleitet.

Nach der Bearbeitung durch den InterfaceServer-Thread wird die Message dem Message Handler zur weiteren Bearbeitung übergeben. Dabei werden folgende Parameter übergeben:

- Die Codierung bzw. die Stärke der Verschlüsselung.
- Die Feststellung, ob die Message überhaupt verschlüsselt worden ist.
- Der Name bzw. die IP-Adresse des Senders.
- Die Angabe, ob die Message auf dem internen oder externen Port empfangen worden ist.
- Die entschlüsselte Message.

Anhand dieser Parameter kann der Message Handler selbst entscheiden, wie und ob er Messages weiterverarbeiten will. Insbesondere ist eine Abbildung dieser Parameter auf eine interne Skala vorgesehen, womit der Message Handler unabhängig vom effektiv eingesetzten Verschlüsselungs- oder Authentisierungsverfahren Entscheide fällen kann.

4.2.1.2 Abgehende Messages senden

Die Übermittlung von Messages an andere Broker wird auch durch das Communication Interface gewährleistet. Dabei wird zur fehlertoleranten und parallelen Verarbeitung für jede abgehende Verbindung ein eigener InterfaceClient-Thread erzeugt. Der InterfaceClient erfüllt folgende Aufgaben:

1. Verschlüsselung der Message.
2. Umsetzung der internen Codierung in eine BSP-Syntax.
3. Der InterfaceClient implementiert die fehlertolerante Übermittlung, indem insgesamt fünf Versuche unternommen werden, die Message dem Empfänger zuzusenden. Zwischen diesen Versuchen wird ein inkrementelles Timeout eingefügt. Bei Misserfolg erfolgt eine Mitteilung an den Broker.
4. Nach erfolgter Zusendung wird die verschlüsselte Message zwecks Nachweis dem Message Handler zur Archivierung übergeben.

Welche Art von Verschlüsselung gewählt wird, wird vom Message Handler bestimmt, wobei je nach Destination (intern oder extern) und je nach Message-Typ andere Verfahren oder Schlüsselstärken gewählt werden. So werden im vorliegenden Prototyp Queries prinzipiell nur mit MD5 verschlüsselt, Requests und Commits aber mit PGP und einer Schlüsselstärke von 1024 Bits. Die Wahl des Verfahrens kann im Composer GUI für jede Message gewählt werden.

4.2.2 ebb.MessageHandler

Der Message Handler bearbeitet Messages weiter, welche vom Communication Interface, vom Composer GUI oder einer anderen Komponente her kommen.

4.2.2.1 Forwarding

Im Fall von Transitverkehr (siehe Kapitel 3.3.3) ist ein sogenanntes Forwarding notwendig, d.h. die Sicherstellung dass nicht nur die Verbindung zwischen dem Customer und dem Provider, sondern auch die Verbindung zwischen dem Provider und dem gewünschten Endpoint durch entsprechende SLAs abgedeckt wird. Das Forwarding wird statisch implementiert (siehe auch Kapitel 4.2.4.3). Dies bedeutet, dass für jede Anfrage, dessen Endpoint nicht im eigenen Netz liegt, der Provider selbst dafür sorgen muss, dass SLAs bis zum gewünschten Endpoint entstehen. Erst wenn diese SLAs vereinbart sind, kann der Provider dem Customer die Anfrage bestätigen.

4.2.2.2 Bearbeitung selbst erzeugter Messages

Selbst erzeugte Messages (Queries, Requests und Commits) sind Messages, welche im GUI erzeugt oder weiterverarbeitet werden. Automatisch erzeugte Antworten (Replies), welche in Kapitel 4.2.2.3 erwähnt werden, erfahren dieselbe Behandlung und werden deshalb auch hier erwähnt.

Diese Messages werden im Message Handler als Erstes signiert und je nach Message Typ zunächst gemäss folgender Auflistung behandelt:

- **Queries**:
  Der Identifier von jeder Query-Message wird in einer Query-Liste festgehalten, um eine Antwort darauf eindeutig zuordnen zu können.
- **Requests**:
  Der Identifier jeder Request-Message wird in einer Request-Liste festgehalten, um eine Antwort darauf eindeutig zuordnen zu können. Danach wird diese Message inklusive der nun vorhandenen Signatur als Request mit dem Zustand 'request sent' dem SLA Handler mitgeteilt.
- **Commits**:
  Der Identifier jeder Commit-Message wird in einer Commit-Liste festgehalten, um eine Antwort darauf eindeutig zuordnen zu können. Danach wird diese Message inklusive der nun vorhandenen Signatur als Commit mit dem Zustand 'commit sent' dem SLA Handler mitgeteilt.
- **Replies**:
  Bei Reply-Messages wird nur die Signatur dem SLA Handler mitgeteilt, welcher selbst erkennt, ob er die Signatur überhaupt benötigt und zu welcher Anfrage die Signatur gehört.

Erfolgen diese Schritte ohne Fehler, werden die Messages via Communication Interface unter Angabe der gewünschten Codierung den Empfängern zugestellt.

4.2.2.3 Bearbeitung ankommender Messages

Messages vom Communication Interface werden nach Typ sortiert und danach gemäss folgender Liste bearbeitet. Die Flussdiagramme der Abbildung 4.3 - 4.5 dokumentieren den Ablauf für jeden Message-Typen.

- **Queries** (siehe Abbildung 4.3):
  1. Überprüfung, ob der Status 'please complete' ist.
  2. Message-Parsing (siehe Kapitel 4.2.3).
  3. Je nach Ergebnis des Parsings erfährt die Message eine der folgenden Behandlungen:
     (a) Falls der Endpoint der Query nicht im eigenen Netz liegt, erfolgt ein Eintrag in die Forwardingliste sowie eine Query-Anfrage (forwarded query) an einen weiteren BB.

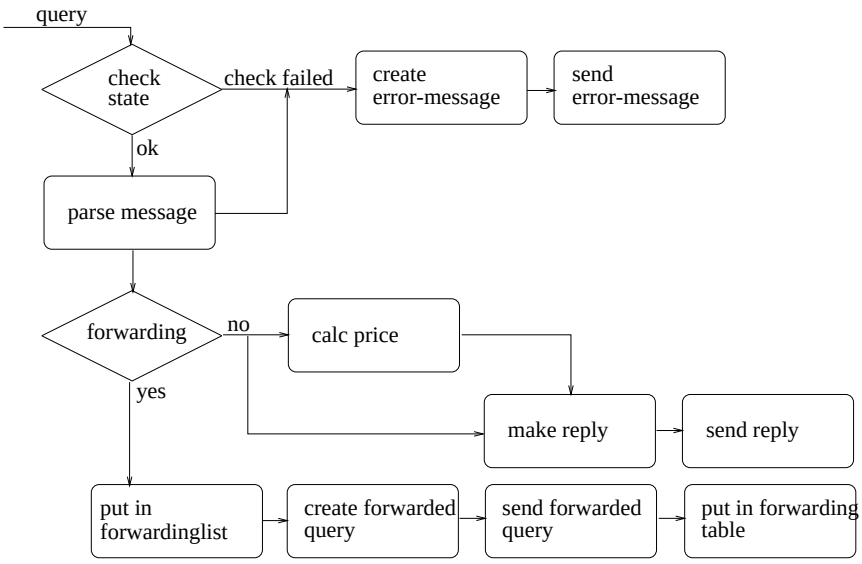


Abbildung 4.3: Query-Flussdiagramm

(b) Falls das Endziel der Query im eigenen Netz liegt, erfolgt eine Preisberechnung oder die Erstellung einer Antwort (Reply-Message) und deren Rücksendung an den Kunden.

(c) Entsteht bei der Überprüfung (1.) oder dem Parsing (2.) ein Fehler, wird eine Fehlermeldung an den Absender gesendet.

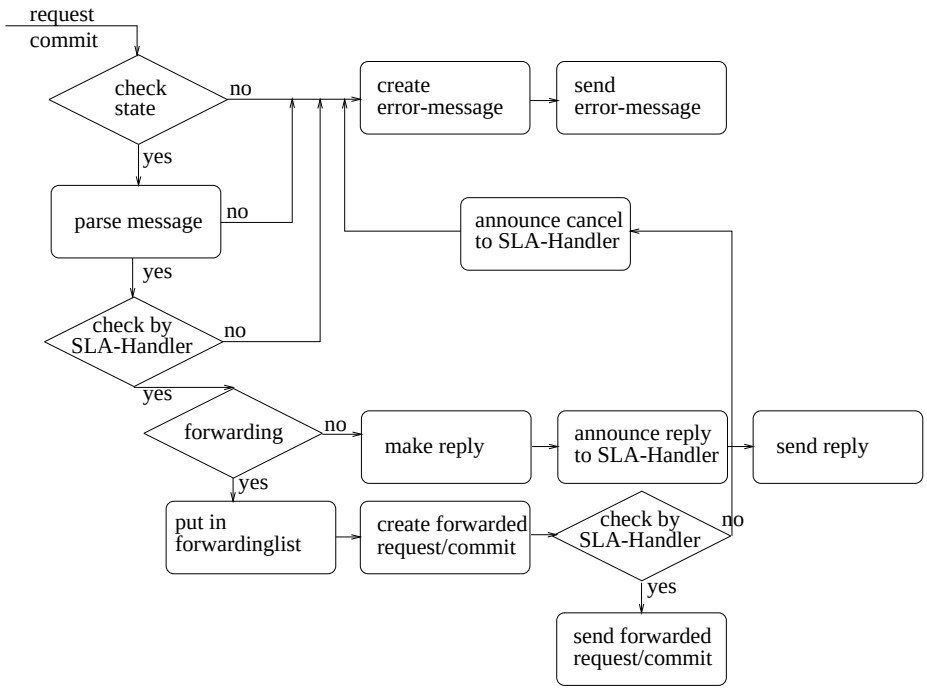


Abbildung 4.4: Request/Commit-Flussdiagramm

- **Requests und Commits** (siehe Abbildung 4.4):

  1. Überprüfung, ob der Status 'fixed' ist.

2. Message-Parsing (siehe Kapitel 4.2.3).
3. Weiterleitung der Message an den SLA Handler, welcher überprüft, ob das gewünschte SLA eingegangen werden kann.
4. Je nach Ergebnis des Parsings und der Überprüfung durch den SLA Handler erfährt die Message eine der folgenden Behandlungen:
   (a) Falls der Endpoint der Anfrage (request/commit) nicht im eigenen Netz liegt und falls die Überprüfung durch den SLA Handler positiv verläuft, erfolgt ein Eintrag in die Forwardingliste sowie eine Request- bzw. Commit-Anfrage (forwarded request/commit) an einen weiteren BB.
   (b) Falls der Endpoint der Anfrage im eigenen Netz liegt und falls die Überprüfung durch den SLA Handler positiv verläuft, erfolgt die Erstellung einer Reply, die Benachrichtigung an den SLA Handler, dass das SLA eingegangen wird, und die Rücksendung der Antwort an den Kunden.
   (c) Entsteht bei der Überprüfung (1.), dem Parsing (2.) oder der Überprüfung durch den SLA Handler ein Fehler, wird eine Fehlermeldung an den Absender gesendet.

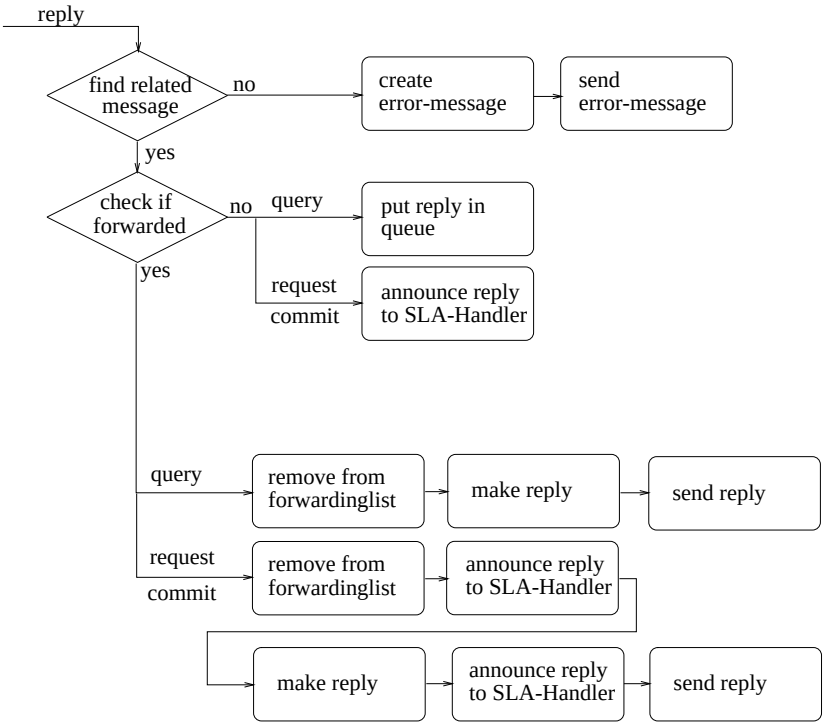


Abbildung 4.5: Reply-Flussdiagramm

- **Replies** (siehe Abbildung 4.5):

1. Überprüfung, ob es sich um eine Forwarding-Anfrage handelt, d.h. einer Message, welche explizit durch eine Anfrage ausgelöst wurde und dank dieser Reply beantwortet werden kann.
2. Falls es sich nicht um eine Forwarding-Anfrage handelt, wird
   (a) bei einer Query-Reply die Reply in die Queue geleitet,

   (b) bei einer Request- oder Commit-Reply der SLA Handler benachrichtigt.

3. Falls es sich um eine Forwarding-Reply handelt, wird

   (a) bei einer Query-Reply die zugehörige Query aus der Queue entfernt, eine entsprechende Antwort (Reply) generiert und diese Reply an den Kunden zurückgesendet,

   (b) bei einer Request- oder Commit-Reply die entsprechende Anfrage (Request oder Commit) aus der Queue entfernt. Danach wird der SLA Handler benachrichtigt, dass eine Request- oder Commit-Reply eingetroffen sei und eine entsprechende Antwort (Reply) generiert. Schliesslich wird das Versenden dieser Antwort dem SLA-Handler mitgeteilt und diese Reply dem entsprechenden Kunden zurückgesendet.

- **Error, Cancel** und **Signal**:
  Falls eine Error-, Cancel- oder Signal-Message eintrifft, wird diese ohne weitere Überprüfung in die Queue geleitet.

4.2.2.4 Error-Handling

Dem Error-Handling wird besondere Beachtung geschenkt, indem möglichst alle auftretenden Fehler korrekt behandelt werden und wenn möglich sowohl dem fehlerverursachenden wie auch dem fehlerfeststellenden BB kommentierte Fehlermeldungen gemeldet werden. Die Form einer solchen Fehler-Meldung wird in Abbildung 4.6 erläutert. Dabei wird im Object 'error_message' eine textuelle Beschreibung der Fehler angegeben und wenn möglich die Einträge in den betroffenen Objects rot hervorgehoben und der Object-Name entsprechend dem Status ('erroneous') rot geschrieben. Eine Error-Message kann im GUI direkt weiterverarbeitet bzw. in eine Query-, Request- oder Commit-Message umgewandelt werden. Die Darstellung des Inhaltes des Objects 'error_message' wird im vorliegenden Prototyp leider leserunfreundlich in der standardisierten Breite dargestellt, obschon der Inhalt aus einem oder sogar mehreren Sätzen bestehen könnte.

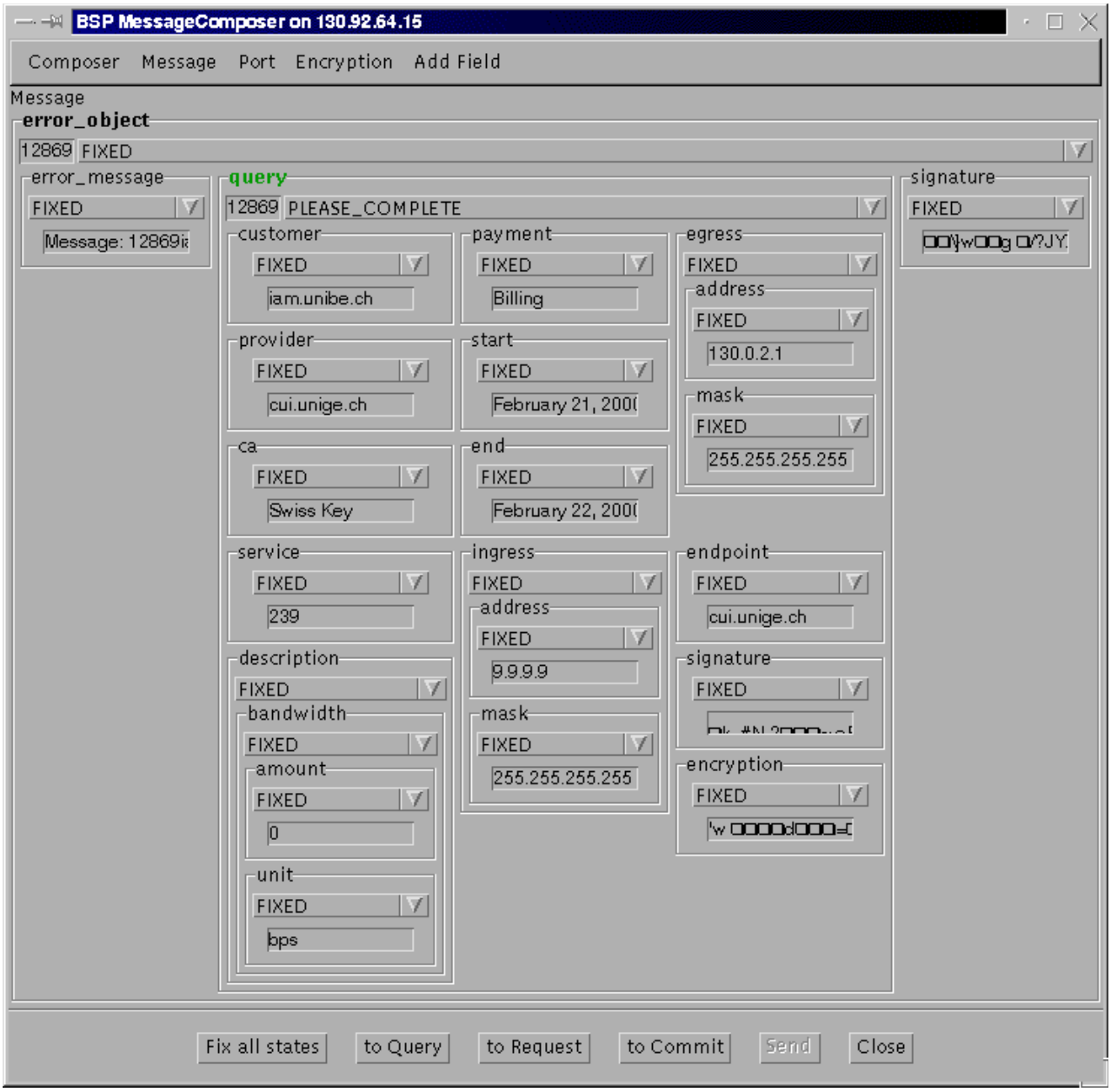


Abbildung 4.6: Darstellung einer Error-Message im GUI

4.2.3 OO-Parser

Der OO-Parser gehört zum Message Handler, wird aber aufgrund seiner Wichtigkeit in einem eigenen Abschnitt behandelt. Eigentlich ist der OO-Parser sogar der Komponente 'Autonomous Behavior' zuzuordnen. OO steht für **o**bjekt**o**rientiert und wird hier speziell hervorgehoben, da einige Vorteile objektorientierter Programmierung sehr deutlich zum Vorschein kommen.

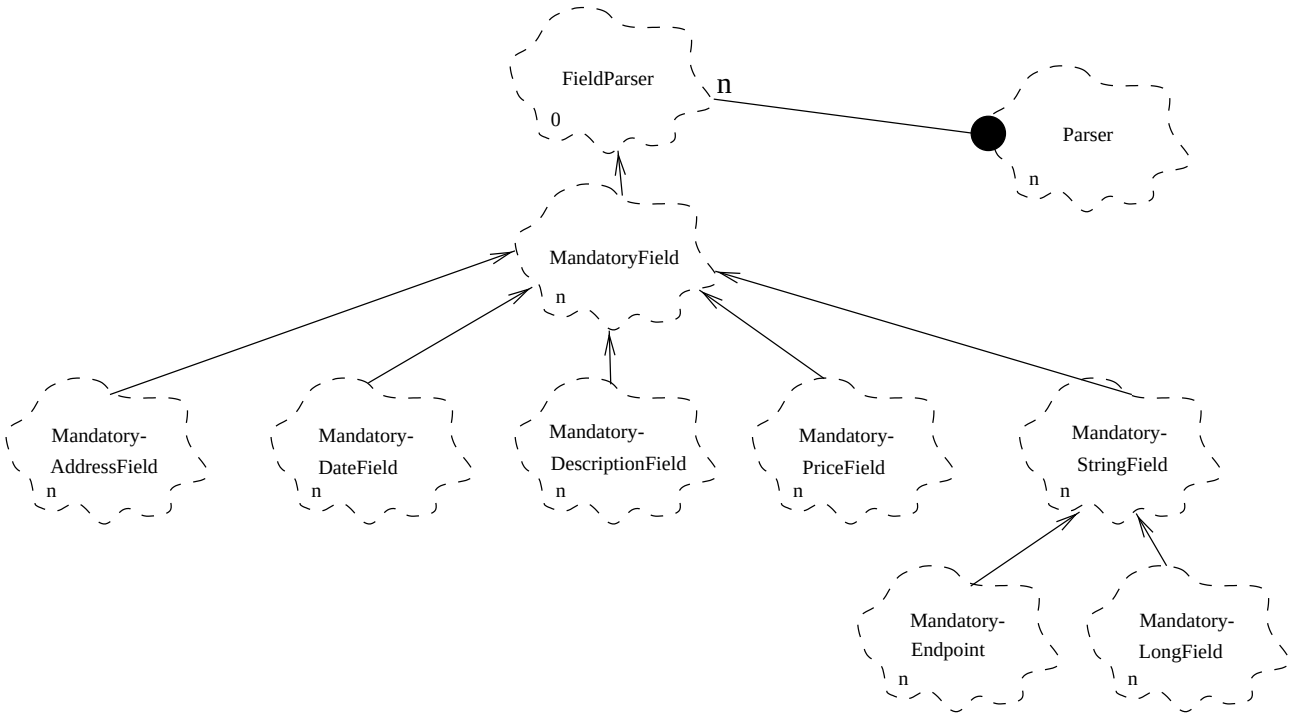


Abbildung 4.7: Parser Hierarchie

Wie in Abbildung 4.7 ersichtlich, besteht der Parser, welcher dem Message Handler die Überprüfung der Inhalte der Messages abnimmt, aus einer Vielzahl von Field Parsern. Diese Field Parser stellen für jeden Object-Typ wenn nötig eigene Funktionen zur Verfügung. Jeder Field Parser wird mit den Inhalten initialisiert, welche beim Parsen akzeptiert werden. Die Liste der zulässigen Inhalte kann auch nach der Initialisierung ergänzt werden.

Der Parser erfüllt im Wesentlichen folgende drei Aufgaben:

1. Falls ein Object-Typ eine bestimmte Struktur (bestimmte Objects) in seiner Collection verlangt, wird überprüft, ob diese Objects vorhanden sind. Fehlen einige Objects, werden diese hinzugefügt und mit je einem gültigen Inhalt versehen.

2. Falls der Object-Status 'fixed' ist, wird der Inhalt der Collection überprüft, indem dieser Inhalt mit den verschiedenen Initialisierungswerte verglichen wird.

3. Falls der Object-Status 'please complete' ist, ergänzt der Parser das Object in der Collection mit einer Auswahl an gültigen Inhalten oder aber mit einer Auswahl gültiger Strukturen (Selection).

Mit diesen drei Aufgaben kann in einem ersten Schritt die Struktur eines Objects oder einer Message erfragt, in einem zweiten Schritt die Objects dieser Struktur mit dem Status 'please complete' versehen und damit gültige Inhalte in Erfahrung gebracht und schliesslich in einem dritten Schritt durch Angabe des Status 'fixed', z.B. bei einer Request-Message, die Überprüfung einer gesamten Message erreicht werden.

4.2.4 SLA Handling: ebb.SLAHandler

Das effektive Bandbreitenmanagement und die Verwaltung der SLAs wird im SLA Handler vorgenommen. Der SLA Handler hat folgende drei Hauptaufgaben, welche nachfolgend einzeln erläutert werden:

1. Verwalten der Ressourcen, d.h. konkret feststellen, welche Ressourcen vorhanden und wie diese belegt sind.
2. Überwachung der Zustände der SLAs.
3. Synchronisation von Requests und Commits.

4.2.4.1 Verwalten der Ressourcen

Bei der Verwaltung von Ressourcen muss zuerst festgestellt werden, welche Ressourcen vorhanden sind. Diese Informationen bezieht ein External Bandwidth Broker über den IBB von den CDs. Da der IBB noch nicht implementiert ist, kann keine Initialisierung erfolgen. Deshalb wurde die Ressourcenverwaltung im Rahmen dieser Arbeit nicht implementiert.

4.2.4.2 Überwachung der SLA-Zustände

Die Entstehung eines SLAs geschieht in zwei Schritten. Aus der Sicht eines Customers wird zuerst eine entsprechende Anfrage (Request oder Commit) gestellt und nach Erhalt der positiven Antwort ein SLA erzeugt. Aus der Sicht eines Providers besteht der erste Schritt aus dem Erhalt einer Anfrage und der zweite Schritt aus einer nachfolgenden Übermittlung der positiven Antwort. Damit sind vom BB aus gesehen, welcher sowohl Customer wie auch Provider sein kann, vier Zustände möglich:

1. Anfrage abgesendet
2. Anfrage erhalten
3. Antwort abgesendet
4. Antwort erhalten

1. und 4. bilden ein erstes Paar, 2. und 3. ein zweites solches Paar. Die vorliegende Implementierung verlangt vom Message Handler die Meldung all dieser Zustände und speichert nebst dem Inhalt des SLAs auch die MD5- oder PGP-Signaturen der zugehörigen Anfrage und

Antwort. Damit ist ein vollständiger Nachweis möglich, sogar wenn der eine Vertragspartner keine Informationen mehr besitzt.

Die Überwachung der Zustände heisst auch, dass auslaufende SLAs rechtzeitig erkannt werden. Einerseits kann der BB die entsprechenden Änderungen der Konfigurationen auf den Routern initiieren oder überprüfen. Andererseits kann der BB den betroffenen Kunden entweder ein Anschluss-SLA vorschlagen oder die Abschaltung des Dienstes mitteilen.

4.2.4.3 Synchronisation von Requests und Commits

Die Synchronisation zwischen Requests und Commits der vorliegenden Implementierung erfolgt statisch. Dabei wird pro Commit ein Request mit identischem Inhalt verlangt. Die Referenzierung eines Requests durch ein Commit erfolgt durch Angabe des Identifiers des Requests. In Kapitel 2.2 wird angedeutet, dass diese statische Synchronisation nicht notwendig ist. Durch entsprechende Funktionalität zur dynamischen Synchronisation kann den Betreibern von Service Brokern viel Flexibilität bei der Gestaltung der angebotenen Dienste ermöglicht werden, ohne dass alle Betreiber diese Flexibilität unterstützen müssen.

4.2.5 Benutzerinterface: ebb.MasterInterface

Das MasterInterface soll dem Netzadministrator ermöglichen, den Betrieb des BBs zu überwachen, während des Betriebs Einstellungen zu verändern und alle Messages, welche nicht automatisch verarbeitet werden konnten, manuell zu verarbeiten. Grundsätzlich sind zwei Arten von Benutzerinterface vorgesehen: ein Grafisches User Interface (GUI) und ein Command Line Interface (CLI).

Im vorliegenden Prototyp ist das CLI nur sehr rudimentär realisiert und unterstützt ausser des sauberen Herunterfahrens des Brokers keine weiteren Funktionen. Die Überwachung des Brokers kann anhand des Logfiles erfolgen.

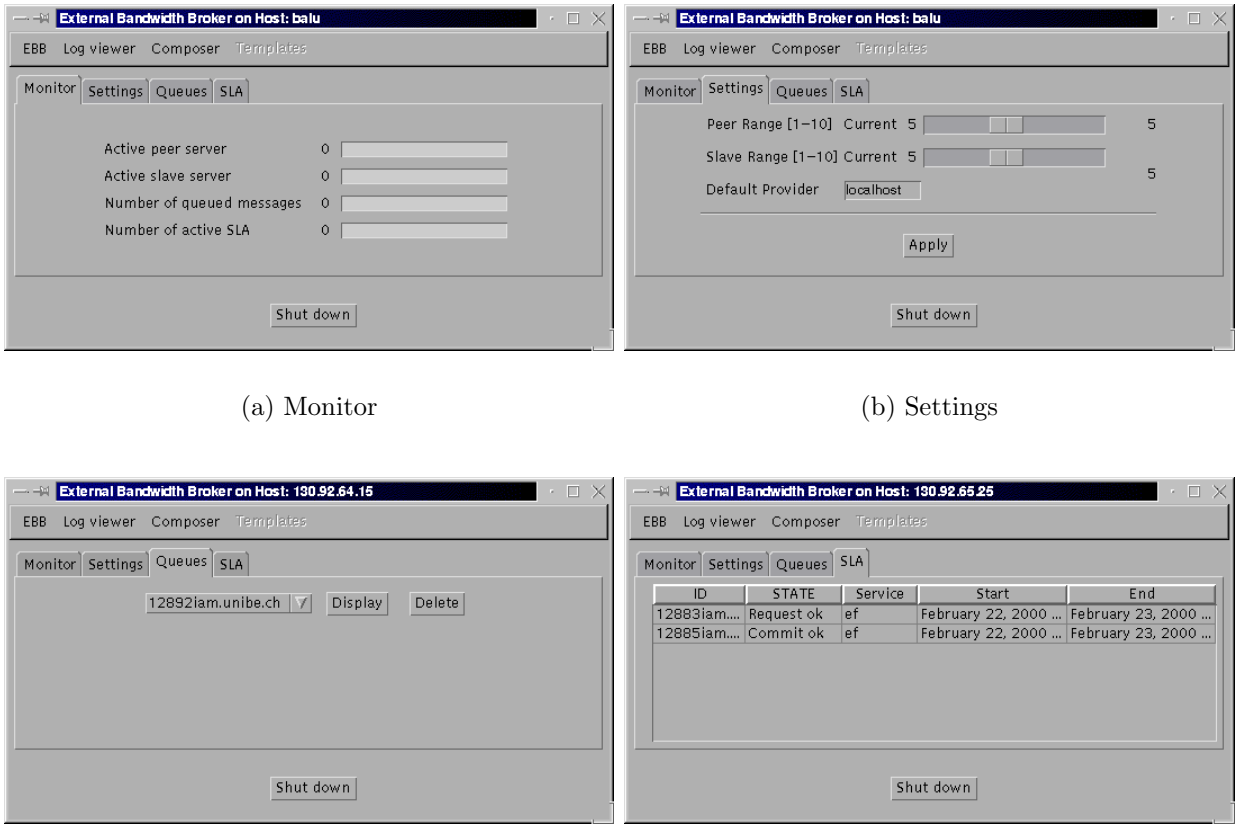
Das im Prototyp realisierte GUI besteht aus drei Fenstern: dem Master GUI, dem Composer GUI und einem Online Log.

4.2.5.1 Master GUI

Das Master GUI erlaubt nebst dem sauberen Herunterfahren des Brokers die Anzeige des Composer GUIs und des Online Logs (Log viewer). In Abbildung 4.8 (a) ist das Master GUI ersichtlich, welches durch Statusanzeigen eine Übersicht über die Anzahl aktiver Client- und Server-Threads, über die Anzahl unverarbeiteter Messages und über die Anzahl aktiver SLAs gibt. Die angezeigten Parameter sind im realisierten Prototyp als Beispiele zu betrachten und sollen lediglich aufzeigen, welche Art von Statusanzeigen möglich sind. Eine weitere Statusanzeige ist die SLA-Tabelle der Abbildung 4.8 (d), welche eine Auswahl von Parametern aktiver SLAs anzeigt.

Die Abbildung 4.8 (b) zeigt einige Parameter, welche im laufenden Betrieb, ohne Unterbrüche zu verursachen, verändert werden können. Auch hier sind einige Parameter gewählt, welche für die Präsentation des Prototyps oder zu Testzwecken verwendet wurden.

In Abbildung 4.8 (c) ist das Panel ersichtlich, welches die Bearbeitung von Queues erlaubt. Der Prototyp verfügt über eine Queue, welche alle unverarbeiteten Messages in Echtzeit aufnimmt.



(a) Monitor

(b) Settings

(c) Queues

(d) SLAs

Abbildung 4.8: Master GUI

Nach Auswahl einer Message kann diese mittels 'Display' im Composer GUI angezeigt oder mittels 'Delete' gelöscht werden.

4.2.5.2 Composer GUI

Das Composer GUI soll die dynamische Darstellung und Verarbeitung von Messages erlauben. Die Verarbeitung besteht darin, den Status von Objects oder Inhalte von Lists zu verändern. Die Ergänzung einer Ordered Collection mit weiteren Objects soll ebenso möglich sein. So können neue Messages erzeugt werden und bestehende Messages aus der Queue des Master GUIs bearbeitet werden. Die Verschachtelung der Objects, welche durch das BSP-Protokoll vorgegeben ist, findet sich in der GUI Darstellung wieder und wird mittels grafischer Elemente verdeutlicht (siehe Abbildung 4.10). Die Verschachtelung von Messages wird in der Error-Message der Abbildung 4.6 auf Seite 63 am Deutlichsten hervor.

Darstellung von Objects
Wie in Abbildung 4.9 dokumentiert, wird jedes Object in einem eigenen Rahmen dargestellt, welcher den Object-Namen enthält. Der Object-Name wird je nach Status bei 'please complete' grün, bei 'erroneous' rot und ansonsten schwarz geschrieben. Messages-Namen werden fett hervorgehoben.

Bei Objects mit Status 'fixed' wird darauf verzichtet, die Veränderung des Inhalts der Col-

lection zu verunmöglichen, obschon eine solche Veränderung nicht gewünscht wird und im allgemeinen wenig Sinn macht. Die entsprechende Funktionalität, welche Objects mit Status 'fixed' so darstellt, dass keine Veränderungen mehr möglich sind, ist jedoch bereits implementiert, so dass dieses Verhalten in einer weiteren Entwicklungsstufe des BBs durch einfache Einstellungen zur Verfügung gestellt werden kann.

Innerhalb des Object-Rahmens werden eine Auswahlbox für den Status und der Inhalt der Collection übereinander dargestellt. Bei Messages erfolgt zusätzlich die Angabe des Identifiers, welcher im Gegensatz zu den Object-Zuständen und Lists nicht verändert werden kann. Durch Änderungen des angezeigten Status in der Auswahlbox wird der ausgewählte Status für das angezeigte Object geändert. Diese Änderung wirkt sich jedoch erst beim Versenden der Message effektiv aus und widerspiegelt sich deshalb nicht unmittelbar in der Darstellung.

Abbildung 4.9: Composer GUI

Darstellung von Ordered Collections
Besteht die Ordered Collection aus mehreren Objects, werden diese Objects in einer Spalte übereinander dargestellt, solange die vorgegebene maximale Höhe des Fensters nicht erreicht ist. Enthält die Ordered Collection zu viele Objects, so dass diese übereinander dargestellt werden können, werden zusätzliche Spalten erzeugt, was in Abbildung 4.10 dargestellt ist.

Darstellung von Selections
Eine Auswahl von Objects (Selection) wird durch den ersten Eintrag in der Collection dargestellt. Bei einfachen Strukturen (siehe Object 'ca' in Abbildung 4.10) werden alle Inhalte der Lists in einer editierbaren Auswahlbox gleichzeitig dargestellt. Falls die Strukturen der Auswahlmöglichkeiten (Elemente der Collection) komplex sind (siehe Object 'ingress' in Abbildung 4.10), wird die Auswahl mittels einer zusätzlichen Auswahlbox (Choices) ermöglicht und nur eine Auswahlmöglichkeit dargestellt. Durch die Wahl eines anderen Elementes der Choice-Auswahlbox wird in der zugehörigen Object-Darstellung die Änderung unmittelbar vollzogen und dieses neue Element dargestellt.

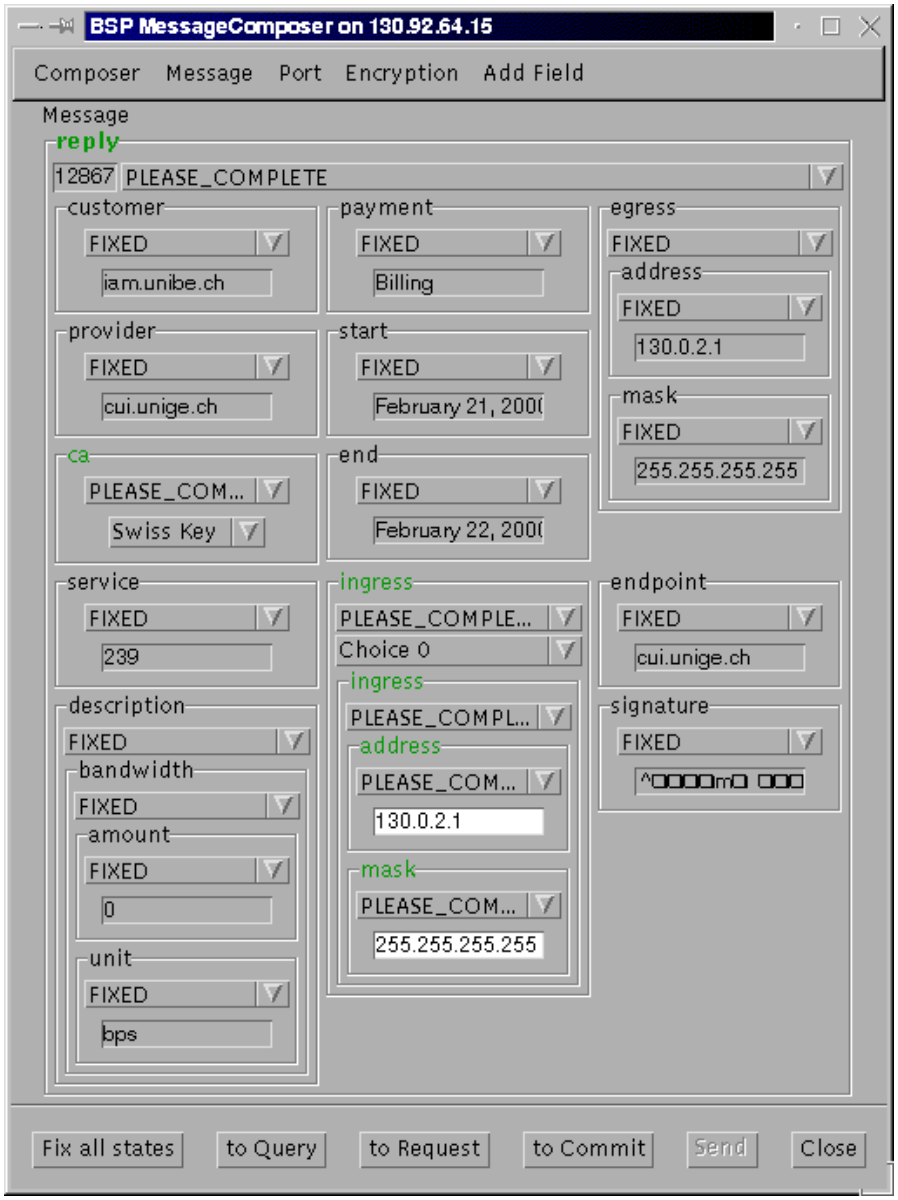


Abbildung 4.10: Composer mit Selections

Darstellung von Lists

Wie in Abbildung 4.10 ersichtlich, werden Lists je nach Codierung unterschiedlich dargestellt. Alle Lists sind editierbar, obschon die Darstellungen in der BSP-Syntax, in der internen BSP-Darstellung und im GUI sehr unterschiedlich sein können. Beispielsweise wird das Datum, welches in der BSP-Syntax eine bytecodierte Zahl ist, intern als 'long' gespeichert und im GUI in der üblichen lokalen Notation und Sprache dargestellt. Diese Darstellung ist intuitiv editierbar und wird für die Generierung der DateList entsprechend analysiert und umgesetzt (geparst). Das Datum hat somit drei Darstellungsformen:

1. Als bytecodierte Hexadezimaldarstellung in der BSP-Syntax (BSP-Byte-Stream).
2. Als Datentyp 'long' in der internen Darstellung des BSP-Objects.
3. In der lokalen ASCII-Darstellung (Sprache und Anordnung der Elemente) zur Verarbeitung im GUI.

Diese mehrfachen Darstellungsformen sind für alle Objects bzw. Lists notwendig. Die Umsetzung der ersten in die zweite Darstellungsform erledigen die Konstruktoren beim Einlesen jedes BSP-Byte-Streams bzw. entsprechende Funktionen beim Schreiben des entsprechenden BSP-Byte-Streams von jedem BSP-Object. Die Umsetzung der internen Darstellung eines BSP-Objects in die ASCII-Darstellung im GUI erledigt eine Object-spezifische Funktion ('display'). Die symmetrische Operation, d.h. die Umsetzung von ASCII-Eingaben im GUI in die interne Darstellung, erledigt ebenso eine Object-spezifische Funktion ('update'). Es ist denkbar, um die Object-Strukturen nicht zu überlasten, diese Funktionalität in ein eigenes Displayer-Objekt auszulagern. Damit wäre auch eine bessere Flexibilität und Erweiterbarkeit gewährleistet.

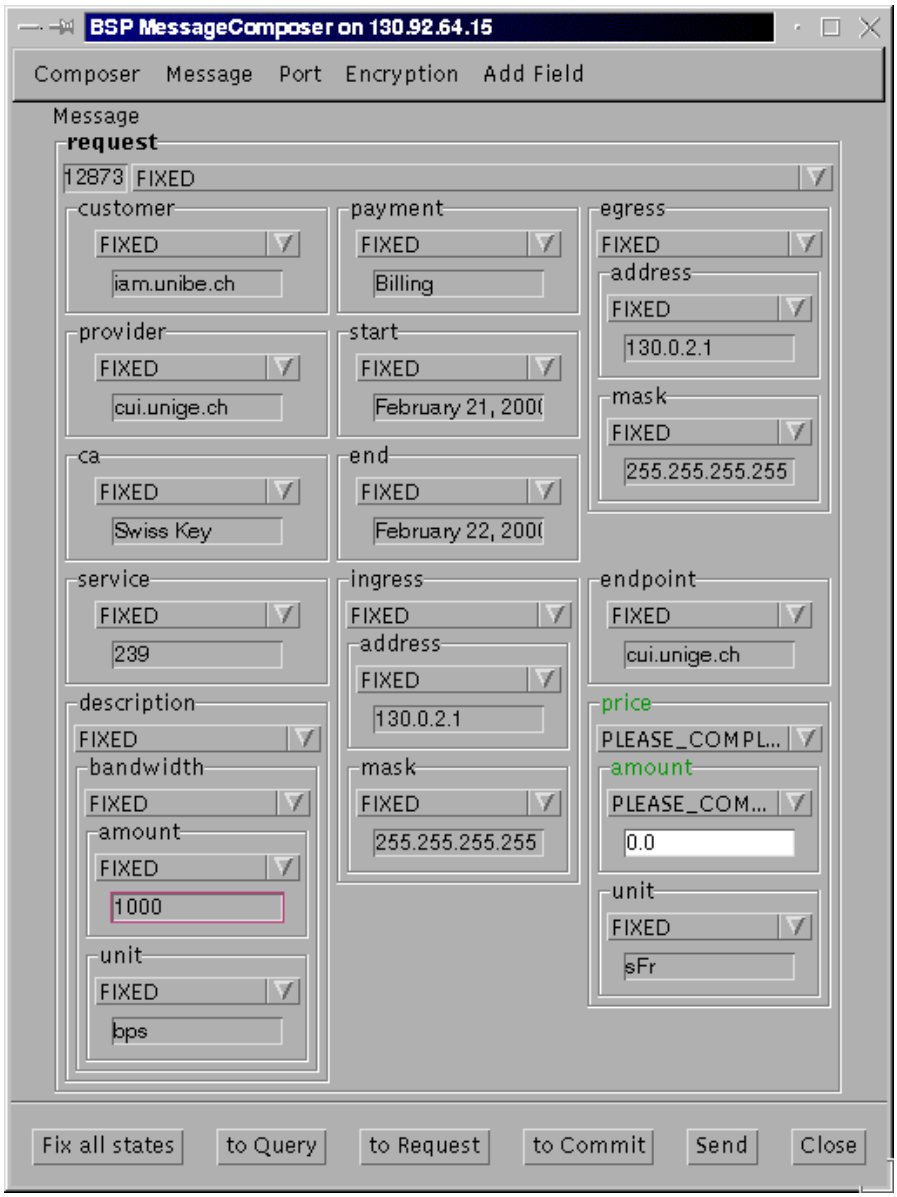


Abbildung 4.11: Price Query

Menübeschreibung des Composer GUIs:

- Composer:
  Das Composer-Menü hat den Eintrag 'Close', welcher erlaubt, das Fenster zu schliessen.

- Message:
  Das Message-Menü erlaubt die Erzeugung neuer Messages, die Umwandlung der angezeigten Message in einen anderen Typ und das Absenden der angezeigten Message. Ausser der Erzeugung neuer Messages sind diese Funktionen auch über Schaltflächen erreichbar.

- Port:
  Das Menü 'Port' ermöglicht die Auswahl zwischen dem Peer- und dem Slave-Port für die Übermittlung der angezeigten Message und damit die Unterscheidung zwischen Intra- und Inter-Domain Messages.

- Encryption:
  Die Encryption erlaubt den Verschlüsselungsalgorithmus auszuwählen. So kann für die Übermittlung der angezeigten Message ganz auf eine Verschlüsselung verzichtet, MD5 oder PGP gewählt werden. Unabhängig von dieser Wahl wird beim Versenden die Message mit einer MD5-Signatur versehen.

- Add Field:
  Unter diesem Menüpunkt sind Einträge zu finden, welche eine Ergänzung der angezeigten Message erlauben. So ist es beispielsweise möglich, einer Query-Antwort (Reply) am Ende der Ordered Collection ein Price-Object hinzuzufügen und damit eine Preis-Anfrage zu erstellen (siehe Abbildung 4.11). Diese Technik erlaubt es auch, eine vollständige Message von Hand zu erzeugen. Die Grundstruktur (Customer, Provider und Message-Typ) liefert die Funktion 'new Message'. Die Ordered Collection kann nun mit entsprechenden Objects ergänzt werden, bis alle gewünschten Parameter vorhanden sind. Ohne jemals eine Query an einen Provider gesendet zu haben. Vorausgesetzt die verlangte Struktur und die akzeptierten Inhalte sind bereits bekannt, kann damit z.B. eine gültige Request-Message erzeugt werden. (Die Möglichkeiten dieser Technik sind im vorliegenden Prototyp nur teilweise implementiert).

4.2.5.3 Online Log

Das Online Log (siehe Abbildung 4.12) ermöglicht es, die aktuellen Meldungen, welche in das

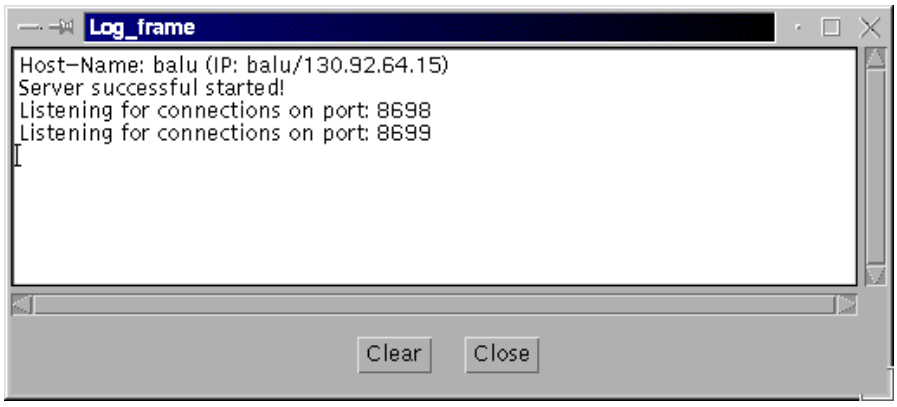


Abbildung 4.12: Online Log

Logfile geschrieben werden, live mitzuverfolgen. Im anschliessenden Auszug eines Logfiles soll

die Darstellung und der Informationsgehalt ersichtlich werden:

```
Mon Feb 21 15:15:27 CET 2000 : Host-Name: 130.92.65.25 (IP: milou/130.92.65.25)
Mon Feb 21 15:15:27 CET 2000 : Listening for connections on port: 8698
Mon Feb 21 15:15:27 CET 2000 : Listening for connections on port: 8699
Mon Feb 21 15:15:27 CET 2000 : Server successful started!
Mon Feb 21 15:16:36 CET 2000 : Server 1 is: Socket[addr=balu/130.92.64.15,port=33474,localport=8698]
Mon Feb 21 15:16:38 CET 2000 : Message 12863 received as encrypted message with ID 12864
Mon Feb 21 15:16:41 CET 2000 : New client: Socket[addr=balu/130.92.64.15,port=8698,localport=58402]
Mon Feb 21 15:16:42 CET 2000 : Message 12863 send as encrypted message with ID 3935
Mon Feb 21 15:16:56 CET 2000 : Server 2 is: Socket[addr=balu/130.92.64.15,port=33475,localport=8698]
Mon Feb 21 15:16:58 CET 2000 : Message 12865 received as encrypted message with ID 12866
Mon Feb 21 15:17:02 CET 2000 : New client: Socket[addr=balu/130.92.64.15,port=8698,localport=58403]
Mon Feb 21 15:17:05 CET 2000 : Message 12865 send as encrypted message with ID 3936
Mon Feb 21 15:17:53 CET 2000 : Server 3 is: Socket[addr=balu/130.92.64.15,port=33478,localport=8698]
Mon Feb 21 15:17:55 CET 2000 : Message 12867 received as encrypted message with ID 12868
Mon Feb 21 15:18:00 CET 2000 : New client: Socket[addr=balu/130.92.64.15,port=8698,localport=58406]
Mon Feb 21 15:18:02 CET 2000 : Message 12867 send as encrypted message with ID 3937
Mon Feb 21 15:19:14 CET 2000 : Server 4 is: Socket[addr=balu/130.92.64.15,port=33481,localport=8698]
Mon Feb 21 15:19:16 CET 2000 : Message 12869 received as encrypted message with ID 12870
Mon Feb 21 15:19:18 CET 2000 : W -> Message: 12869iam.unibe.ch => Faulty query: IP address does not match profile.
Mon Feb 21 15:19:20 CET 2000 : New client: Socket[addr=balu/130.92.64.15,port=8698,localport=58407]
Mon Feb 21 15:19:23 CET 2000 : Message 12869 send as encrypted message with ID 3939
Mon Feb 21 15:23:11 CET 2000 : Server 5 is: Socket[addr=balu/130.92.64.15,port=33486,localport=8698]
Mon Feb 21 15:23:15 CET 2000 : Message 12873 received as encrypted message with ID 12874
Mon Feb 21 15:23:17 CET 2000 : W -> Message: 12873iam.unibe.ch => Faulty request: Field price must be fixed for re-
quests.
Mon Feb 21 15:23:20 CET 2000 : New client: Socket[addr=balu/130.92.64.15,port=8698,localport=58414]
Mon Feb 21 15:23:22 CET 2000 : Message 12873 send as encrypted message with ID 3941
Mon Feb 21 15:24:06 CET 2000 : Server 6 is: Socket[addr=balu/130.92.64.15,port=33489,localport=8698]
Mon Feb 21 15:24:09 CET 2000 : Message 12877 received as encrypted message with ID 12878
Mon Feb 21 15:24:14 CET 2000 : New client: Socket[addr=balu/130.92.64.15,port=8698,localport=58417]
Mon Feb 21 15:24:15 CET 2000 : Message 12877 send as encrypted message with ID 3942
```

4.2.6 Datenbankanbindung

Die Datenbankanbindungen wurden aufgrund des beträchtlichen Umfanges dieser Arbeit nicht umgesetzt. Dies hat zur Folge, dass alle Daten im Hauptspeicher gehalten werden. Deshalb empfiehlt es sich den Broker mit der Java-Option -mx 64'000'000 zu starten, um zusätzliche 64 MB Hauptspeicher zur Verfügung zu stellen.

4.2.7 Autonomous Behavior

Die Autonomie des Brokers besteht in dieser Implementierung aus der automatischen Verarbeitung von Queries, Requests und Commits sowie aus einem guten Error-Handling mit kommentierten Meldungen in das Logfile und an den Absender. Zu dieser automatischen Verarbeitung gehören insbesondere die koordinierte Erzeugung von SLAs mit benachbarten Providern, falls der Endpoint nicht im eigenen Netz ist. BSP-Meldungen, welche nicht automatisch verarbeitet werden können, werden in eine Queue geleitet, woher sie manuell weiterverarbeitet werden können. Zur Autonomie gehören aus logischer Sicht auch die Fähigkeiten des Parsers, welche in Kapitel 4.2.3 erläutert worden sind.

Kapitel 5

Resultate

5.1 Konkretisierung und Umsetzung der Konzepte

5.1.1 Service Broker Modell

Das Design des vorliegenden Prototyps ermöglicht, einen beliebigen Broker zu implementieren. Durch den Austausch oder durch eine Erweiterung des SLA Handlers und eine alternative Initialisierung des Parsers bekommt der Broker eine andere Aufgabe. Der implementierte SLA Handler verwendet Schnittstellen zum Message Handler, welche speziell auf die Aufgaben als Bandwidth Broker zugeschnitten sind. Eine allgemeine Definition ist jedoch möglich, so dass durch die Definition einer abstrakten Klasse die Schnittstellen zum Message Handler standardisiert werden könnten. Durch die bei der OO-Programmierung möglichen Vererbung könnte ähnlich wie beim Parser das Design verallgemeinert und dadurch die Flexibilität der Implementierung nochmals erhöht werden.

Durch Parametrisierung können die Interfaces einzeln gestartet werden, so dass der External Bandwidth Broker z.B. ohne GUI hochgefahren werden kann. Die Parametrisierung kann dabei durch Änderungen des Konfigurationsfiles erfolgen. Diese Parametrisierung erlaubt zudem einem Internal Service Broker, welcher nur ein internes Communication Interface benötigt, auch nur das Slave Interface hochzufahren.

5.1.2 Two Phase Commitment

Das Two Phase Commitment wurde im Message Handler und im SLA Handler implementiert, wobei der Message Handler die unterschiedliche Behandlung je nach Phase bzw. je nach Message-Typ übernimmt. Die Koordination zwischen den Phasen erfolgt im SLA Handler.

Das Forwarding wird im Message Handler des Prototyps statisch implementiert. Für ein dynamisches Forwarding ist die zugehörige Funktionalität in den SLA Handler auszulagern, um die dynamische Beziehung zwischen Requests und Commits und die dynamische Beziehung zwischen den SLAs eines Pfades zu ermöglichen.

5.1.3 BSP-Protokoll

Das in Kapitel 2 vorgestellte BSP-Protokoll wurde implementiert und erste Erfahrungen damit gesammelt. Um die Flexibilität des Protokolls und die Flexibilität des GUIs zur Darstellung von BSP-Objects zu verbinden, wurde die interne Darstellung der BSP-Objects konsequent in allen Komponenten verwendet.

Die Verwendung des Identifiers bei Objects ist nicht klar, womit die Definition erst im abgeleiteten Objekt 'Message' erfolgen sollte. Denn bei Messages ist die eindeutige Vergabe und die Verwendung des Identifiers wohldefiniert.

Um gewisse Redundanzen vermeiden zu können, wäre es sinnvoll, die Gestaltung der Collections zu überdenken. So wären Selections als Liste von Lists modellierbar, statt wie im vorliegenden Prototyp als Liste von Objects.

Im Prototyp wurde die Funktionalität zur Darstellung der BSP-Entitäten in jeder Klasse implementiert. Nebst eindeutigen Vorteilen durch die automatische Erkennung, um welche Instanz es sich handelt, und die Konzentration aller Funktionen eines BSP-Elementes in einer Klasse hat sich die Handhabung besonders in Bezug auf den Speicherverbrauch als schwerfällig erwiesen, da die Darstellungselemente für jede Message lokal vorhanden sein müssen. Die gesamte Funktionalität im Zusammenhang mit der Darstellung der BSP-Entitäten sollte in eine eigene Klasse (Displayer-Objekt) ausgelagert werden, ähnlich der Auslagerung des Parsers beim Message Handler. Denn sowohl die Darstellung der BSP-Entitäten im GUI und die Übertragung der Eingaben des GUIs in die interne Darstellung sind Parsing-verwandte Aufgaben. Ein solches Displayer-Objekt könnte einen ähnlichen Aufbau wie das Parser-Objekt haben.

5.2 Implementierung

5.2.1 Programmiersprache Java

Die gewählte Implementierung in Java hat einige Vorteile in Bezug auf die Verwendung von Klassen der java.net-Library, welche besonders die Handhabung der TCP-Verbindungen vereinfacht und dadurch ein Ziel dieser Arbeit, einen weitgehend funktionierenden Prototyp zu erstellen, erst ermöglicht hat. Leider hat sich herausgestellt, dass die Parallelität und in diesem Zusammenhang die Zuverlässigkeit nicht gegeben waren. Wenn ein Broker in sehr kurzen Abständen mehrere Messages je in eigenen Threads einem zweiten Broker senden wollte, wurden diese angeblich gesendet, ohne irgendeine Fehlermeldung zu verursachen. Diese Messages wurden jedoch nicht alle empfangen, obschon durch die Verwendung einer TCP-Verbindung dies hätte sichergestellt werden sollen. Die Möglichkeiten zur Eingrenzung des Fehlers sind in Java sehr knapp, so dass die Ursache nicht eruiert werden konnte. Das Problem lag sehr wahrscheinlich im Zusammenspiel der Java-Library-Klassen mit den Betriebssystemfunktionen. Das Problem wurde im Prototyp so behoben, dass ein minimales Handshaking vor und nach der Übermittlung der Messages implementiert wurde.

Die Verwendung von Java hat die Programmierung der GUIs stark vereinfacht, da hierzu geeignete Klassen der Swing-Bibliothek zur Verfügung standen. Nebst der Verwendung von klassischen Elementen wie Fenster, Menüs und Schaltflächen konnte auch auf Funktionen

zur Darstellung von Tabellen zurückgegriffen werden. Die sinnvolle Darstellung der BSP-Protokollelemente war mit den folgenden Darstellungselementen möglich:

- Panel mit zugehörigem Rahmen inklusive Beschriftung (JPanel.java)
- Layout-Manager zur Positionierung der Elemente (GridBagLayout.java)
- Editierbare Auswahlboxen (JComboBox.java)
- Editierbare Eingabefelder für Texteingaben (JTextField.java)

5.2.2 Portabilität

In Bezug auf die Portabilität haben sich alle Erwartungen erfüllt. Die vorliegende Implementierung wurde sowohl auf Sun Solaris wie auch auf Linux unter unterschiedlichen Window-Managern getestet. Auf diesen beiden Systemen erfolgte die Übersetzung sowohl mit Java 1 (Java Development Kit 1.1.8) und Swing (Version 1.1.1) wie auch mit Java 2 (Java Development Kit 1.2.2), wo die GUI-Bibliothek Swing integriert ist. Ausser kleinen Unterschieden in der Darstellung einiger Standardelemente sind bei der Ausführung weder nennenswerte Unterschiede in der Handhabung noch anderweitige Probleme aufgetreten.

Die Portabilität auf UNIX-fremde Betriebssysteme könnte in Bezug auf die Verschlüsselung mittels PGP nicht gewährleistet sein, da die PGP-Funktionalität mittels Skripts genutzt wird und nicht in Java portiert wurde.

5.2.3 Flexibilität

Wie in Kapitel 5.1.1 bereits erwähnt, weist die vorliegende Implementierung wesentliche Flexibilitätsmerkmale auf, welche jedoch noch erweitert werden könnten. Der OO-Ansatz hat dabei seine Vorteile voll ausspielen können.

5.2.4 Prototyp

Der Prototyp kann zwischen zwei oder mehreren Brokern SLAs aushandeln und die SLAs eines Pfades auf statische Weise koordinieren. Die statische Koordination zwischen Requests und Commits ist auch umgesetzt. Nachdem entsprechende SLAs ausgehandelt worden sind, sind die Parameter zur konkreten koordinierten Umsetzung in Routerkonfigurationen in den zuständigen BBs vorhanden, so dass effektive Dienste aufgesetzt werden könnten.

5.2.4.1 Funktionalität

Der Prototyp erlaubt es, mittels einer Message, welche nur aus Customer- und Provider-Namen besteht, eine Anfrage (Query) an den Provider zu stellen. Der Provider antwortet automatisch mit einer Liste von angebotenen Diensten. Dabei setzt der Provider die Strukturen für jeden angebotenen Dienst in die BSP-Message ein. Der Customer kann nun einen Dienst auswählen und die Objects einer dieser Strukturen mit dem Status 'please complete' versehen. Diese Message kann wiederum als Query dem Provider gesendet werden, welcher für jedes Object eine

Auswahlliste generiert, diese in die BSP-Message einfügt und die BSP-Message dem Customer zurücksendet. Der Customer kann nun durch Auswahl eines Eintrages in jeder Auswahlliste den Dienst definieren und z.B. durch Hinzufügen eines Price-Objects eine Preis-Anfrage stellen. Sobald die Antwort auf diese Preis-Anfrage eintrifft, ist der Customer in der Lage, durch Umwandlung der Antwort in ein Request, ein Request-SLA zu initiieren. Nach Erhalt der Antwort auf ein Request besteht ein Request-SLA zwischen dem Customer und dem Provider, was bereits Kosten nach sich zieht, weil der Dienst soweit möglich eingerichtet wird oder zumindest die entsprechenden Ressourcen freigehalten werden. Das Zustandekommen des SLAs ist durch den Eintrag in der SLA-Tabelle des Master GUIs ersichtlich. Durch Vereinbarung eines Commit-SLAs auf dieselbe Weise wie für Request-SLAs kann nun die Freischaltung des Dienstes verlangt werden. Dieses SLA erscheint ebenso in der SLA-Tabelle.

5.2.4.2 Statische Zuordnungen

Die Umsetzung zwischen dem BSP-Namen des EBBs und seiner IP-Adresse geschieht statisch anhand einer fest codierten Tabelle. Dies war der einzige Weg, um in einer Testumgebung ohne Zugriff auf entsprechend konfigurierte DNS-Server in den BSP-Messages bereits DiffServ- statt DNS-Namen verwenden zu können.

Jedes Commit setzt ein entsprechendes Request voraus. Die statische Zuordnung erfolgt durch Angabe des Identifiers des Requests im Commit.

Die Synchronisation zwischen SLAs auf einem Pfad ist statisch implementiert.

5.3 Related Work

5.3.1 Bandwidth Broker for IP-Networks

Stephan Heuscher hat im Winter 1998/99 am Institut für Technische Informatik und Kommunikation der ETH in Zürich eine Diplomarbeit mit dem Titel 'Bandwidth Broker for IP-Networks' geschrieben. In dieser Arbeit wird mittels NS-Simulationen untersucht, wie dem schwankenden Bedürfnis an Bandbreite im Tagesverlauf Rechnung getragen werden kann. Es sollen klassische langfristige (statische) Verträge mit fixer Bandbreite durch kurzfristige Verträge (z.B. mit einer Dauer von wenigen Stunden) ersetzt werden. Um dies zu ermöglichen sollen zwischen sogenannten Bandwidth Broker elektronische Verträge ausgehandelt werden. Der Autor geht davon aus, dass ISPs neue Dienste anbieten, welche besondere QoS-Eigenschaften aufweisen und ein Kunde bei Bedarf solche Dienste abonnieren kann. In der Arbeit wird nicht darauf eingegangen, wie Provider diese Dienste erbringen, sondern nur wie solche Dienste elektronisch koordiniert werden können.

Grundsätzlich muss auch in diesem Ansatz in jeder Domain ein BB vorhanden sein. Es wird davon ausgegangen, dass die BBs auf der ganzen Welt in hierarchischer Weise angeordnet werden. Bei einer Anforderung eines Dienstes, welcher innerhalb derselben Domain erbracht werden kann, entscheidet der BB autonom auf welchem Pfad der Dienst erbracht wird. Bei einer Anforderung, welche einen Dienst über die Grenzen der eigenen Domain hinaus verlangt, muss der lokale BB eine entsprechende Anforderung einem übergeordneten BB stellen. Dieser übergeordnete BB fragt nun seine untergeordneten BBs an, wer zu welchen Konditionen den

Dienst erbringen kann. Der günstigste Anbieter erhält dann den Zuschlag. Ist kein untergeordneter BB in der Lage den Dienst zu erbringen, fragt auch dieser BB einen übergeordneten BB an usw. bis der 'oberste' erreicht wird. Einige BBs, zumindest aber dieser 'oberste' BB, sind nicht mehr fest einer Domain zugewiesen.

Genau diese hierarchische Anordnung der BBs sehe ich als grosser Nachteil, weil dies bedingt, dass sich der BB eines ISPs einem anderen (fremden) BB unterordnet und damit eine gewisse Selbständigkeit verliert. Da es im Internet keine strenge Hierarchie gibt, ist zudem zu untersuchen, wie sich das Konzept verhält, wenn der hierarchische Baum durch einen Graph ersetzt wird. Aus diesen Gründen denke ich, dass sich dieser Ansatz nicht durchsetzen wird.

Die Diplomarbeit von Stefan Heuscher ist nicht öffentlich zugänglich, so dass keine diesbezüglich Referenz möglich ist.

5.3.2 Resource Reservation Agents in the Internet

In [Reservation Agents] wird beschrieben, wie sogenannte Agents durch passive Teilnahme an einem Link-State Routing Protokoll wie z.B. OSPF die Topologie des Netzwerkes und die Eigenschaften entsprechender Link in Erfahrung bringen. Dieser 'passive' Ansatz scheint mir besonders für die Realisierung von Internal Service Broker bzw. für die Initialisierung oder Überwachung der Ressourcen-Matrix der External Service Broker interessant (siehe Kapitel 4.2.4.1).

5.3.3 Service Level Agreement Traders

Das Dokument 'Service Level Agreement Traders (SLAT) and Cost-based Routing for Differentiated Services' beschreibt eine Architektur zur Ressourcenverwaltung im DiffServ-Umfeld. Das Dokument beschränkt sich, wie die vorliegende Arbeit, im wesentlichen auf Inter-Domain QoS-Routing aber abstrahiert zudem die gesamte Inter-Domain Problematik sehr stark, indem diese ausgeklammert wird. Dabei liegt der Fokus der Arbeit im Vergleich und in der Analyse des Verhaltens von sogenannten Jumbo-Flows in einer NS-Simulation. Die Analyse basiert auf dem Vergleich des Routings mittels SLAT gegenüber einem DV (Distance Vector) Routing-Verfahren. Zum Austausch von Meldungen, um SLAs aufzusetzen, wurde ein einfaches SLAT-Protokoll (SLATP) entwickelt, wobei nur der rudimentäre Teil der Logik des Protokolls entwickelt aber kein reales Protokoll verwendet wurde. Die Kompatibilität des Verfahrens zu etablierten Intra-Domain Routing Protokolle wird in der Arbeit hervorgehoben, so dass Best-Effort-Flows weiterhin und parallel zu SLAT z.B. durch BGP gerouted werden können.

Die Strategien, welche rund um SLAT entwickelt und verglichen wurden, könnten in der Komponente Autonomous Behavior des in dieser Arbeit entwickelten BBs weiterverwendet werden und damit Analysen von SLAT in einem realen Umfeld ermöglichen.

Zu Beginn meiner Arbeit lag mir eine Version 'TIK Report No. 59, V1.0' vor, welche nun in der offenbar endgültigen Version mit ca. doppeltem Umfang vorliegt. Diese neuste Version trägt nun den Titel: 'Service Level Agreement Trading for the Differentiated Services Architecture' [SLAT]. In diesem Dokument ist eine interessante Literaturliste rund um die Themen SLA, DiffServ und QoS-Routing zu finden.

5.3.4 QBone

What is the QBone?

„Launched in October 1998, the QBone is an Internet2 initiative to build a testbed for new IP quality of service (QoS) technologies. New advanced network applications like remote instrument control, scientific collaboratories, and virtual classrooms will give our universities the tools that they need to fulfill their teaching and research missions in the coming century but only if the demands that these new applications place on the network can be met. The QBone testbed will initially implement the differentiated services (DiffServ) approach to QoS that is now taking shape within the IETF. DiffServ has great potential to overcome some of the complexities of earlier IP QoS architectures, but requires a great deal of implementation experience, engineering, and study before it will mature to offer production-quality QoS.“

Unter http://www.internet2.edu/qos/qbone/GBBAC.shtml ist eine Übersicht über weitere Bandwidth Broker Projekte zu finden, welche im Zusammenhang mit QoS oder mit DiffServ im Internet entstanden sind und woran z.T. noch gearbeitet wird. QBone http://www.internet2.edu/qos/qbone gibt zudem einen Überblick über Neuigkeiten und weitere Informationen rund um das Thema QoS und DiffServ. Besondere Beachtung verdient der Beitrag [CA*net II], welcher das Design und die Implementation eines Bandwidth Brokers mit zugehörigem Protokoll (BBTP) zu Ziel hatte.

5.4 Zusammenfassung und Ausblick

Die Umsetzung des Service Broker Models für DiffServ, des Two Phase Commitments und des BSP-Protokolls in einer objektorientierten Programmiersprache hat sich als geeignet erwiesen. Die Wahl von Java hat einige Vorteile aber auch wesentliche Nachteile gebracht, so dass ich empfehle, den Broker in einer anderen objektorientierten Programmiersprache wie z.B. C++ zu implementieren, um systemspezifischer, leistungsorientierter und zuverlässiger umsetzen zu können. Diese Eigenschaften sind für eine Client-Server-Anwendung und speziell für einen Broker von ausschlaggebender Bedeutung.

Bei der Umsetzung in C++ könnte sehr wahrscheinlich auf das Handshake bei der Übermittlung verzichtet werden und nur BSP-Messages über eine TCP-Verbindung gesendet werden, wie dies ursprünglich vorgesehen war. Die in Java sehr ausgeprägte Unterstützung im Tread Handling dürfte aber in C++ einen wesentlich grösseren Aufwand bedeuten.

Die Implementierung der GUIs in Java kann weiterempfohlen werden, wobei die Integration in eine Web-Umgebung und somit die Realisierung als Java-Applet ins Auge gefasst werden sollte.

Vieler BSP-Namen und viele BSP-Strukturen sind in dieser Arbeit definiert worden, wobei allgemeingültige Standardisierungen notwendig sind, um das Verständnis der Broker untereinander zu gewährleisten. Im vorliegenden Prototyp wurde weitgehend darauf verzichtet unterschiedliche Darstellungen im Protokoll, in der internen Darstellung und im GUI zu verwenden. Am Beispiel der Darstellung eines Datums sind die Möglichkeiten aufgezeigt worden. Die Standardisierung des Formats für ein Datum wurden im BSP-Protokoll als Anzahl Millisekunden seit dem 1.1.1970 0 Uhr 00 GMT definiert. Die Unterscheidung zwischen Standards im Protokoll und Standards in der internen Darstellung muss klar vollzogen werden. Diese Standards müssen zudem eineindeutig aufeinander abgebildet werden können. So könnte

in der internen Darstellung bei einem Datum statt GMT die lokale Zeit verwendet werden. Alternativ könnte diese Umsetzung auch nur zwischen interner Darstellung und dem GUI erfolgen.

Die Abhängigkeit zwischen der Sicherheit und der Rechenintensität der verwendeten Verfahren zur Ver- und Entschlüsselung wurde erläutert und Lösungen ansatzweise vorgeschlagen. Ein konkretes Konzept im Zusammenhang mit der Weiterentwicklung des BSP-Protokolls zu einem allgemeinen und sicheren Broker Signalisierungs Protokoll ist wünschenswert. So könnte ein Austausch von MD5 Passwörtern zwischen bekannten Brokern mittels spezieller PGP-codierten BSP-Messages in regelmässigen Zeitabständen erfolgen. Dadurch wird die Verwendung von MD5 statt PGP bei der Verschlüsselung und Identifikation aller anderen BSP-Messages möglich und somit die Leistungsfähigkeit eine Brokers wesentlich verbessert werden, ohne Einbussen bezüglich Sicherheit in Kauf nehmen zu müssen. Diese Funktionalität kann vollständig im Communication Interface und durch die Abbildung der sicherheitsrelevanten Parameter auf eine eigene Skala beim Übergang vom Communication Interface zum Message Handler ohne grosse Anpassungen erfolgen.

Die Koordination zwischen Anfrage und Antwort wurde für die Message-Typen Query, Request und Commit im Message Handler implementiert, wobei die automatische Zuordnung der Replies umgesetzt worden ist. Ein Ansatz, welcher nicht auf die erwähnten Message-Typen beschränkt ist, wäre sinnvoll, um die Erweiterbarkeit und Flexibilität der Implementierung zu erhöhen und somit auch andere Message-Typen koordiniert behandeln zu können.

Der Umfang der vollständigen Umsetzung dieser Konzepte hätte den Rahmen dieser Diplomarbeit gesprengt, so dass doch wesentliche Teile ausgelassen werden mussten. Die Entwicklung allgemeiner Konzepte zur Ressourcenverwaltung wie auch die Schaffung eines Regelwerkes für das Policy Interface wären weitere interessante Forschungsbereiche rund um das Thema Brokering im DiffServ-Umfeld.

Anhang A

Terminologie

- Admission Control
  Mechanismen zur Kontrolle von QoS-Parametern.
- Assured Services
  Bezeichnung für die DiffServ-Dienstklassen, bei welchen in Überlastungssituationen nur ein bestimmter Prozentsatz der Bandbreite garantiert ist.
- Backbone
  Netzwerk zur Interconnection anderer Netzwerke
- Bandwidth Broker
  Ein spezieller Broker für die Verwaltung der Bandbreite in einem Netzwerk (siehe Kapitel 3).
- Best Effort
  Verhalten von Netzelementen im Internet nach dem Motto „jeder macht so viel er kann“.
- Border Router
  Router an der Grenze eines LANs oder WANs, welcher mit anderen Border Router physisch verbunden ist.
- Broker
  Server, welche zur Verwaltung von bestimmten Ressourcen in einem Netzwerk, ihre Dienste anbieten.
- Collection
  Abstrakte Klasse des BSP-Protokolls. Sinngemäss eine Ansammlung von gleichartigen Objekten.
- Customer
  Dienstnutzer im DiffServ-Umfeld.
- Dequeueing
  Mechanismus oder Verfahren zum Entleeren von Queues.
- Domain
  Domäne bzw. Netzwerk eines ISP. Meist ist eine Domain identisch mit einer DNS-Domain.

- Dropen
  Fallen lassen bzw. nicht beachten von IP-Paketen.

- Drop Probabilities
  Wegwerfwahrscheinlichkeiten

- Egress Router
  Border Router zur speziellen Bearbeitung von abgehenden IP-Paketen.

- Entity
  Elementare abstrakte Klasse des BSP-Protokolls. Jede Klasse des BSP-Protokolls ist von einer Entity abgeleitet.

- Flow
  Anwendungsdatenfluss

- Frame Relay
  Verbindungsorientierte paketvermittelnde Technologie, die Mitte der achtziger Jahre für Fernnetze (WANs) entwickelt wurde

- Gateway
  Netzelement, welches unterschiedliche Netztechnologien verbindet.

- Ingress Router
  Border Router zur speziellen Bearbeitung von ankommenden IP-Paketen.

- Leased Line
  Telekommunikationsleitungen, welche für einen privaten Zweck gemietet werden.

- Link
  Verbindung zwischen zwei Netzelementen.

- Message
  Klasse des BSP-Protokolls, welche eine vollständige Nachricht beschreibt.

- Multicast
  Eine Telekommunikationstechnik, durch die ein Informationsfluss von einer Quelle an bestimmte potentielle Empfänger verbreitet werden kann.

- Object
  Klasse des BSP-Protokolls.

- Ordered Collection
  Klasse des BSP-Protokolls, welche eine geordnete Liste von Objects darstellt.

- Port
  Eine Art Briefkastennummer einer IP-Adresse aber auch die Bezeichnung eines Anschlusses in einem Netzelement.

- Premium Service
  DiffServ-Dienstklasse

- Provider
  Dienstanbieter im DiffServ-Umfeld.
- Queueing
  Verfahren zum Management von Warteschlange zur temporären Lagerung von PDU.
- Router
  Bezeichnung für ein Netzelement in einem verbindungslosen paketvermittelnden Netz.
- Selection
  Klasse des BSP-Protokolls, welche eine Auswahl von Objects darstellt.
- Service Broker
  Siehe Broker
- Swing
  GUI-Bibliothek von Java. Swing ist in Java 2 integriert.
- Token Bucket
  Token-Behälter, welcher regelmässig mit Token gefüllt wird und unregelmässig geleert werden kann. Dient zur Regulierung von Datenflüssen.
- Traffic Policing
  Mechanismus zur Überwachung von QoS-Parametern
- Traffic Shaping
  Mechanismus zur Durchsetzung von QoS-Parametern
- Verbindungslos
  Ein Netzzugriffsmodus, bei dem kein expliziter Verbindungsaufbau erfolgt. Die Pakete werden von einem Netzelement zum anderen gesendet, ohne dass der Absender eine Rückmeldung erhält, ob und wie die Pakete angekommen sind.
- Verbindungsorientiert
  Ein Netzzugriffsmodus, der bedeutet, dass Pakete nur über Verbindungen ausgetauscht werden, welche dem Netz bekannt sind. Vor einer Übermittlung ist deshalb oft ein Verbindungsaufbau und nach der Übermittlung ein Verbindungsabbau notwendig.
- Virtual Private Network
  Logischer Verbund von Endsystemen

Anhang B

Abkürzungen

- ATM
  Asynchronous **T**ransfer **M**ode: eine von der ITU für Fernnetze mit hohen Bitraten entwickelte Norm, die auf dem Prinzip des asynchronen Zeitmultiplexverfahrens basiert.
- BSP
  Broker **S**ignaling **P**rotocol (siehe Kapitel 2.3)
- CATI
  Charging and **A**ccounting **T**echnologies for the **I**nternet: Zusammenfassung der Schweizerischen Nationalfondprojekte CAPIV und MEDeB. CAPIV: Charging and Accounting Protocols in the Internet and in VPNs. MEDeB: Management, Evaluation, Demonstrators, and Business Models.
- CD
  Instanzen der Schicht 4 in der Broker Hierarchie (siehe Kapitel 2.1) werden **C**onfiguration **D**aemon genannt.
- CL
  Controlled **L**oad Service [RFC2212]
- CoS
  Class **o**f **S**ervice
- CSS
  Instanzen der Schicht 1 in der Broker Hierarchie werden **C**omposite **S**ervice **S**erver genannt.
- DiffServ
  DiffServ steht für **Diff**erentiated **Serv**ices und wird in Kapitel 1.2.7 erläutert.
- DS
  Abkürzung für das ToS-Feld des IP-Headers, wenn dieses für DiffServ verwendet wird.
- EBB
  External **B**andwidth **B**roker: eine Instanz der Schicht 2 in der Broker Hierarchie (siehe Kapitel 2.1).

- EDB
  External **D**elay **B**roker: Eine Instanz der Schicht 2 in der Broker Hierarchie.

- ESB
  Instanzen der Schicht 2 in der Broker Hierarchie werden **E**xternal **S**ervice **B**roker genannt.

- EVB
  External **V**PN **B**roker: Eine Instanz der Schicht 2 in der Broker Hierarchie.

- FlowSpec
  Flussspezifikation bei IntServ, welche Informationen über die vom Empfänger gewünschte Dienstqualität enthält.

- GS
  Garanteed **S**ervice [RFC2211]

- GUI
  Grafical **U**ser **I**nterface

- HTTP
  Hyper**t**ext **T**ransport **P**rotocol

- IBB
  Internal **B**andwidth **B**roker

- IntServ
  Intergrated **Serv**ices

- IP
  Internet **P**rotocol

- IPv4
  Internet **P**rotocol **V**ersion **4**

- IPv6
  Internet **P**rotocol **V**ersion **6**

- IPTV
  Proprietäres Verfahren von CISCO zur Übertragung von Audio- und Videodaten in Echtzeit. Die Übertragung dieser MPEG-codierten Daten erfolgt via IP-Multicast. Die Frame-Rate und damit die benötigte Bandbreite kann variiert werden.

- ISB
  Instanzen der Schicht 3 in der Broker Hierarchie werden **I**nternal **S**ervice **B**roker genannt.

- ISDN
  Integrated **S**ervices **D**igital **N**etwork

- ISP
  Internet **S**ervice **P**rovider

- LAN
  Local **A**rea **N**etwork: lokales Netz innerhalb eines Gebäudes oder Firmengeländes, bei welchem private Übertragungsmedien und Vermittlungsanlagen verwendet werden.

- MD5
  Message **D**igest **5**: Hashfunktionalität, welche zur Verschlüsselung verwendet werden kann.

- OSI
  Open **S**ystems **I**nterconnection. ISO-Standard 7498.

- PDU
  Protocol **D**ata **U**nit

- PGP
  Pretty **G**ood **P**rivacy: Verfahren zur elektronischen Verschlüsselung und Authentisierung.

- QoS
  Quality **o**f **S**ervice

- RSpec
  Dienstspezifikation einer FlowSpec bei IntServ

- RSVP
  Resource **Reserv**ation Setup **P**rotocol

- RTP
  Real **T**ime **P**rotocol

- SLA
  Service **L**evel **A**greement: Vertrag zwischen zwei ISPs oder einem ISP und dessen Kunde.

- SNMP
  Simple **N**etwork **M**anagement **P**rotocol gemäss [RFC1157]

- TCP
  Transport **C**ontrol **P**rotocol: Protokoll der OSI-Schicht 4 welches IP um einen Flusskontrollmechanismus erweitert.

- ToS
  Type **o**f **S**ervice

- TSpec
  Empfängerspezifikation einer FlowSpec bei IntServ

- UDP
  User **D**atagram **P**rotocol

- VoIP
  Voice **o**ver **IP**

- VPN
  Virtual **P**rivate **N**etwork

- WAN
  Wide **A**rea **N**etwork: Weitverkehrsnetzwerk, das grosse Entfernungen abdecken und über Vermittlungseinrichtungen von Netz- bzw. Telekommunikationsbetreibern zugänglich ist.

Literaturverzeichnis

[BB Signaling] M. Günter and T. Braun: Evaluation of Bandwidth Broker Signaling; Proceedings of the International Conference on Network Protocols ICNP'99, IEEE Computer Society, p.145-152, November 1999; ISBN 0-7695-0412-4; http://www.tik.ee.ethz.ch/~cati/paper/icnp99.pdf.

[CATI] T. Braun, B. Stiller et al.: The CATI Project: Charging and Accounting Technology for the Internet; Proceedings of Multimedia Applications, Services and Techniques ECMAST'99, LNCS 1629, p.281-296, Mai 1999, ISBN 3-540-66082-8; http://www.tik.ee.ethz.ch/~cati/.

[CATI BSP] T. Braun, M. Günter: CATI Internal Paper: Broker Signaling Protocol, März 1998; http://www.tik.ee.ethz.ch/~cati/deliv/CATI-IAM-DN-P-001-1.0.pdf.

[CA*net II] British Columbia Institute of Technology: CA*net II Differentiated Services: CANARIE ANA Bandwidth Broker High-Level Design; 1999; http://www.gait.bcit.ca/Projects/bandwidth_broker.

[DiffServ Layer] M. Günter: Präsentation zu Half-Time Results des CATI WP2: VPN Management in a Broker Hierarchy, 1999; http://www.iam.unibe.ch/~rvs/cati/Presentations/sitevisitShow.pdf.

[DiffServ Simulationen] Dobreff, Alexander: Comparison of simulation and real functionality for the mapping of Differentiad Services to ATM, Diplomarbeit der Philosophisch-naturwissenschaftlichen Fakultät der Universität Bern, 1999; http://www.iam.unibe.ch/~rvs/publications/dobreff.Diplom.pdf.

[I2Qbone] Rob Neilson, Jeff Wheeler et al.: Internet2 Qbone BB Advisory Council: A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment; Version 0.7, August 1999; http://www.merit.edu/internet/working.groups/i2-qbone-bb/doc/BB_Req7.pdf.

[IntServ - DiffServ] R. Balmer, F. Baumgartner, M. Günter and T. Braun: A Concept for RSVP over DiffServ; Juni 2000.

[IPnG] Braun, Torsten: IPnG: Neue Internet-Dienste und virtuelle Netze, dpunkt.verlag, 1999; ISBN 3-920993-98-5; http://www.dpunkt.de/produkte/ipng.html.

[Kyas98] Kyas, Othmar: ATM-Netzwerke, Technologie, Design, Betrieb. (DATACOM). 4., aktualis. u. erw. Aufl. 1998.

[NiBl98] K. Nichols, S. Blake: Differentiated Services Operational Model and Definitions, Internet Draft, work in progress, Februar 1998.

[QoS and VPN] M. Günter and T. Braun and I. Khalil: An Architecture for Managing QoS-enabled VPNs over the Internet: An Architecture for Managing QoS-enabled VPNs over the Internet Proceedings of the 24th Conference on Local Computer Networks LCN'99, IEEE Computer Society, p.122-131, Oktober 1999; ISBN 0-7695-0311-X; http://www.iam.unibe.ch/~rvs/publications/LCN99.pdf.

[Reservation Agents] O.Schelén, S. Pink: Resource Reservation Agents in the Internet, Lulea University of Technology, Sweden; 1998; http://www.cl.cam.ac.uk/Research/SRG/nossdav98/papers/nossdav98-052.ps.gz.

[RDG] Balmer, Roland: Integration von Integrated und Differentiated Services, Diplomarbeit der Philosophisch-naturwissenschaftlichen Fakultät der Universität Bern, 1999; http://www.iam.unibe.ch/~rvs/publications/diplomrb.pdf.gz.

[RFC793] Internet Transport Control Protocol, DARPA Internet Program Protocol Specification, RFC793, September 1981.

[RFC1157] J. Case, M. Fedor, M. Schoffstall, J. Davin: A Simple Network Management Protocol (SNMP), Mai 1990.

[RFC1190] C. Topolcic: Experimental Internet Stream Protocol, Version 2 (ST-2), RFC 1190, Oktober 1990.

[RFC1301] S. Armstrong, A. Freier, K. Marzullo, Multicast Transport Protocol, RFC1301, Februar 1992.

[RFC1633] R. Braden, D. Clark, S. Shenker: Integrated Services in the Internet Architecture: An Overwiew, RFC 1633, Juni 1994.

[RFC1819] L. Delgrossi, L. Berger: Internet Stream Protocol, Version 2 (ST2+), RFC 1819, November 1995.

[RFC1889] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson: RTP: A Transport Protocol for Real-Time Applications, RFC 1889, Januar 1996.

[RFC2205] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin: Resource ReSerVation Protocol (RSVP), RFC 2205, September 1997.

[RFC2211] J. Wroclawski: Specification of the Controlled & Load Network Element Service, RFC 2211, September 1997.

[RFC2212] S. Shenker, C. Patridge, R. Guerin: Specification of Guaranteed Quality of Service, RFC 2212, September 1997.

[RFC2474] F.Baker, K.Nichols, S. Blake, D. Black: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC2474, Dezember 1998.

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss: An Architecture for Differentiated Services, RFC 2475, Dezember 1998.

[RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: Hypertext Transfer Protocol – HTTP/1.1, Juni 1999.

[SLAT] G. Fankhauser: Service Level Agreement Traders (SLAT) and Cost-based Routing for Differentiated Services; TIK Report No. 59, Computer Engineering and Networks Laboratory, ETH Zürich; Januar 1999; http://www.tik.ee.ethz.ch/~gfa/paper/TR59.pdf.

[ShaperTest] M. Reardon: Traffic Shapers: IP in Cruise Control; 1998; http://www.data.com/issue/980921/traffic.html.

[Swing] The JFC Swing Tutorial: A Guide to Constructing GUIs; http://java.sun.com/docs/books/tutorial/uiswing/index.html.

[Tan] A.S.Tannenbaum: Computernetzwerke; (Prentice Hall) 3., revidierte Auflage 1998; ISBN 3-8272-9568-8.

[Two-bit] K. Nichols, V. Jacobson, L. Zhang: A Two-bit Differentiated Services Architecture for the Internet; RFC 2638; Juli 1999; http://irl.cs.ucla.edu/papers/twobit.pdf; http://sunsite.cnlab-switch.ch/ftp/doc/standard/rfc/26xx/2638.

[Two-Tier] F. Reichmeier, L. Ong et al.: A Two-Tier Resource Management Model for Differentiated Service Networks; Internet Draft; November 1998; http://irl.cs.ucla.edu/papers/2-tier-draft.ps.

[VPN Architecture] T. Braun, M. Günter et al.: Virtual Private Network Architecture; Technical Report; IAM-99-01; April 1999; http://www.iam.unibe.ch/~rvs/publications/TR-IAM-99-001.pdf.

Tabellenverzeichnis

Abbildungsverzeichnis