Integration von Integrated und Differentiated Services

Roland Balmer
RVS-Gruppe
IAM Uni Bern

13. November 1999

Abstract

Die Ideen von Integrated (IntServ) und Differentiated Services (DiffServ) sind in verschiedenen Zeitpunkten der Entwicklung des Internets entstanden. Bei dem älteren IntServ-Ansatz sollten Ressourcen für jeden einzelnen Datenstrom verwalten werden, während beim neueren DiffServ-Ansatz die Pakete einzelnen Transportklassen zugewiesen werden und somit eine bestimmte Transport-Qualität (QoS) erreicht werden kann. Da beide Ansätze ihre Vorteile haben, ist es sinnvoll die beiden Ideen miteinander zu verbinden und eine möglichst effiziente Zusammenarbeit zu ermöglichen. Diese Arbeit soll in die verschiedenen Konzepte einführen, diese erweitern und die Implementation einer Variante vorstellen.

Inhaltsverzeichnis

1 Einführung **4**

1.1 Überblick 4

1.2 Problemstellung 4

1.3 Danksagung 5

2 Integrated und Differentiated Services **6**

2.1 Best-Effort Services 7

2.2 Integrated Service (IntServ) 8

2.2.1 IntServ Modell 8

2.2.2 Ressource **R**eser**v**ation Setup **P**rotocol (RSVP) 11

2.3 Differentiated Services (DiffServ) 16

2.3.1 DiffServ Code Point 18

2.3.2 Traffic Conditioners 19

2.3.3 Verschieden DiffServ-Dienstgüten 20

2.3.4 Verschiedene Queueing-Verfahren für den Traffic Conditioner 21

2.4 Unterschied zwischen Integrated und Differentiated Services . . . 22

2.4.1 Informationen über die Datenströme 22

2.4.2 Aufsetzen einer Verbindung 22

2.4.3 Verschieden Dienstgüten 23

3 Verschiedene Konzepte für die Zusammenarbeit von Integrated und Differentiated Services **24**

3.1 IP-Tunneling 24

3.2 TOS-Field Aggregation 26
3.3 Bandwidth-Broker 27

4 Eigene Ansätze **29**
4.1 Erweiterung von RSVP 29
4.1.1 Beispiel einer Verbindung 30
4.2 Ansätze mit zwei kommerziellen Backbone-Routern 32
4.2.1 Verwendung der ResvErr-Nachricht 34
4.2.2 Verwenden der Resv-Nachricht 37
4.2.3 Verwenden der Path-Nachricht 39
4.2.4 Backbone-Router mit maximaler Reservierung für RSVP-Flows 41
4.2.5 Aggregation über den Linux-Router 43

5 Vergleiche der RSVP-DS Konzepte **46**
5.1 IP-Tunneling, TOS-Field-Aggregation und Bandwidth-Broker . . 46
5.2 Vergleich der eigene Ansätze 47
5.2.1 Verschiedene Probleme 47
5.2.2 Nötige Anpassungen 49
5.2.3 Abschliessender Vergleich 50

6 Tools **53**
6.1 **R**SVP **D**iffServ **G**ateway (RDG) 53
6.1.1 Definition der Schnittstelle 53
6.2 RSVP-Bandwidth-Broker 56
6.3 RSVPServer 57
6.3.1 Definition der Schnittstelle 58
6.4 RSVPTool 63
6.5 tc– Traffic-Control-Steuerung 64
6.5.1 Probleme mit RSVPd und tc 67

7 Realisierung **69**

8 Zusammenfassung und Ausblick **72**

8.1 Zusammenfassung 72

8.2 Ausblick 73

A Traffic-Control unter Linux **75**

B RSVP Definitionen **77**

B.1 Einige funktionale Definitionen 77

B.2 RSVP Header (Common Header) 78

B.3 RSVP-Objekt Formate 79

B.3.1 SESSION Objekt: Class = 1 81

B.3.2 RSVP_HOP Objekt: Class = 3 82

B.3.3 TIME_VALUES Objekt: Class = 5 82

B.3.4 ERROR_SPEC Objekt: Class = 6 83

B.3.5 STYLE Objekt: Class = 8 84

B.3.6 FLOWSPEC Objekt: Class = 9 85

B.3.7 FILTER_SPEC Objekt: Class = 10 87

B.3.8 SENDER_TEMPLATE Objekt: Class = 11 87

B.3.9 SENDER_TSPEC Objekt: Class = 12 88

B.3.10 ADSPEC Objekt: Class = 13 88

C Abkürzungen **90**

Literaturverzeichnis **91**

Kapitel 1

Einführung

1.1 Überblick

In einem ersten Teil soll ein Überblick über diese Arbeit und ein Einblick in deren Aufgabenstellung gegeben werden. Zudem werden kurz die Ideen, die hinter den Konzepten von Integrated (IntServ) und Differentiated Services (DiffServ) stehen, erläutert und auf Probleme bei der Zusammenarbeit eingegangen. Im folgendem Teil werden verschiedene Lösungen präsentiert, wobei in Kapitel 3 auf bereits bestehende Ansätze und in Kapitel 4 auf neue oder abgeleitete, an die Problemstellung angepasste, Ansätze eingegangen wird. In Kapitel 5 werden diese Lösungen miteinander verglichen. Im dritten Teil wird die Implementation einer Lösung und einige Tools die zu Testzwecken entworfen wurden vorgestellt. Im abschliessenden Kapitel 8 wird das Erreichte zusammengefasst und ein kleiner Ausblick in die Zukunft gegeben. Im Anhang finden sich dann noch einige Definition, die bei einigen Ansätzen gebraucht werden, sowie eine Zusammenfassung der verwendeten Literatur.

1.2 Problemstellung

Die ursprüngliche Ausschreibung der Arbeit sah wie folgt aus:

> *Aufgabe dieser Diplomarbeit ist die Identifikation und Bewertung von Integrationskonzepten der beiden Ansätze Differentiated und In-*

tegrated Services. Während der Integrated Services Ansatz für kleine, abgeschlossene Netze (z.B. corporate networks) geeignet ist, bietet sich der Differentiated Services Ansatz für den Einsatz in grösseren IP-Backbone-Netzen an. Die hierzu existierenden Konzepte sollen basierend auf der vorgenommenen Bewertung entweder verworfen oder weiterentwickelt werden. Ein geeignetes Konzept ist auszuwählen und in einem experimentellen Router-Netzwerk prototypisch zu implementieren.

Mit der Zeit wurden die Rahmenbedingungen der Arbeit angepasst, da verfügbare Hardware und Software in die endgültige Implementierung eingefügt werden musste und die Kompatibilität mit anderen in der Gruppe entwickelten Konzepten sichergestellt werden sollte. So wurden für den Backbone auf professionelle Router zurückgegriffen, die über eine DiffServ $\leftrightarrow$ IntServ Unterstützung verfügten sollten, in deren Betriebssystem aber keine Anpassungen vorgenommen werden konnte. Im IntServ Bereich wurde auf frei verfügbare Implementationen zurückgegriffen, die, da der Source-Code frei verfügbar war, abgeändert werden konnten. So kam es, dass sich das Problem auf die Entwicklung eines Konzeptes und die Integration der einzelnen Komponenten verlagerte.

1.3 Danksagung

Mein Dank gilt allen Personen der RVS-Gruppe. Im Besonderen gilt mein Dank dreien Personen die sehr wichtig für das entstehen dieser Arbeit waren. Zuerst möchte ich Prof. Dr. Torsten Braun danken, da ohne ihn diese Arbeit nie Zustande gekommen wäre. An zweiter Stelle gilt mein Dank Florian Baumgartner, der mir bei der Erarbeitung und Umsetzung der einzelnen Ansätze zur Seite stand. Last but not least geht mein Dank an Ibrahim Khalil, da ohne ihn die Integration der Backbone-Routers um einiges aufwendiger geworden wäre.

Kapitel 2

Integrated und Differentiated Services

In den Anfängen des Internet stand der reine Transport der Daten im Vordergrund. Die zu transportierenden Datenmengen waren relativ klein und es Bestand auch keine Notwendigkeit, dass dies in Echtzeit zu geschehen habe. Mit der Zeit wuchs das Verkehrsaufkommen, allerdings auch die zur Verfügungstehenden Ressourcen. Erst mit dem Boom des letzten Jahrzehntes und der Entwicklung von immer neueren Anwendungen kam man an einen Punkt, an dem spezielle Qualitätsanforderungen an das Internet gestellte wurde.
Da am Anfang das Internet hauptsächlich zum Informationsaustausch zwischen den Universitäten diente und auch von diesen unterhalten wurde, gab es kaum kommerzielle Interessen. Da aber in neuerer Zeit immer mehr Personen Zugriff auf das Internet haben und die Zugangsnetze immer häufiger durch private Betreiber zur Verfügung gestellt werden, sind die wirtschaftlichen Interessen gestiegen. Diese Internet Service Provider (ISP) haben den Wunsch ihre Leistungen vergütet zu bekommen. Da die Abrechnung für jeden Datenaustausch aufwendig ist, entsteht durch das Einführen von verschiedenen Dienstgüten die Möglichkeit einer einfachen Abrechnung für den Kunden. Wer mehr will, soll dafür mehr bezahlen.
Bevor wir nun auf die verschiedenen Ansätze Integrated und Differentiated Services eingehen, betrachten wir den *ist* Zustand.

2.1 Best-Effort Services

Beim bestehenden Service-Modell im Internet [Cla88] werden in den Routern alle Pakete mit einer FIFO-Strategie (First In First Out) weitergeleitet. Die Pakete werden auf den unteren Schichten im ISO-Modell gleich behandelt. Die einzigen Unterschiede entstehen nur durch verschieden Transportprotokolle und deren Strategie bei der Fehlererkennung und der Fehlerbehebung, die sich aber in höheren Schichten abspielen.
Da alle Pakete gleich behandelt werden, brauchen nur die beiden Endpunkte einer Verbindung die Informationen über den Datenstrom zu besitzen. Dazwischen genügt es die Empfängeradresse zu wissen, um die Daten dorthin weiterzuleiten. Dadurch kann auch eine gewisse Anonymität der Daten gewährleistet werden.
Mit der Entstehung von neuen Anwendung wurde auf der einen Seite das Verlangen nach Änderungen des bestehenden Modells laut, auf der anderen Seite wurden dagegen die folgenden Argumente gebraucht:

- Durch neue Entwicklung im Hardwarebereich wird die zur Verfügung stehende Bandbreite immer grösser sein, als die transportierte Datenmenge.
- Die Anwendungen sollen sich den gegebenen Strukturen anpassen und entstehende Engpässe ausgleichen können.
- Bei der Vergabe von Prioritäten ändert nichts am zu Grunde liegenden Modell. Es ist dadurch nicht sichergestellt, dass die Vergabe von Prioritäten den gewünschten Effekt ergibt. Man käme relativ schnell an einen Punkt, an dem sich der Verkehr wieder gegenseitig blockiert.

Mit der Zeit zeigte sich aber, dass sich diese Annahmen als falsch herausstellten. So musste die Aussage der unbegrenzten Bandbreite dahingehend revidiert werden, dass der Anstieg der zu transportierenden Daten grösser ist, als der Anstieg der zur Verfügung stehenden Bandbreite.
Die Aussage, dass sich die Anwendungen anpassen sollen musste verworfen werden, da immer mehr zeitkritische Anwendungen auf das Internet zugreifen. So

kann eine Verzögerung von 1 Sekunde bei einer Videokonferenz oder ein Bildausfall ähnlicher Länge sehr unangenehm sein. Zudem wurden die Erwartungen an solcher Anwendungen mit der Verbreitung des Internets immer grösser.
Und die letzte Aussage, dass Prioritäten nichts bringen, wurde durch die Entwicklung ausgeklügelter Systeme, wie sie in den zwei folgenden Kapitel gezeigt werden, widerlegt.

2.2 Integrated Service (IntServ)

IntServ war dazu gedacht, das bestehende Internet zu ergänzen und nicht zu ersetzen. Dies bedeutet, dass das bestehende Modell des Internets um neue Komponenten und Definitionen sinnvoll ergänzt werden musste. Dazu mussten auch einige Komponenten neu entwickelt werden.
Im nächsten Abschnitt wird gezeigt wie das bestehende Modell zu ergänzen ist. Im zweiten Abschnitt wird mit dem Ressource **Reserv**ation Setup **P**rotocol (RSVP) die aktuelle Implementierung des IntServ-Ansatzes kurz vorgestellt.

2.2.1 IntServ Modell

Das IntServ Modell wurde in der ersten Hälfte der 90er Jahren entwickelt und wurde 1994 in [BCS94] von R. Braden e.a. beschrieben. Dort wurde verlangt, das bestehende Service-Modell zu ergänzen, da es nicht sinnvoll wäre, eine eigenständige Struktur für bestimmte Transportarten zu erzeugen und verwalten. Braden stellte zwei neue Transportklassen für Echtzeit Anwendungen vor. Dabei soll die eine gewisse Transportqualität garantieren (Guaranteed Service [SPG97]) und die andere die bestmögliche Qualität unter Ausnutzung der vorhanden Ressourcen ermöglichen (Controlled Load Service [Wro97]). Diese Dienste sollten so beschaffen sein, dass der End-User sich auf eine zugesagte Transportqualität verlassen kann. Um dies zu ermöglichen, muss die vorhandene Bandbreite möglichst kontrolliert verwaltet werden. Um eine gewisse Dienstgüte (QoS) sicherzustellen, muss die Möglichkeit bestehen Ressourcen zu reservieren. So müssen auf der einen Seite Möglichkeit zur Reservierung von Ressourcen zur

Verfügung gestellt werden, auf der anderen Seite impliziert dies die Schaffung von Schutzmechanismen um den Zugriff auf Ressourcen zu regeln.
Um eine bestehende Qualität zu garantieren müssen alle Komponenten (Router), die Datenpakete weiterleiten, fähig sein diese Dienstgüte zu unterstützen. Da im IntServ Modell einzelne Datenströme verwaltet werden, müssen somit alle Router in der Lage sein, die Zustände der Datenströme zu verwalten. Diese dazu nötige Kennung (FlowID) sollte aus eindeutigen Daten bestehen, die einen Datenstrom genau definieren. Aus diesem Grunde verwendet man ein Tupel aus der Adresse und Portnummer des Senders und des Empfängers. Dies widerspricht dem ursprünglichen Konzept des Internets, in dem nur die Endgeräte Informationen über die Datenströme haben. Da das Internet sehr dynamisch ist, müssen diese Zustände (Flow-States) möglichst flexibel verwaltet werden (Soft-States).
Damit die Reservierung von Bandbreite sinnvoll genutzt werden kann, benötigt man in den Routern Komponenten, die die einzelnen Datenströme, bzw. deren Pakete unterscheiden können. Da nicht alle gleichzeitig Anrecht auf privilegierten Datentransport haben können, braucht man auserdem eine verbesserte Transport- und Zugangskontrolle.
Da die bereits bestehende Infrastruktur verwendet werden soll, ist es sinnvoll bestehende Protokolle (TCP, UDP) auch bei Echtzeit Datenverkehr weiter zu verwenden. Damit ist es eher möglich ein *homogenes* Internet beizubehalten. Ein weiterer Vorteil ist, dass die Dienstgüten besser zugesichert werden können, weil unvorhersehbare Verzögerungen auf Teilabschnitten oder an Schnittstellen entstehen.
Abbildung 2.1 zeigt das Schema eines Routers, der an den IntServ-Ansatz angepasst wurde.
Dabei erkennt man, dass der Router in zwei Teile unterteilt wird. Im oberen Teil sind die adminstrativen Komponenten, im unteren die transportierende Schicht abgebildet. Die einzelnen Komponenten sollten nun folgende Eigenschaften haben, die in heutigen Implemationen auch vorhanden sind:

- Der **Routing Agent** wird nicht angepasst, da er die Auswahl der be-

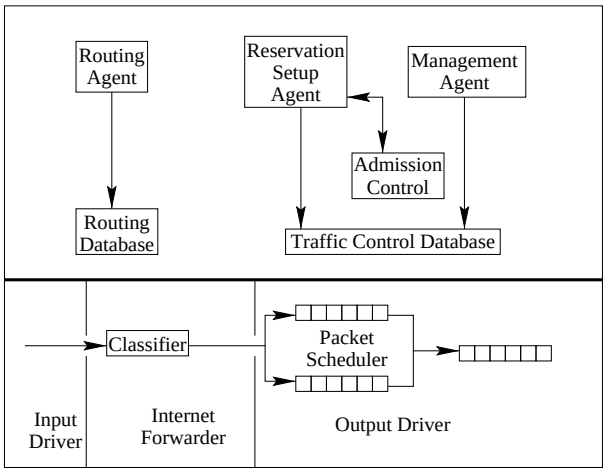

Abbildung 2.1: Modell eines Routers, der Ressourcen-Reservierung unterstützt.

sten/einzigen Route der Pakete wie bisher machen kann.

- Die **Routing Database** muss ebenfalls nicht erneuert werden.
- Die **Admission Control** ist ein wichtiger und neuer Teil im Router. Sie muss entscheiden, ob ein neuer Datenstrom zugelassen werden kann oder nicht. Sie stellt sicher, dass bestehende Verbindungen nicht gestört werden.
- Der **Reservation Setup Agent** ist eine neue Komponente. Er ist für die Umsetzung eines Reservations-Wunsches zuständig. Mit Hilfe der Admission Control setzt er einen neuen Datenstrom auf, wenn dies möglich ist. RSVP sieht vor, dass die Reservation Setup Agenten miteinander kommunizieren können.
- Der **Management Agent** ist eine bereits bestehende Komponente, die dafür sorgt, dass bestehende Verbindungen korrekt verwaltet werden, so dass keine Schwierigkeiten während des Betriebs auftreten.
- Die **Traffic Control Database** enthält alle Daten, die gebraucht werden, um die verschiedenen Datenströme zu verwalten. Sie wird durch den Reservation Setup Agent und den Management Agent aktualisiert.

- Der **Paket Scheduler** ist dafür zuständig, dass alle Datenströme ihre zugesicherte Dienstgüte bekommen. Er wird mit Hilfe der Daten der Traffic Control Database konfiguriert. Im Normalfall stellt er Warteschlangen (Queues) zur Verfügung, die die einzelnen Datenströme im Router puffern. Danach holt er aus den Queues die entsprechenden Pakete, um sie an das ausgehende Interface weiterzuleiten. Der Paket Scheduler muss in der Lage sein einen Überlauf seiner Warteschlangen zu verhindern, so dass überzählige Pakete kontrolliert weggeworfen werden können.

- Der **Classifier** weisst jedem Paket eine bestimmte Warteschlange zu, die der Paket Scheduler erzeugt hat. Er verwendet dafür sowohl die Routing-, als auch die Traffic Control Database. Der Classifier muss dahingehend erweitert werden, dass er zur Entscheidungsfindung nicht nur die Empfängeradresse, sondern auch andere Informationen des IP-Header heranziehen kann. Zur Unterscheidung eines Datenstrom reicht die oben vorgestellte FlowID.

Wie oben erwähnt werden verschieden neue Komponenten und Protokolle benötigt. So wird im nächsten Abschnitt das RSVP-Protokoll vorgestellt, das die verbreitetste Umsetzung des IntServ-Ansatzes ist. Unter Linux wird dafür der offizieller RSVP-Daemon des ISI (aktuelle Version 4.2a4), ein Linuxkernel, der über TrafficControlMechanismen verfügt (aktuelle Version $>$ 2.2.12) verwendet. Mehr dazu im Anhang A.

2.2.2 Ressource Reservation Setup Protocol (RSVP)

In diesem Abschnitt werden die Grundzüge von RSVP erklärt. Für eine ausführliche Beschreibung wird auf [BZB+97] verwiesen. Auf die TrafficControl Komponenten wird im Kapitel 6.5 näher eingegangen. Um einige Eigenschaften des RSVP-Protokolls zu verstehen, muss eine Eigenschaft des RSVP-Daemons erklärt werden:

- **Soft States:** Die Daten für das Umsetzen der Dienstgüte werden flexibel verwaltet und haben nur eine gewisse Lebenszeit. Dadurch muss nicht

explizit eine Beendigung der Reservierung durchgeführt werden. Dies entspricht der Idee des IntServ-Modells.

In diesem Soft-State werden alle Daten, die für die Reservierung gebraucht werden gespeichert. Dies umfasst die vorhergehende und nachfolgende RSVP-fähige Maschine, die Beschreibung des Datenstromes und einer eventuell erzeugten Reservierung.

Beispiel eines Reservierungsaufbaues

Um die Idee hinter RSVP zu verdeutlichen, betrachten wir das folgende Beispiel. In Abb. 2.2 sehen wir den Aufbau einer Reservierung mit anschliessender Datenübermittlung und darauffolgender Freigabe der Ressourcen.

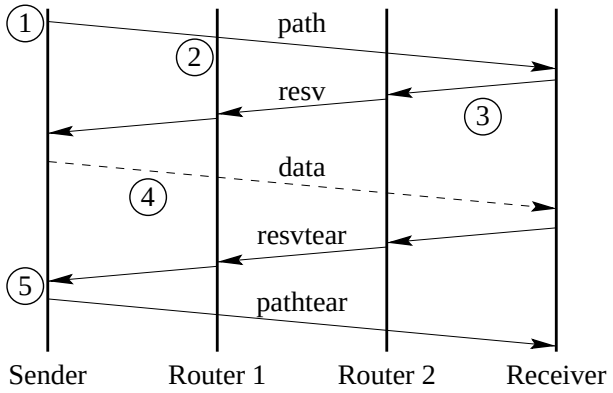

Abbildung 2.2: Flussdiagramm eines normalen Auf- und Abbau einer Reservierung.

Anhand dieses Beispieles können wir einige Eigenheiten zeigen:

1. Die **Path**-Nachricht wird gebraucht, um die Route durch das Netz festzustellen, die die Datenpakete nehmen werden. Die PathNachricht wird dabei direkt an den Empfänger der Daten geschickt. Ausserdem enthält sie Informationen über die Eigenschaften der zu übertragenden Daten ($\rightarrow$ Sender Tspec).

2. Da die Path-Nachricht direkt an den Empfänger geschickt wird, muss sie speziel gekennzeichnet werden. Dies erfolgt durch das Setzen des Router-

Alert Flags. Damit reagiert jeder Router auf die Path-Nachricht und initialisiert seine Soft-States. Zudem aktualisiert er das Objekt in der Path-Nachricht, das den letzten Router/Host enthält. Sobald die Path-Nachricht beim Empfänger angekommen ist, weiss jeder Router auf der Strecke, welcher Router, der RSVP unterstützt, vor ihm liegt.

Da die so erzeugten Soft-States nur eine beschränkte Lebenszeit haben, muss die Path-Nachricht regelmässig wiederholt werden. Um nicht das Netz mit Path- und Resv-Nachrichten zu überlasten, werden diese standartmässig alle 30 Sekunden wiederholt. Um nicht wegen einer einzigen verlorenen Path-Nachricht die Reservierung zu beenden, entspricht die Lebenszeit der Soft-States dem dreifachen dieses Zeitintervalles.

Durch die Wiederholung wird auch auf eine Routenänderung der Daten reagiert, da der Transport der Path-Nachricht und der Daten auf der gleichen Route erfolgt.

3. Mit der **Resv**-Nachricht wird nun die Reservierung aufgesetzt. Dabei wird diese im Gegensatz zu Path immer an die nächste Maschine mit RSVP-Unterstützung weitergeleitet. In jedem RSVP-fähigen Router/Host wird nun kontrolliert, ob eine Reservierung möglich ist oder nicht. Wird sie akzeptiert, so wird die Resv-Nachricht weitergeleitet, sonst aber eine Fehlermeldung erzeugt und zum Initiant der Reservierung zurück geschickt.

   Dem Initianten der Reservierung bleibt dabei überlassen, wieviel Ressourcen er reservieren will. Normalerweise werden aber Ressourcen den Eigenschaften entsprechend reserviert, die vom Sender übermittelt wurden (Sender Tspec $\Rightarrow$ flowspec).

   Auch die Resv-Nachricht muss periodisch wiederholt werden.

4. Sobald die Resv-Nachricht den Sender erreicht hat, kann dieser die Daten über diese **unidirektionale** Reservierung senden. Sendet der Sender mehr Daten als reserviert wurde, so werden diese einfach als Best-Effort weitergeleitet.

5. Wenn die Datenübertragung beendet ist, gibt es mehrere Möglichkeiten die Reservierung zu beenden. Die einfachste ist, das Senden der Path-, bzw. der Resv-Nachricht einzustellen. Sobald die Soft-States nicht mehr erneuert werden, werden diese gelöscht. Weiter kann man die Reservierung auch explizit löschen, indem man durch die ResvTear-Nachricht die reservierten Ressourcen freigibt oder mit der PathTear-Nachricht die eingerichtete Soft-States entfernt.

Aggregration von Reservierungen

Eine weitere Eigenschaft von RSVP ist die Multicastunterstützung. D.h. Daten können an mehrere Empfänger gleichzeitig gesendet werden. Betrachten wir dazu Abb. 2.3.

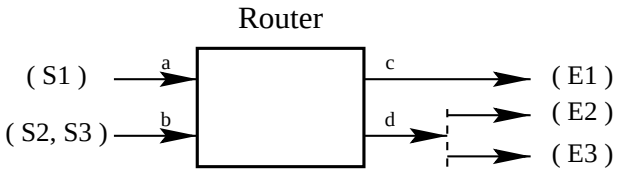

Abbildung 2.3: Schema eines Routers mit 4 Interfaces. Die Sender (S1, S2, S3) senden ihre Daten zu den Empfängern (E1, E2, E3).

Es ist ein Router mit vier Interfaces dargestellt. Auf der linken Seite des Interfaces befinden sich die Sender (S1, S2 und S3) und auf der rechten Seite die Empfänger (E1, E2 und E3). Werden nun vom Sender S1 Pakete an die Empfänger E1 und E2 gesendet, so wird der Datenstrom im Router auf zwei verschiedenen Interfaces weitergeleitet. Für die Reservierung gilt das genaue Gegenteil. Da ja der Empfänger die Reservierung verlangt, bestimmt dieser auch wieviel er reservieren will. Im Router kommen die beiden Anfragen der Empfänger zusammen. Würden die Reservierungen einfach weitergeleitet und z.B. einer der End-User nicht die volle Bandbreite möchte, so würde die reservierte Bandbreite regelmässig erhöht, bzw. erniedrigt.
Betrachten wir dazu folgendes Beispiel: S1 sendet 2 Einheiten an E1 und E2. E2 reserviert aber nur 1 Einheit, während E1 2 Einheiten reserviert. Die Reser-

vierungsanfragen treffen im Router aufeinander. Werden dies einfach weitergeleitet, so wird einmal 1 Einheit reserviert, danach wieder 2 Einheiten, usw.
Um dies zu verhindern, werden an einem Kreuzungspunkt mehrerer Reservierungen einer Multicast-Verbindung zu einer Reservierung zusammengelegt, so dass die weitergeleitete Reservierungsanfrage genau so gross ist, wie die grösste erhaltenen Anfrage.
RSVP unterstützt aber auch die Möglichkeit, Reservierungen von mehreren Sendern zu mehreren Empfängern zusammenzufassen. Typische Anwendungen wären Telefon-, oder Videokonferenzen. Eine Möglichkeit die Daten zu transportieren, wäre für jede Verbindung zwischen Sender und Empfänger eine eigene Reservierung aufzusetzen. Dies würde aber ein relativ grossen Aufwand von Ressourcen bedeuten. In RSVP gibt es deshalb die Möglichkeit, verschieden Reservierungen zu einer einzigen zusammenzufassen. Der End-User hat die Möglichkeit zu bestimmen, wie dies zu geschehen hat. Dafür werden drei verschiedene Styles definiert, deren Zusammenhang in der Tabelle 2.1 gezeigt wird.

Sender	Reservierung	
Auswahl	Getrennt	Geteilt
Explizit	Fixed-Filter (FF)	Shared-Explicit (SE)
Wildcard	-	Wildcard-Filter (WF)

Tabelle 2.1: Zusammenhang der verschieden Reservierungsstyles.

Wie man aus der Tabelle entnehmen kann, gibt es keinen Style für eine getrennte Reservierung bei einer Wildcard-Auswahl der Sender, was ja auch keinen Sinn machen würde. Ein Beispiel, wie eine solche Zusammenlegung von Reservierungen erfolgt ist in der Tabelle 2.2 erläutert.
In diesem Beispiel bestehen folgende Verbindungen:

- S1 $\rightarrow$ E1, E2
- S2 $\rightarrow$ E2, E3
- S3 $\rightarrow$ E2

	OUT	Resv	IN	
(a)	SE(S1 {3E})	(S1, S2) {E}	SE((S1, S2) {E})	(c)
(b)	SE((S2, S3) {3E})	(S1, S2, S3) {3E}	SE((S1, S3) {3E}) SE(S2 {2E})	(d)

Tabelle 2.2: Zusammenlegen bei Shared-Explicit.

Betrachtet man die erzeugte Reservierung des Interface (d), so erkennt man, dass die verschiedenen Sender zusammengefasst werden und die grösste Reservierungsanfrage übernommen wird. D.h. für die Daten von den Sender S1, S2, S3 sind im Interface (d) 3 Einheiten reserviert. Betrachtet man dazu die Anfragen, sieht man relativ schnell, dass eventuell ein Problem daraus entstehen kann, wenn alle Sender gleichzeitig voll senden. Es gilt deshalb anzumerken, dass eine Teilung der Reservierung bei verschiedenen Sendern nur sinnvoll ist, wenn nicht alle gleichzeitig senden. Daraus folgt, dass die Reservierungsstyles WF und SE nur bei Audioübertragungen sinnvoll sind und bei Videoübertragungen FF verwendet werden muss.

2.3 Differentiated Services (DiffServ)

Mit dem IntServ-Ansatz konnte man eine Reservierung für einzelne Datenströme erreichen, doch befürchtete man Schwierigkeiten in grossen Netzen. Durch die Reservierung jedes einzelnen Datenflusses fallen für die Verwaltung viele Daten an, was speziell im Backbone-Bereich zu problemen führt.. Also wurden neue Überlegungen angestellt um ein geeigneteres Modell zu entwickeln. Die Architektur von DiffServ wird in [BBC+98] genauer beschrieben. Sie basiert auf der einfachen Idee, dass Datenpakete, die in ein Netz gelangen an deren Grenze klassifiziert und gegebenenfalls weitergeleitet werden. Im Gegensatz zu RSVP, muss der Netzbetreiber dafür sorgen, dass je nach Güteklasse genug Ressourcen zur Verfügung stehen, da der Benutzer keine Reservierungsbestätigung erhält. Die Umsetzung einer gewünschten QoS bleibt ihm überlassen, wobei er durch Verträge mit anderen Netzbetreibern an gewisse Rahmenbedingungen

gebunden ist. Diese Service Level Aggrements (SLA) können angepasst werden. Jedoch muss der ISP diese SLA auch einhalten können (network provisioning). Um dies und die DiffServ-Architektur zu erklären, betrachten wir das Beispiel von Abb. 2.4.

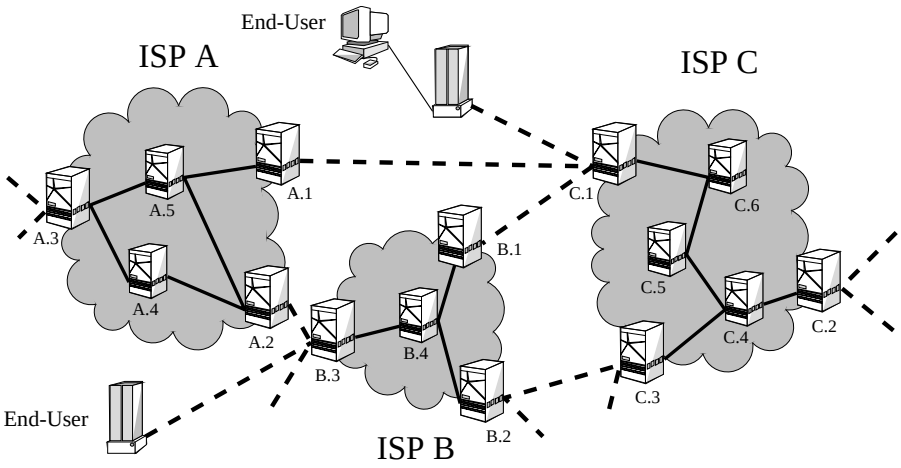

Abbildung 2.4: Aufbau eines Netzes mit drei verschiedenen ISP's und zwei End-Usern.

In diesem Beispiel sind drei Netzbetreiber (ISP) dargestellt. Es darf angenommen werden, dass innerhalb dieser Netze die unterstützten Dienstgüten homogen ist. Wäre dies nicht der Fall könnte man das Netzwerk weiter unterteilen. Um eine Verbindung zwischen den beiden End-Usern aufzubauen, müssen die Netze von mindestens zwei ISPs benutzt werden. Verfolgt man solch einen Pfad, so trifft man auf mehrere Router. Betrachten wir die aufgabe diese Router.

Boundary- und Interiorrouter

In der DiffServ-Architektur existieren zwei Arten von Routern. Dies sind Boundary- und Interiorrouter (z.B. B.4). Beide Routerarten müssen die vom Netz definierten Dienstgüten voll unterstützen. Zudem sollten sie in der Lage sein Pakete neu zu markieren. Zusätzlich steht der Boundaryrouter (z.B. B.1, B.3) an der Grenze des Netzwerkes und hat Verbindungen zu anderen Netzen. Dadurch bildet jeder Boundaryrouter einen Zugangsknoten zum Netz und hat somit zwei verschieden Funktionen. Für Pakete, die in das Netz gelangen oder

aus dem Netz gehen, muss geprüft werden, ob diese den getroffenen Vereinbarungen (SLA) entsprechen.
Ein einzelnes oder auch mehrere Netze zusammen können eine DiffServ-Region bilden, in der die unterstützten Dienstgüten gleich definiert werden. Befindet sich ein Boundaryrouter an der Grenze einer solchen Region, so muss er auch sicherstellen, dass die unterschiedlichen Definitionen, gemäss SLA, aufeinander abgebildet werden können.
Wir haben nun gesehen, was für Eigenschaften ein DiffServ-Netzwerk und seine Hardwarekomponenten haben müssen. Wie diese nun eingesetzt werden, sehen wir im nächsten Abschnitt

2.3.1 DiffServ Code Point

Bei DiffServ wird jedes einzelne Paket markiert. Um zu sehen wie und wo dies geschieht betrachten wir Abb. 2.5. Dort sind die IP-Header für die IP-Version 4 und 6 abgebildet.

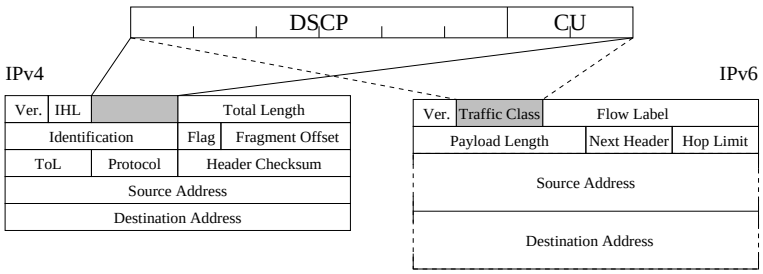

Abbildung 2.5: Die Plazierung des DiffServ-Fields im IP-Header und dessen Einteilung.

Man erkennt, dass im IPv4 Header das alte TOS-Field wiederverwendet wird, während beim IPv6, das dafür vorgesehene Traffic-Class-Feld gebraucht wird. In diesem Byte werden 6 Bit verwendet, um den DiffServ-Code-Point (DSCP) zu definieren. So kann die verlangte Dienstgüte für in jedes einzelne Paket eingetragen werden. Die übrigen 2 Bits werden zur Zeit nicht gebraucht.
Die definierten DSCP's sollten innerhalb einer DiffServ-Region homogen sein, weil es sonst zu Problemen bei der Einhaltung der Dienstgüte kommen kann.
Um ein Paket einzustufen, erlaubt DiffServ zwei verschiedene Möglichkeiten.

Die erste ist der Behavior Aggregate (BA) Classifier, der nur auf dieses DSCP zugreift. Die zweite Möglichkeit ist der Multi-Field (MF) Classifier, der auch andere Felder des IP-Headers verwendet wie z.B. Quell- und Zieladresse, Portnummern, usw.

Traffic Profiles

Für jeden DSCP wird ein Profil definiert. Eine solche Definition könnte etwa wie folgt aussehen:

- DSCP=X, use token-bucket r, b.

Dieses Profil bedeutet, dass alle Pakete mit diesem DSCP zusammen nur eine Token-Rate von r und einen maximalen Burst von b haben dürfen. Dementsprechend sind alle Pakete, die ausserhalb dieser Grenze liegen, auch ausserhalb des Profils (Out-of-profile). Alle Pakete, die innerhalb der Grenzen sind (in-profile), müssen, der Dienstgüte entsprechend, weitergeleitet werden. Out-of-profile Paket werden je nach verfügbarer Bandbreite und Taktik weggeworfen oder z.B. bei mehrstufigen Verfahren neu klassifiziert.

2.3.2 Traffic Conditioners

Der Traffic Conditioner ist verantwortlich, dass der ausgehende Datenstrom die erlaubten Parameter nicht überschreitet. Um dies zu erreichen besitzt er verschiedene Komponenten, wie sie in Abb. 2.6 gezeigt werden.

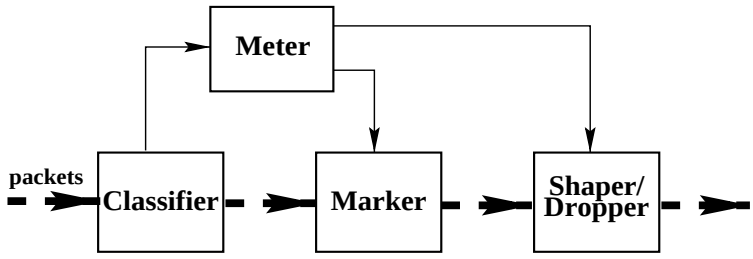

Abbildung 2.6: Einzelne Elemente eines Traffic Conditioners und wie sie auf wen wirken.

Die einzelnen Komponenten haben die folgenden Eigenschaften:

- Der **Classifier** muss mit Hilfe des DSCP die eingehenden Pakete klassifizieren. Er informiert den Meter über die Menge und Eigenschaften der ankommenden Pakete, während er diese an die entsprechende Warteschlange weiterreicht.
- Der **Meter** vergleicht die wahre Grösse des Datenstroms mit der im SLA vereinbarten. Aufgrund seiner Resultate werden die anderen Komponenten entsprechend gesteuert.
- Der **Marker** muss in den Paketen den DSCP entsprechend den Angaben des Meters neu (classification) oder umsetzen (reclassification).
- Der **Shaper** dient dazu den Datenstrom zu glätten. Er verfügt über eine endliche Warteschlange um Pakete zwischenzuspeichern.
- Der **Dropper** ist eine Art zweite Stufe des Shapers. Ist es für diesen nicht mehr möglich die Pakete in seiner Warteschlange unterzubringen, so müssen die entsprechenden Pakete sinnvoll weggeworfen werden, was die Aufgabe des Droppers ist.

2.3.3 Verschieden DiffServ-Dienstgüten

Mit der Entwicklung des DiffServ-Ansatzes sind immer mehr neue oder abgeänderte Idee von verschiedenen Dienstgüten endstanden. Da bis jetzt nur die Architektur einen stabilen Status hat und deren Umsetzung ein Problem der Netzbeteiber ist, werden nur die folgenden Ansätze kurz vorgestellt.

- Beim **Premium Service** wird eine gewünschte Bandbreite zugesichert. Bei unregelmässigen Übertragungsraten ist zum einen mehr Bandbreite reserviert als gebraucht wird, zum andernen aber werden Übertragungsspitzen nicht weitergegeben. Dies entspricht einer Mietleitung.
- Der **Assured Service** ist für unregelmässigen Datenverkehr flexibler geeignet. Er garantiert aber nicht einen Transport aller Pakete.
- Beim **Assured Forwarding** werden die Pakte in N unabhängige AF-Klassen verteilt. In jeder Klasse kann ein Paket verschieden Wegwerf-

prioritäten M haben. Zur Zeit wird N=4 und M=3 gebraucht. Bei der Bearbeitung von verschieden AF-Klassen, sollten Pakete mit einer kleineren Priorität weniger Bandbreite erhalten, als höher dotierte Pakete in anderen Klassen.

Auf die Vorstellung weiterer Vorschläge wird verzichtet, da diese zum Teil lediglich Verfeinerungen darstellen und zum Anderen den Rahmen dieser Arbeit sprengen würden.

2.3.4 Verschiedene Queueing-Verfahren für den Traffic Conditioner

Um das Weiterleiten der Pakete sicherzustellen und somit die Dienstgüte zu garantieren, gibt es mehrere Varianten. Da auch hier immer wieder neue Ideen entwickelt werden, werden hier nur einige wenige kurz vorgestellt.

- Die **Absolute Priority Queueing** ist ein unfäres Verfahren. Es stellt für verschiedene Prioritäten verschiedene Queues zur Verfügung. Kann ein Paket gesendet werden, wird zuerst die Queue mit der höchsten Priorität geleert. Auf diese Art können Flüsse mit niederiger Priorität ganz ausgebremst werden.

- Bei **Weighted Fair Queueing (WFQ)** wird für jede Traffic-Klasse eine eigene Queue verwaltet. Aus den einzelnen Queue's werden danach mit Round-Robin immer wieder Pakete entnommen. Durch einen Gewichtungsfaktor können einzelne Queues eine höhere Bandbreite erhalten.

- Mit **Class Based Queueing (CBQ)** [FJ95] kann die Traffic Control einer Baumstruktur ähnlich verwaltet werden. Dabei wird auf jeder Stufe die zur Verfügung stehende Bandbreite aufgeteilt. Häufig wird beim Auslesen der Queues ein ähnlicher Scheduler verwendet wie beim WFQ.

- Bei **Random Early Discard (RED)** [FJ93] wird nur eine Queue verwendet. Damit diese nicht überläuft, werden schon frühzeitig Pakete verworfen. Dies geschieht mit einer zunehmenden Wahrscheinlichkeit, je vol-

ler die Queue ist. Zusammen mit der TCP-Flusskontrolle sollen so Stausituation vermieden werden.

- Der **RED with In and Out (RIO)** [CW97] ist eine Erweiterung vom RED. Anstatt alle Pakete gleich zu behandelen, werden inprofile Pakete mit einer kleineren Wahrscheinlichkeit weggeworfen, als out-of-profile Pakete. Somit wird bei congestion die Queue eher mit inprofile gefüllt. Da nur eine Queue verwendet wird, wird hier auch die Reihenfolge der Pakete beibehalten.

2.4 Unterschied zwischen Integrated und Differentiated Services

2.4.1 Informationen über die Datenströme

Die Informationen über Datenströme werden sehr unterschiedlich gehandhabt. Bei IntServ werden Informationen über jeden einzelnen Datenfluss benötigt und gespeichert. Die Pakete werden anhand der Adressen und Portnummern des Empfängers und des Senders (flowspec) einem Datenströmen zugeordnet.
Bei DiffServ hingegen werden nicht einzelne Datenflüsse verwaltet, sondern jedes Datenpaket einzeln behandelt. Es enthält alle Informationen, die zu dessen Einteilung in die entsprechende Güteklasse führen. Im Normalfall ist dies das DSCP-Feld, es können aber auch noch weitere Informationen hinzugezogen werden.
Deswegen benötigt der DiffServ- Ansatz bedeutend weniger Speicher und Rechenleistung in den einzelnen Netzkomponenten.

2.4.2 Aufsetzen einer Verbindung

Bei DiffServ ist kein Aufsetzen der Verbindung nötig, da nur der SLA eingehalten werden muss.
Bei RSVP ist der Aufwand bedeutend grösser, da eine Reservierung aufgesetzt und regelmässig aktualisiert werden muss. Durch das Verfahren entsteht eine

Verzögerung bis die Reservierung zustande kommt. Sie ist also für kurze Verbindungen nicht geeignet.
Muss eine Reservierung angepasst werden, so kann bei RSVP einfach die bestehende Reservierug ergänzt werden.
Im DiffServ-Fall geschieht dies, indem mehr Pakete markiert werden. Dies geht aber nur solange der SLA nicht verletzt wird. Geht eine Reservierung über den SLA hinaus, muss ein neuer SLA ausgehandelt werden.

2.4.3 Verschieden Dienstgüten

Eigentlich werden in beiden Ansätzen ähnliche Dienstgüten verwendet, da sie aus der selben Idee entstanden sind. Ihr Zusammenhang sieht wie folgt aus:

IntServ		DiffServ
Best-Effort	$\leftrightarrow$	Best-Effort
Controlled-Load	$\leftrightarrow$	Assured
Guaranteed	$\leftrightarrow$	Premium

Die anderen Dienstgüten aus dem DiffServ-Ansatz werden hier nicht weiter betrachtet.

Kapitel 3

Verschiedene Konzepte für die Zusammenarbeit von Integrated und Differentiated Services

Sobald zwei verschiedene Arten von Services zusammenarbeiten müssen, entstehen an den Schnittstellen Probleme. In den zwei nächsten Kapiteln werden einige verschiedene Strategien der Zusammenarbeit gezeigt. Dabei handelt es sich bei den ersten Ansätzen um aktuelle Vorschläge aus Internet-Drafts der IETF und bei den weiteren um eigene Vorschläge, die auf der zur Verfügung stehenden Hardware basieren. In allem Fällen wird davon ausgegangen, dass in den Zugangs-Netzwerken das RSVP-Protokoll verwendet wird und erst in Backbonebereich auf DiffServ-Mechanismen zurückgegriffen wird.

3.1 IP-Tunneling

Wenn man verschieden RSVP-Netze untereinander vernetzen will und genau definierte Grenzen hat, so kann man eine Aggregration von mehreren Datenströmen erreichen, indem man einen IP-Tunnel [RG97] verwendet. Betrachten

wir dazu Abb. 3.1.

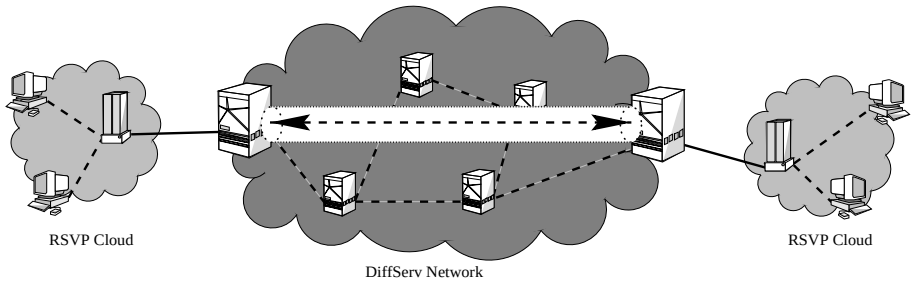

Abbildung 3.1: Zwischen den zwei Routern an der Grenze des DiffServ-Netzes ist ein Tunnel aufgebaut.

In den Zugangs-Netzwerken wird dabei ganz normal RSVP verwendet. Zwischen jedem RSVP-Bereich werden zwei IP-Tunnels etabliert, damit Controlled-Load und Guaranteed Service getrennt transportiert werden können.
Beim Eintritt in den Tunnel werden die einzelnen Pakete in andere IP-Pakete gepackt, die vom Anfangsspunkt (Ingress-Router) zum Endpunkt (Egress-Router) des Tunnels gesendet werden. Auf normale Datenpakete hat dies, ausser einer längeren Verarbeitungszeit, keinen Einfluss. Für die gekapselten RSVP-Path-Nachrichten treten zusätzliche Probleme auf. Da die Path-Nachricht unter anderem auch Transitzeiten ermittelt, dies aber bei einer gekapselten Nachrichten nicht aktualisiert werden kann, muss bei jeder Path-Nachricht im Egress-Router das entsprechende Objekt (Ad-spec) aktualisiert werden.
Bei der Reservierung von Bandbreite muss sichergestellt werden, dass die neu hinzugefügte Reservierung nicht die Kapazität des Tunnels übersteigt. Eine weitere Möglichkeit wäre, die Dimensionen des Tunnels anzupassen, was aber nicht immer möglich sein wird. Die selbe Taktik wird z.B. in einem VPN-Netz verwendet, wo die Pakete zusätzlich verschlüsselt werden können.
Wenn ein Datenstrom mehr Daten sendet, als er reserviert hat, so sollten diese nicht über den Tunnel transportiert, sondern gemäss RSVP-Spezifikation als Best-Effort neben dem Tunnel transportiert werden.
Beim IP-Tunneling ist im Backbonebereich sowohl der IntServ-, als auch der DiffServ-Ansatz vorstellbar. Bei einem RSVP-Backbonenetz würden für die aggregierten Datenströme nur noch ein einzelner Soft-State nötig sein, der vom

Tunnel erzeugt wird.

3.2 TOS-Field Aggregation

Bei der TOS-Field Aggregation wird eine andere Technik verwendet. Betrachten wir dazu die Abb. 3.2 und erinnern wir uns an Abb. 2.5

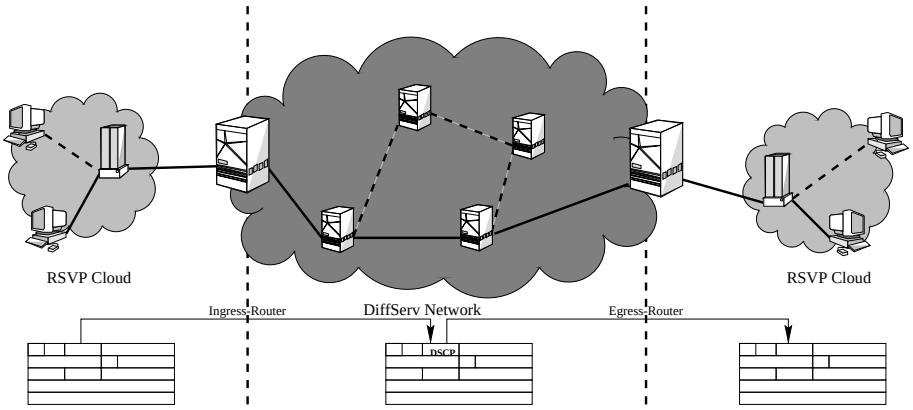

Abbildung 3.2: Netzwerk, bei dem zwischen den beiden RSVP-Netzen eine TOS-Field Aggregation durchgeführt wird.

Verfolgen wir ein Datenpaket. Sobald es den Eintrittspunkt zum DiffServ-Netz erreicht muss das Paket markiert werden, um die entsprechende Dienstgüte sicherzustellen. Danach wird das Paket dem DSCP entsprechend weitergeleitet. Sobald das Paket den Austrittspunkt erreicht, kann die Markierung wieder rückgängig gemacht werden. Das Paket wird dann normal an den Empfänger weitergeleitet.
Um die Markierung zu vereinfachen, kann bereits der Sender den entsprechenden DSCP setzten, da dieser in einem RSVP-Netz nicht entscheidend ist.
Will man eine Reservierung der Bandbreite zwischen dem Eintritts- und dem Austrittspunkt erhalten, muss eine spezielle Kommunikation zwischen diesen beiden existieren. Eine weitere Möglichkeit wäre es auf bestimmte DSCP zurückzugreifen, die eine gewisse Bandbreite zur Verfügung stellen und statisch vorkonfiguriert sind. Wird auf einen vorkonfigurierten DSCP zurückgegriffen, so müssen die Eintrittspunkte so intelligent sein die erlaubte Reservierung nicht

überschreiten.
Dieser Ansatz entspricht auch einer Ausweitung des DiffServ-Ansatzes auf ein globales Netz. So kann an einem Eintrittspunkt ein Remarking durchgeführt werden, um verschieden Datenströme zu vereinen.
Da es vorkommen kann, dass im DiffServ-Bereich auch Router existieren, die das RSVP-Protokoll verstehen, aber eine doppelte Reservierung nicht sinnvoll ist, muss die Path-Nachricht so behandelt werden, dass dies nicht geschehen kann. Am einfachsten ist dies, indem man die Router-Alert-Option ausschaltet. Dies verlangt aber, dass beim Austrittspunkt das Router-Alert-Flag wieder gesetzt wird, was bedeutet, dass jedes Paket darauf kontrolliert werden muss ob es sich um eine Path-Nachricht handelt.

3.3 Bandwidth-Broker

Wenn die Struktur der zu durchquerenden Netze kompliziert wird, kann es zu Problemen bei der Etablierung einer gewissen Dienstgüte kommen, da die Daten zu häufig einer Transformation unterliegen. Betrachten wir die Abb. 3.3.

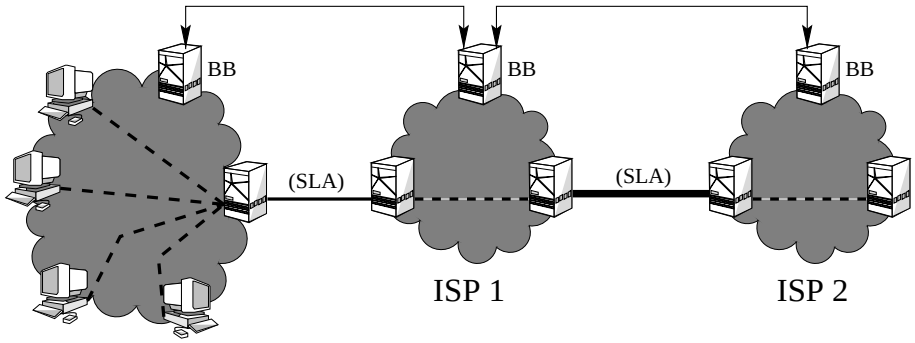

Abbildung 3.3: Netzwerk, bestehend aus zwei ISP, welche untereinander ihre SLA mit Hilfe von Bandwidth Brokern (BB) aushandeln und etablieren.

Wenn die Verbindung über mehrere Internet Service Provider (ISP) geht, gibt es bei der Vergabe von Ressourcen Probleme. Als Endbenutzer schliesst man im Prinzip nur mit seinem lokalen ISP eine Vereinbarung. Dieser hat wiederum mit anderen angrenzenden ISP Service Level Agreement (SLA) vereinbart. Dadurch kann der Fall auftreten, dass ein Endbenutzer von seinem ISP noch verfügbare

Ressourcen möchte, aber bei einem entfernten ISP keine weiteren Ressourcen vorhanden sind und die gewünschte Dienstgüte nicht garantiert werden kann. Um dies zu verhindern kann man nun zwei Ansätze machen.

- statisches Management: Die SLA werden auf Grund von statistisch erhobenen Daten fest ausgehandelt. Es wird dabei immer eine gewisse Reserve eingeplant. Trotzdem können immer Spitzen auftreten, die die Reserve überschreiten. Entsprechen dieser SLA wird auch die zugrundeliegende Hardware angepasst.

- dynamisches Management: Bei diesem Ansatz können die SLAs neu Ausgehandelt werden. Wie eine entsprechende Kommunikation aussehen könnte wird in [GB99] gezeigt. Nun ganz kurz, wie ein dynamisches Management aussehen könnte. Jeder ISP verfügt über eine Kontrollinstanz (Bandwidth Broker BB), die den aktuellen Zustand des eigenen Netzwerkes kennt. Wenn Ressourcen verlangt werden, entscheidet der BB ob diese vorhanden sind. Sind genügend Ressourcen vorhanden, so können diese vergeben werden. Geht eine Reservierung über mehrere ISPs hinweg (Abb. 3.3), so muss der BB bei den BB's der angrenzenden ISPs anfragen ob die Ressourcen auch dort vorhanden sind. Sind bei einem ISP nicht mehr genügend Ressourcen vorhanden, so wird die Reservierung auf der ganzen Strecke nicht etabliert.

Kapitel 4

Eigene Ansätze

4.1 Erweiterung von RSVP

Eigentlich ist dies eine Kombination von Bandwidth-Broker und TOS-Field Aggregation. Es soll dabei im letzten Router vor dem DiffServ-Netz eine Transformation von IntServ $\leftrightarrow$ DiffServ stattfinden (TOS-Field Aggregation). Der Bandwidth-Broker Ansatz wird verwendet, um eine dynamische Reservierung von Bandbreite im DiffServ-Netz zu ermöglichen.
Dabei sollte durch Kommunikation die Grenzen des DiffServ-Netzes festgestellt werden.
Für diesen Ansatz werden folgende Annahmen getroffen:

- Für die Umsetzung der verschiedenen Dienstgüten zwischen den einzelnen DiffServ-Netzbetreiber sind diese selber verantwortlich.
- Jeder Ingress, bzw. Egress-Router eines DiffServ-Netzes muss das RSVP-Protokoll unterstützen. Oder der RSVP-Egress/Igress-Router muss für die Umsetzung von Integrated-Services zu Differentiated-Services sorgen ($\rightarrow$ Border-Router).
- Definition eines neues Klassen Objektes *Edge-Router*, dass von normalen RSVP-Routern nicht verarbeitet wird (Class-Num = 11bbbbbb), aber trotzdem weitergeleitet werden muss [BZB+97]. Dies ermöglicht die Kom-

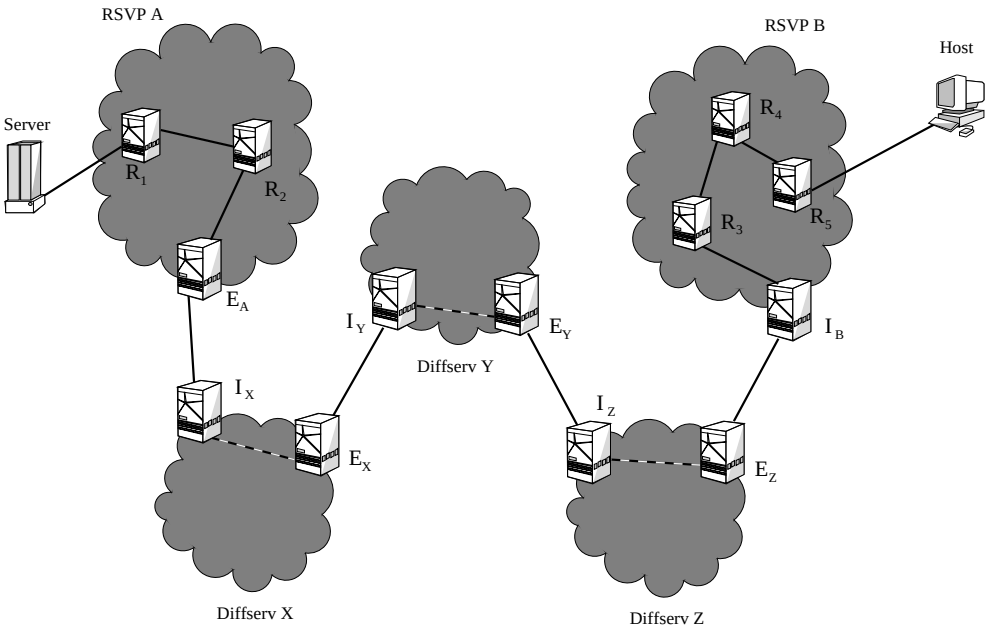

Abbildung 4.1: Beispiel einer Verbindung über zwei RSVP und drei DiffServ-Netzwerke. Dabei unterstützen die mit R_x bezeichnete Router RSVP. Die beiden Router E_A und I_B sind Border-Router und stellen den Kontakt und die Transformationen in die DiffServ-Regionen sicher.

munikation mit Hilfe von RSVP. Das Objekt wird mit der Path-Nachrichten übertragen.

4.1.1 Beispiel einer Verbindung

Eine Verbindung mit Datenfluss würde dann beispielsweise wie folgt aussehen:

1. Der Empfänger (Host) öffnet eine Verbindung (z.B. ftp) und verlangt spezifische Daten.

2. Der Sender (Server) initiiert eine Path-Nachricht mit seiner Adresse als Absender und die des Empfängers als Empfänger Adresse, sowie allen relevanten Daten für eine Reservierung ($\rightarrow$ Tspec).

   (a) Bei jedem Router (z.B. R_1), bei dem das Paket vorbei kommt und der RSVP unterstützt, wird der PHOP (Bei R_1 wäre dies die Adresse des Server) gespeichert (RSVP_HOP-Objekt der Path-Nachricht) und

neues RSVP_HOP-Objekt erzeugt, das das empfangene ersetzt.

(b) Befindet sich der Rechner an der Grenze eines Netzes (I_X oder E_X), so kontrolliert er, ob er an der Grenze zwischen einem IntServ/DiffServ Netz ist. Ist dies der Fall (z.B. E_A), so erzeugt er ein neues *EdgeRouter* Objekt und wertet ein eventuell bestehendes *EdgeRouter* Objekt aus.

3. Sobald die Path-Nachricht angekommen ist, kontrolliert der Empfänger deren Inhalt und erzeugt die entsprechende Resv-Nachricht. Diese wird nun gesendet, wobei die folgenden Aktionen bei deren Verarbeitung ablaufen:

   (a) Handelt es sich um einen normalen RSVP-Router, so wird die Reservierung normal durchgeführt (Merging, ...) und dann normal weitergeleitet.

   (b) Handelt es sich um einen Border-Router (I_B), bei dem die Resv-Nachricht in ein DiffServ-Netz geht, so muss er ermitteln, wieviel Bandbreite ihm noch zur Verfügung steht:

   - Hat er genug, so übermittelt er dem nächsten Border-Router, dass er für die Reservierung noch genügend Bandbreite zur Verfügung hat und leitet die Resv-Nachricht weiter, nachdem er die Reservierung gemacht hat.
   - Hat er nicht genug Bandbreite zur Verfügung, so fragt er beim Netzbetreiber an, ob er mehr Bandbreite bekommen kann. Ist dies der Fall, wird wie oben fortgefahren. Ist dies nicht der Fall wird die Resvierung abgelehnt.

   (c) Handelt es sich um einen Border-Router (E_A), bei dem die Resv-Nachricht von einem DiffServ-Netz zurück in ein IntServ-Netz geht, werden die entsprechenden Reservierungen gemacht und alles für eine TOS-Field Aggregation vorbereitet.

4.2 Ansätze mit zwei kommerziellen Backbone-Routern

Während dieser Arbeit stellte sich heraus, dass die zur Verfügung stehenden kommerziellen Backbone-Router über eine RSVP-Unterstützung und einer Möglichkeit der Aggregation über DiffServ verfügen. Darum fiel der Entscheid, dass die zwei Router fest im Szenario verankert werden mussten. Dies führt zu einem vereinfachten Ansatz eines Netzes, welches der Abb. 4.2 entspricht. Das dazwischenliegende DiffServ-Netz ist bei der weiteren Betrachtung irrelevant.

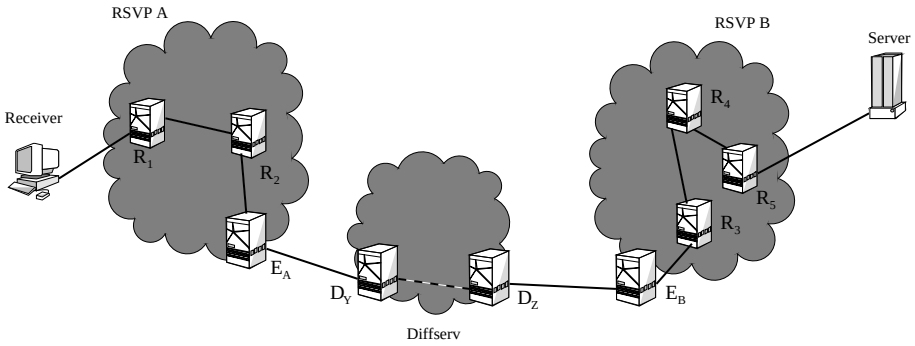

Abbildung 4.2: Beispiel einer Verbindung zweier RSVP-Netzwerke über ein DiffServ-Netzwerken. Dabei unterstützen die mit R_x bezeichnete Router RSVP. Die beiden Router E_A und E_B sind Edge-Router und stellen den Kontakt und die Transformationen in die DiffServ-Regionen (Router D_x) sicher.

Die Steuerung für die Bandbreite über das DiffServ-Netz soll dabei über einen Bandwidth Broker (BB) sichergestellt werden, der für die Konfiguration der Router zuständig ist.
Da das Betriebssystem der beiden Backbone-Router nicht abänderbar war, wurde beschlossen ihnen einen Linux-Router zur Seite zu stellen. Dieses Paar bildet nun die Verbindung zwischen dem IntServ- und dem DiffServ-Netz. Daraus entstand das in Abb. 4.3 aufgezeigte Testnetzwerk.
Um entstandene Probleme zu erkennen oder vorzubeugen, kann man auf drei verschieden RSVP-Nachrichten zurückgreifen. Es sei an die Abb. 2.2 erinnert.

- **Path-Nachricht:** Die Path-Nachricht passiert zuerst den Linux-Router,

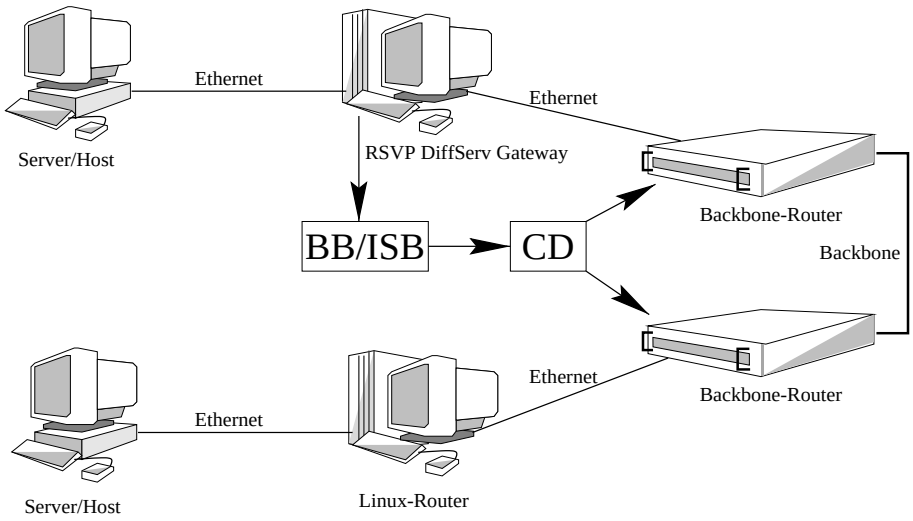

Abbildung 4.3: Aufbau eines Netzes mit 2 Backbone- und 2 Linux-Routern.

bevor sie an die Backbone-Router weitergegeben wird. Der Linux-Router kontrolliert ob der Backbone-Router noch genügend Bandbreite zur Verfügung hat und handelt dem entsprechend.

- **Resv-Nachricht:** Der eigene Linux-Router steht hierbei hinter dem Backbone-Router. Also kann dieser nicht auf Probleme direkt reagieren. Es gibt jetzt zwei Möglichkeiten, um dies zu lösen. Die eine Möglichkeit ist, dass die RSVP-Kontrolle des Backbone-Routers ausgeschaltet wird und der Linux-Router die Verantwortung für beide Maschinen übernimmt. Bei der zweiten Möglichkeit, würde der Linux-Router des anderen Paares anhand der Resv-Nachricht die entsprechenden Massnahmen einleiten. Was zu einem ähnlichen Ablauf wie bei der nächsten Idee führt.

- **ResvErr-Nachricht:** Der Linux-Router hat keine speziellen Kenntnisse über den Backbone-Router und reagiert erst, wenn eine Reservierung fehlschlägt. Der Linux-Router wertet die ResvErr-Nachricht aus und handelt dementsprechend.

Um Missbrauch zu verhindern, haben die Linux-Router keine direkte Kontrolle über die Backbone-Router. Tritt im Router-Paar ein Problem auf, weil nicht mehr genug Bandbreite zur Verfügung steht, so wird beim BB nachgefragt.

Dieser entscheidet, ob die Anfrage akzeptiert oder endgültig abgelehnt wird.
Aus dem oben beschriebenen Voraussetzungen wurden die fünf Konzepte entwickelt. Dabei wird ein Netzwerk, wie in Abb. 4.3 vorgestellt, verwendet. Die Edge-Router bestehen dabei aus einem Linux- und einem Backbone-Router (Lx und Cx) und zur Konfiguration der Backbone-Router wird der Konfiguration-Daemon (CD) verwendet, der über den Bandwidth-Broker (BB/ISB) gesteuert wird.

4.2.1 Verwendung der ResvErr-Nachricht

Die Idee

Wenn die Backbone-Router volle Kontrolle über die RSVP-Flüsse haben sollen und die Aggregation darüber läuft, so müssen die Linux-Router ausserhalb der Backbone-Router stehen. Sobald aber die Linux-Router ausserhalb stehen, erreichen Reservierungsanfragen zuerst in die Backbone-Router. Dadurch ist der Backbone-Router nicht mehr durch den dazugehörigen Linux-Router zu kontrollieren. Um aber trotzdem eine dynamische Reservierung zu ermöglichen, muss der Linux-Router des anderen (Backbone-Linux) Paares den Backbone-Router konfigurieren. Wenn eine Reservierung scheitert, erhält dieser Linux-Router eine ResvErr-Nachricht, auf die er reagieren kann.
Ein grosses Problem ist dabei, dass bei einer gescheiterten Reservierung in den Backbone-Router Blockade-States bestehen bleiben, die gelöscht werden müssen. Um ein zu häufiges Reagieren zu verhindern, sollte der Backbone-Router so konfiguriert sein, dass eine gewisse Reserve vorhanden ist und nicht bei jeder Reservierung neu konfiguriert werden muss.

Voraussetzungen

Für die Umsetzung der Idee werden folgende neue Komponenten benötigt.

- Ein CONFIG-Objekt, das die Adresse des Backbone-Routers und die Adresse des BB/ISB enthält.

- Eine get-Bandwidth-Nachricht, um beim BB/ISB eine Reservierung der Bandbreite zu verlangen.
- Eine conf-Bandwidth-Nachricht, um eine Bestätigung der Reservierung zu erhalten.
- Eine ERASE-Objekt, um die Path-Blockade-States zu löschen.
- Einer Reset-Path-Nachricht, um den Path neu aufzusetzen.

Erzeugen einer Reservierung

Abbildung 4.4 stellt den Ablauf unter Verwendung der ResvErr-Nachricht dar.

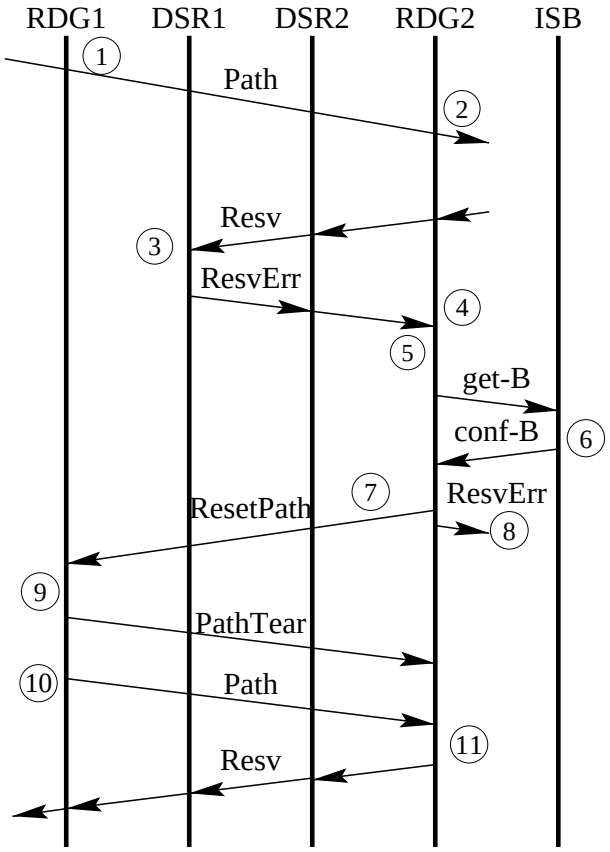

Abbildung 4.4: Zeitdiagramm beim Verwenden von ResvErr

Dabei passiert an den verschiedenen Punkten folgendes:

1. Die Path-Nachricht erreicht den Linux-Router (L1). Dieser hängt ein

CONFIG-Objekt an die Path-Nachricht an und schickt diese normal weiter.

2. Sobald die Path-Nachricht den Linux-Router (L2) erreicht, liest dieser das CONFIG-Objekt aus und löscht das Objekt.

3. Bei der Reservierung der Bandbreite im Backbone-Router (C1) tritt ein Fehler auf, weil die gewünschte Bandbreite nicht verfügbar ist (der andere Fall ist nicht interessant). Der Backbone-Router erzeugt eine ResvErr-Nachricht und sendet diese in Richtung des Initator der Reservierung zurück.

4. Der Linux-Router (L2) erhält die ResvErr-Nachricht und stellt fest, dass sie von einem Router stammt, der konfiguriert werden kann und von dem er die relevanten Daten hat (Adresse des Routers, Adresse des BB/ISB).

5. Der Linux-Router (L2) blockiert die ResvErr-Nachricht und sendet an den BB/ISB eine get-Bandwidth-Nachricht, um die entsprechende Bandbreite für den Backbone-Router zu reservieren.

6. Der BB/ISB kontrolliert ob er die Reservierung machen darf oder nicht. Ist die verlangte Bandbreite erhältlich, so konfiguriert er den Backbone-Router (C1) und sendet dem Linux-Router (L2) eine positive conf-Bandwidth-Nachricht zurück. Ist die Bandbreite nicht verfügbar, so wird beim Backbone-Router nichts geändert und der Linux-Router (L2) erhält eine negative conf-Bandwidth-Nachricht.

7. Die conf-Bandwidth-Nachricht war positiv. Der Linux-Router (L2) meldet dies dem Linux-Router (L1) mit der Reset-Path-Nachricht.

8. Die conf-Bandwidth-Nachricht war negativ. Die Reservierung ist nicht möglich und die ResvErr-Nachricht wird nun normal weitergeleitet.

9. Der Linux-Router (L1) erhält die Reset-Path-Nachricht um den Path-State zu reinitialisieren und erzeugt eine PathTear-Nachricht um die Path-

Blockade-State in den Backbone-Routern zu löschen (Es werden die ganzen Path-States gelöscht). Er hängt an diese Meldung ein Path-ERASE Objekt an.

10. Der Linux-Router erzeugt nun eine neue Path-Nachricht, um die Path-State neu aufzubauen.

11. Nachdem die Path-Nachricht erneut den Linux-Router (L2) erreicht hat, erzeugt dieser eine neue Resv-Nachricht und die Reservierung kann normal erfolgen.

4.2.2 Verwenden der Resv-Nachricht

Die Idee

Auch hier treffen die Reservierungs-Anfragen zuerst beim Backbone-Router ein. Will man die Problem vom ResvErr-Ansatz umgehen, so kann man auch auf Grund der Resv-Nachricht die beteiligten Backbone-Router vorkonfigurieren, so dass kein Fehler entsteht, der dann durch den ResvErr-Ansatz abgefangen wird.

Voraussetzungen

Für die Umsetzung der Idee werden folgende neue Komponenten benötigt.

- Ein CONFIG-Objekt, das die Adresse des Backbone-Routers und die Adresse des BB/ISB enthält.

- Eine get-Bandwidth-Nachricht, um beim BB/ISB eine Reservierung der Bandbreite zu verlangen.

- Eine conf-Bandwidth-Nachricht, um eine Bestätigung der Reservierung zu erhalten.

Erzeugen der Reservierung

In Abb. 4.5 wird der Ablauf mit Verwendung der Resv-Nachricht dargestellt. Dabei passiert an den verschiedenen Punkten folgendes:

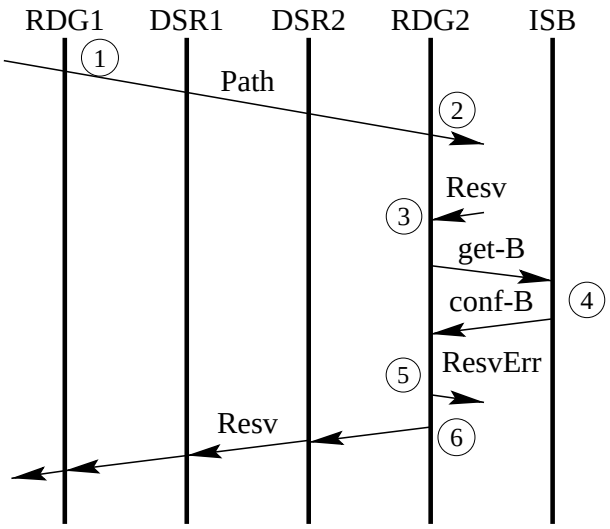

Abbildung 4.5: Zeitdiagramm beim Verwenden von Resv

1. Die Path-Nachricht erreicht den Linux-Router (L1). Dieser hängt ein CONFIG-Objekt an die Path-Nachricht an und schickt diese normal weiter.

2. Sobald die Path-Nachricht den Linux-Router (L2) erreicht, liest dieser das CONFIG-Objekt aus und löscht das Objekt.

3. Die Resv-Nachricht erreicht den Linux-Router (L2). Er sendet an den BB/ISB eine get-Bandwidth-Nachricht, um die entsprechende Bandbreite für den Backbone-Router zu reservieren.

4. Der BB/ISB kontrolliert, ob er die Reservierung machen darf oder nicht. Ist die verlangte Bandbreite erhältlich, so konfiguriert er den Backbone-Router (C1) und sendet dem Linux-Router (L2) eine positive conf-Bandwidth-Nachricht zurück. Ist die Bandbreite nicht verfügbar, so wird beim Backbone-Router nichts geändert und der Linux-Router (L2) erhält eine negative conf-Bandwidth-Nachricht.

5. Die conf-Bandwidth-Nachricht war negativ. Die Reservierung ist nicht möglich und eine ResvErr-Nachricht wird erzeugt und normal weitergeleitet.

6. Die conf-Bandwidth-Nachricht war positiv. Der Linux-Router (L2) sendet die Resv-Nachricht normal weiter.

4.2.3 Verwenden der Path-Nachricht

Die Idee

Damit möglichst wenig Overhead entsteht, sollte eine Anpassung des Backbone-Routers in einem möglichst frühzeitigem Stadium der Signalisierung passieren. Da der Linux-Router nur bei der Path-Nachricht vor seinem Backbone-Router steht, kann das dann geschehen.
Problematisch ist nur, dass kein Feedback vom Backbone-Router kommt. Der Linux-Router sollte aber den Zustand des Backbone-Routers kennen. Ein Vorteil ist, dass der RSVP-Daemon auf dem Linux-Router relativ einfach sein kann, weil er zur Verwaltung nur Path-, PathTear- und ResvTear-Nachrichten verarbeiten muss.

Voraussetzungen

Für die Umsetzung der Idee werden folgende neue Komponenten benötigt.

- Eine get-Bandwidth-Nachricht, um beim BB/ISB eine Reservierung des Bandbreite zu verlangen.
- Eine conf-Bandwidth-Nachricht, um eine Bestätigung der Reservierung zu erhalten.

Zusätzlich wird vorausgesetzt, dass

- der Linux-Router über den Backbone-Router genau informiert ist, d.h. er muss wissen, wieviel Bandbreite vorhanden ist und wieviel schon vergeben wurde.

Erzeugen einer Reservierung

In Abb. 4.6 wird der Ablauf mit Verwendung der Path-Nachricht dargestellt. Dabei passiert an den verschiedenen Punkten folgendes:

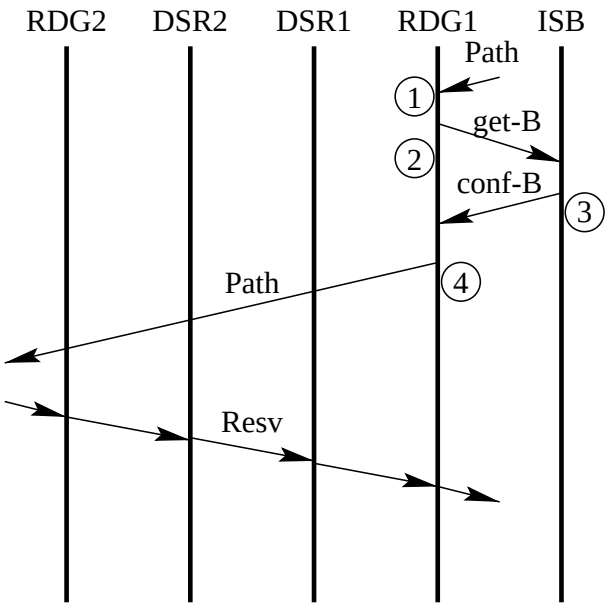

Abbildung 4.6: Zeitdiagramm beim Verwenden von Path

1. Die Path-Nachricht erreicht den Linux-Router (L1). Dieser ermittelt ob noch genug Bandbreite vorhanden ist. Ist dies der Fall, so wird die Path-Nachricht sofort weitergesandt, trifft dies nicht zu, so wird die Path-Nachricht zwischengespeichert.

2. War nicht genug Bandbreite beim Backbone-Router vorhanden, so wird diese beim BB/ISB angefordert.

3. Dieser kontrolliert nun, ob die Bandbreite zur Verfügung gestellt werden kann. Ist dies der Fall, so konfiguriert er den Backbone-Router entsprechend und sendet eine positive Antwort an den Linux-Router. Kann er die Bandbreite nicht zuteilen, so sendet er dem Linux-Router eine negative Antwort zurück.

4. Sobald der Linux-Router eine Antwort des BB/ISB bekommt sendet er die zwischengespeicherte Path-Nachricht weiter, egal ob sie nun positiv oder negativ war.

4.2.4 Backbone-Router mit maximaler Reservierung für RSVP-Flows

Die Idee

Die Backbone-Router werden so konfiguriert, dass sie von Anfang an die maximal für RSVP zugelassene Bandbreite reserviert haben (Bandbreite wird noch nicht gebraucht und steht somit anderen Diensten zur Verfügung). Somit ist gewährleistet, dass ein maximalen Wert (fest vorgegeben) nicht überschritten wird und der Backbone-Router die Aggregation der RSVP-Datenflüsse übernimmt. Der Linux-Router muss jetzt nur noch für jeden einzelnen RSVP-Datenfluss beim BB/ISB nachfragen, ob dieser genehmigt werden kann. Dementsprechend wird dann auch der Linux-Router konfiguriert.

Voraussetzungen

Für die Umsetzung der Idee werden folgende, neue Komponenten benötigt.

- Eine get-Bandwidth-Nachricht, um beim BB/ISB eine Reservierung des Bandbreite zu verlangen.
- Eine conf-Bandwidth-Nachricht, um eine Bestätigung der Reservierung vom BB/ISB zu erhalten.

Zusätzlich wird vorausgesetzt, dass

- die Backbone-Router vorkonfiguriert sind und eine fest vorgegebene maximale Bandbreite für RSVP erlauben, die eventuell angepasst werden kann.
- Die Admission-Control geschieht in den Linux-Routern.

Erzeugen einer Reservierung

In Abb. 4.7 wird der Ablauf mit Verwendung von fest vorkonfigurierten Backbone-Routern dargestellt.
Dabei passiert an den verschiedenen Punkten folgendes:

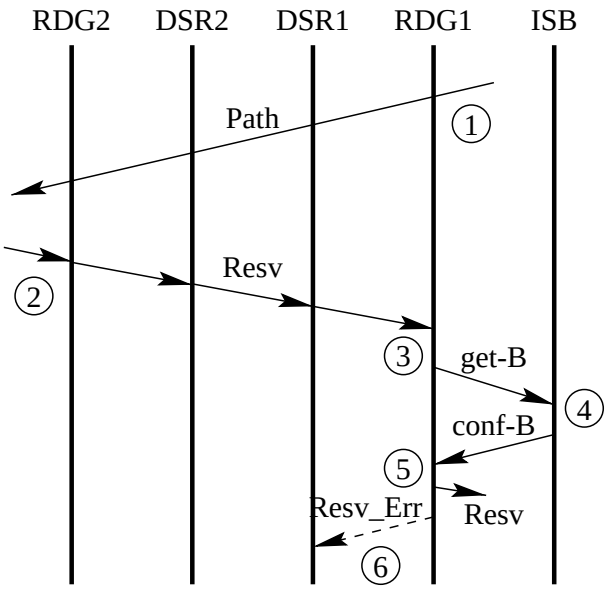

Abbildung 4.7: Zeitdiagramm beim Verwenden von fest vorkonfigurierten Backbone-Routern.

1. Die Path-Nachricht erreicht den Linux-Router (L1) und wird normal durch alle weiteren Stationen weitergeleitet. Wobei die entsprechenden Path-States erzeugt werden (Standart).

2. Die Resv-Nachricht erreicht den Linux-Router (L2) und wird normal weitergeleitet (gemäss RSVP-Standart). Die Backbone-Router machen die Reservierung, wenn die fest vorgegebene Bandbreite nicht überschritten wird.

3. Die Resv-Nachricht erreicht den Linux-Router (L1). Der Linux-Router (L1) sendet dem BB/ISB eine get-Bandwidth-Nachricht um anzufragen, ob die Bandbreite noch gewährt werden darf.

4. Der BB/ISB kontrolliert, ob noch genug Bandbreite vorhanden ist. Er konfiguriert den Backbone-Router, je nachdem, ob dieser dynamisch oder statisch ist und erzeugt eine entsprechend positive oder negative conf-Bandwidth-Nachricht.

5. Ist die Antwort des BB/ISB positiv, wird die Reservierung gemacht und die Resv-Nachricht normal weitergeleitet.

6. Ist die Antwort des BB/ISB negativ, wird die Reservierung verworfen und eine ResvErr-Nachricht erzeugt.

4.2.5 Aggregation über den Linux-Router

Die Idee

Bei den beiden Backbone-Routern ist die RSVP-Unterstützung ausgeschaltet. Die Linux-Router sind für die RSVP-Datenfluss-Kontrolle zuständig und konfigurieren dabei die Backbone-Router mit Hilfe des BB/ISB. Der BB/ISB bekommt Anfragen des Linux-Routers (mit Datenflussbeschreibung) und entscheidet danach, ob die Reservierung zulässig ist oder nicht. Wenn die Reservierung zulässig ist, konfiguriert der BB/ISB den Backbone-Router so, dass die QoS zwischen den Backbone-Routern gewährleistet ist.

Voraussetzungen

Für die Umsetzung der Idee werden folgende neue Komponenten benötigt.

- Eine get-Bandwidth-Nachricht, um beim BB/ISB eine Reservierung des Bandbreite zu verlangen (und eventuell wieder zu löschen).
- Eine conf-Bandwidth-Nachricht, um eine Bestätigung der Reservierung zu erhalten.

Zusätzlich wird vorausgesetzt, dass

- Bei den Backbone-Routern ist RSVP ausgeschaltet.
- TC Mechanismus bei den Backbone-Router.

Erzeugen einer Reservierung

In Abb. 4.8 wird der Ablauf mit Verwendung von zwei Backbone-Routern mit abgeschalteter RSVP-Unterstützung dargestellt.
Dabei passiert an den verschiedenen Punkten folgendes:

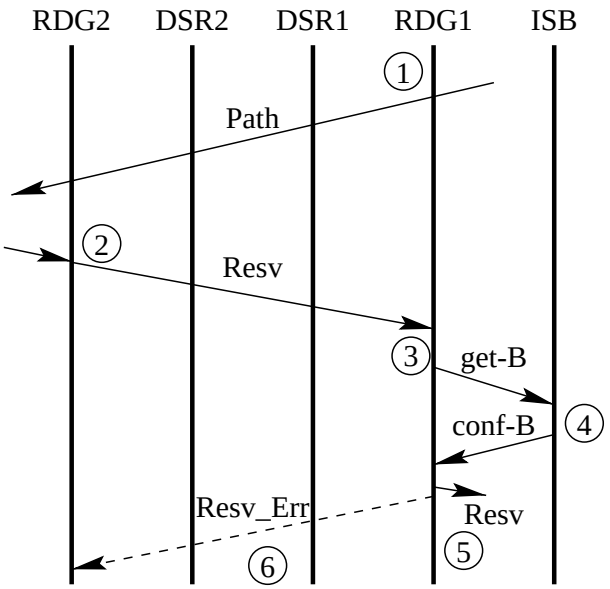

Abbildung 4.8: Zeitdiagramm beim Verwenden von zwei Backbone-Routern mit abgeschaltetem RSVP.

1. Die Path-Nachricht erreicht den Linux-Router (L1) und wird normal durch alle weiteren Stationen weitergeleitet. Dabei geschieht in den Backbone-Routern nichts, da ja RSVP ausgeschaltet ist.

2. Die Resv-Nachricht erreicht den Linux-Router (L2) und wird normal weitergeleitet $\rightarrow$ L1 (gemäss RSVP-Standart). Da die Backbone-Router keine RSVP-Unterstützung haben, machen diese nichts.

3. Die Resv-Nachricht erreicht den Linux-Router (L1). Dieser sendet nun an den BB/ISB eine get-Bandwidth-Nachricht, um die entsprechende Bandbreite zu erhalten.

4. Der BB/ISB entscheidet, ob er die Bandbreite zur Verfügung stellen kann oder nicht. Ist die Bandbreite verfügbar, so wird der Backbone-Router entsprechend konfiguriert. Der BB/ISB schickt dem Linux-Router eine positive oder negative Bestätigung (conf-Bandwidth-Nachricht).

5. Ist die Antwort positiv, wird die Reservierung gemacht und die Resv-Nachricht normal weitergeleitet.

6. Ist sie hingegen negativ, wird die Reservierung abgelehnt und eine ResvErr-Nachricht erzeugt.

Kapitel 5

Vergleiche der RSVP-DS Konzepte

In diesem Kapitel werden die vorgestellten Ansätze miteinander verglichen. Zuerst vergleichen wir die drei aus dem Kapitel 3, bevor wir auf diejenigen aus dem Kapitel 4 eingehen.

5.1 IP-Tunneling, TOS-Field-Aggregation und Bandwidth-Broker

Der Bandwidth-Broker-Ansatz ist eigentlich mit dem IP-Tunneling und der TOS-Field-Aggregation schwer vergleichbar, da er nicht auf bestimmte Komponenten zurückgreifen muss. Der BB-Ansatz befasst sich mit dem Aushandlungsverfahren für eine Bandbreite und befasst sich, im Gegensatz zu IP-Tunneling und TOS-Field-Aggregation, nicht damit, wie die Reservierung realisiert wird. Um die Reservierungen umzusetzen, werden dann die beiden anderen Konzepte verwendet. Der Vorteil eines Bandwidth-Brokers liegt darin, dass für die Reservierung eine Instanz entscheidet, welche sich normalerweise oberhalb der Routerebene befindet und deshalb die Übersicht über das ganze Netz eines ISP hat.
Für IP-Tunneling und TOS-Field-Aggregration betrachten wir die Tabelle 5.1. Daraus können wir ableiten, dass der IP-Tunneling Ansatz für statische Netze

Art	Vorteile	Nachteile
IP-Tunneling	• Gut geeignet, um mehrere verteilte Netze miteinander zu verbinden. • Man kann den RSVP-Ansatz auf das ganze Netz ausdehnen.	• Anfang und Ende des Tunnels sind relativ starr.
TOS-Field-Aggegration	• Anfang und Ende der DiffServ-Region können sehr flexibel sein. • Die DiffServ-Region kann bis zum Endpunkt ausgedehnt werden.	• Je nach Punkt der Initialisierung des TOS-Fields, ein Problem der Sicherheit.

Tabelle 5.1: Vor- und Nachteile der Integration mit IP-Tunneling und TOS-Field-Aggregration.

praktischer ist, als die TOS-Field-Aggegration. Sollen hingegen die Zugriffe im Netz flexibel sein, kommt man mit der TOS-Field-Aggregation besser zurecht.

5.2 Vergleich der eigene Ansätze

Um die Ansätze aus Kapitel 4 zu vergleichen betrachten wir zuerst die einzelnen Probleme, bevor wir einen Vergleich der nötigen Anpassungen von RSVP anstellen. Danach folgt ein abschliessender Vergleich.

5.2.1 Verschiedene Probleme

- Erweiterung von RSVP
  - Das Erkennen der Grenzen zwischen DiffServ- und IntServ-Netzen kann schwierig werden, wenn es Router innerhalb von DiffServ-Netzen gibt, die ebenfalls RSVP unterstützen. Vor allem dann, wenn diese auch ein erweiteres RSVP verwenden.

- Bei Verwendung der ResvErr-Nachricht
  - Sobald eine ResvErr-Nachricht erzeugt wird, entsteht gleichzeitig ein Blockade-State. Dieser blockiert alle neuen Reservierungen für diesen Path-State, die nicht echt kleiner ist, als die, die abgelehnt wurde. Dieser Blockade-State wird normalerweise nach 2 Zeitzyklen gelöscht, was aber bei einem Default-Wert von 30 Sekunden für eine Reservierung zu lange dauert.
  - Wenn die Reservierung nicht im ersten Anlauf erfolgreich war, entsteht ein grosser Overhead um die Reservierung erneut aufsetzen zu können.
- Verwenden der Resv-Nachricht
  - Der Linux-Router muss mit einem Backbone-Router kommunizieren, der nicht zum Router-Paar gehört. Dieser ist eventuell weit entfernt und gehört nicht dem selben ISP. Daraus können sich Sicherheits- und Übertragungsprobleme ergeben.
- Verwenden der Path-Nachricht
  - Der Linux-Router kennt nicht immer den genau Zustand des Backbone-Routers Bescheid.
  - Der Linux-Router weiss nicht ob, eine Reservierung schon zustande gekommen ist oder nicht.
  - Der Linux-Router muss über den Backbone-Router genau informiert sein, d.h. er muss wissen, wieviel Bandbreite vorhanden ist und wieviel schon vergeben wurde.
- Backbone-Router hat maximale Reservierung für RSVP-Flows
  - Backbone-Router wird nur noch statisch angepasst.
- Aggregation über Linux-Router
  - Linux-Router muss volle Flusskontrolle gewährleisten können.

5.2.2 Nötige Anpassungen

Um die einzelnen Ansätze zu unterstützen, muss RSVP etwas erweitert werden. Zuerst betrachten wir die Veränderungen, die bei allen Ansätzen gebraucht werden, da sie im Prinzip alle auf dem Bandwidth-Broker-Modell aufbauen. Für die Kommunikation zum BB benötigt man die folgenden neuen Komponenten:

- Eine neue Nachricht *get-Bandwidth*, um die Reservierung von Bandbreite zu beantragen. Sie sollte mindestens die folgenden Elementen haben: FlowID und benötigte Bandbreite.

- Eine neue Nachricht *conf-Bandwidth*, um die Reservierung zu bestätigen. Sie sollte mindestens die folgenden Elemente haben: Flag ob die Reservierung gestattet wird. Eventuell einen zusätzlicher Parameter, um die jeweilige Reservierung zu identifizieren.

- Eventuell eine neue Nachricht *release-Bandwidth*, um eine Reservierung freizugeben. Sie sollte die folgenden Elemente haben: Empfänger-Adresse und freizugebende Bandbreite.

  Diese Nachricht kann auch mit Hilfe von *get-Bandwidth* erzeugt werden, indem diese um einen entsprechenden Parameter ergänzt wird.

Zusätzlich werden für einige der Ansätze noch zusätzliche Komponenten benötigt:

- Erweiterung von RSVP

  - Ein neues Object *EDGE-ROUTER*, um die Grenzen der verschiedenen Netze zu erkennen. Dieses Objekt sollte von den Edge-Routern erkannt werden, ohne dass diese von Routern innerhalb der Netze verarbeitet oder gelöscht werden.

- Verwendung der ResvErr-Nachricht

  - Eine neue Nachricht *Reset-Path*, um die Neuinitialisierung der Path-Nachricht zu verlangen. Sie sollte die folgenden Elemente haben:

Empfänger-Adresse und Port, Sender-Adresse und Port (und eventuell die Sendespezifikationen).

  - Ein neues Objekt *ERASE*, um die PathTear-Nachricht zu beschränken. Dieses Objekt braucht keine Daten zu enthalten.
  - Ein neues Objekt *CONFIG*, um die Adressen des Backbone- und Linux-Router, sowie des BB/ISB anzugeben.
  - Neue Felder im Path-State-Block, für die Adresse des Linux- und des Backbone-Routers sowie die des Bandwidth-Brokers.

- Verwendung der Resv-Nachricht

  - Neues Objekt *CONFIG*, um die Adressen des Backbone- und Linux-Router, sowie des BB/ISB anzugeben.
  - Neue Felder im Path-State-Block, für die Adresse des Linux-Routers, des Backbone-Routers und des Konfiguration-Daemon.

5.2.3 Abschliessender Vergleich

Die Erweiterung von RSVP ähnelt der Variante mit der Resv-Nachricht. Hier wird nur die Kommunikation zum BB umgangen, indem durch die Erweiterung des RSVP-Protokols die Grenzen der DiffServ-Region festgelegt werden und zudem davon ausgegangen wird, dass die Router an die Voraussetzungen anpassbar sind. Dies ist auch der einzige Ansatz, der nicht ein Router-Paar aus Linux- und Backbone-Router verwendet, wie dies bei den anderen Ansätzen der Fall ist und kann deshalb nicht mit diesen verglichen werden.
Vergleicht man die einzelnen Ansätze untereinander, so kommt man zu dem Punkt, dass bei der Verwendung von ResvErr ein enormer Aufwand an Datenkommunikation auftritt, sofern eine Reservierung nicht auf Anhieb gelingt, da die Blockade-States zuerst entfernt werden müssen. Ausserdem ist nicht sicher, ob die Reservierung bei einem zweiten Anlauf erfolgreich sein wird, was den Aufwand erheblich erhöht. Zudem muss ein Linux-Router reagieren, der nicht direkt zum Router-Paar gehört. Dies wirft die folgenden Probleme auf:

- Sicherheit: Kommunikation über grosse Entfernung. Die Rechner gehöhren nicht dem selben ISP an.
- Übertragung der Nachrichten bei Stausituation.
- Kommunikation zwischen BB und Linux-Router, die nicht dem selben ISP angehöhren. Zudem grosser Aufwand und Verzögerung wegen der Kommunikation zwischen den entsprechenden ISPs.

Die beiden nächsten Ansätze, welche die Path- und Resv-Nachricht verwenden, haben den grossen Nachteil, dass sie auf die Backbone-Router einwirken müssen, bevor diese überhaupt eine Reservierung tätigen. Bei der Variante mit Path geht es sogar so weit, dass der Router konfiguriert wird, bevor eine eindeutige Anfrage an den Router gerichtet wird. Dies widerspricht eigentlich der Idee hinter RSVP. Der Empfänger ist ja nicht gezwungen die Reservierung wirklich zu machen. Bei der Resv-Nachricht muss wie beim ResvErr-Ansatz ein Linux-Router reagieren, welcher nicht zum Router-Paar gehört.
Beim Ansatz mit der maximalen Reservierung für RSVP-Flows wird dem Backbone-Routern eine maximale Schwelle vorgegeben, bis zu der Reservierung erlaubt sind. Geht eine Reservierung darüber hinaus, muss er diese ablehnen. Der Linux-Router übernimmt nach Rückfrage beim BB die Admission Control. Ausserdem informiert er den BB über die Freigabe von Bandbreite. Beim zweiten Ansatz verarbeitet der Backbone-Router keine RSVP-Nachrichten. Der Linux-Router ist alleine zuständig für die Umsetzung der Anfrage und überlässt dem Bandwidth-Broker deren Umsetzung in den Backbone-Routern. Im zweiten Fall sollte der Linux-Router über eine Flusskontrolle verfügen, um die gewährten Dienstgüten kontrollieren zu können.
Wenn man eine Reihenfolge der verschiedenen Ansätze geben müsste, so würde die beiden Ansätze mit entweder fest vordefinierten Schwelle für RSVP oder ganz ausgeschaltetem RSVP die ersten beiden Plätze belegen, während der Ansatz mit der ResvErr-Nachricht ganz am Ende anzutreffen ist. Der Entscheid, welcher der beiden besten Ansätze der bessere ist, ist dann eine Frage der Ansicht. Soll der Backbone-Router die Flüsse selber umsetzen oder soll dies der

Bandwidth-Broker gewährleisten.

Da vorgesehen war, die Datenströme zusammenzufassen, ist die zweite Variante mit ausgeschaltenem RSVP in den Backbone-Routern besser. Dadurch liegt die Kontrolle über die Datenströme bei Komponenten, die von der RVS-Gruppe selber geschrieben oder angepasst wurden. Zudem kann man den Eintritt- und Austrittspunkt genau definieren und trifft nicht auf Probleme wenn in einem weiteren Ausbau des Testnetzes weiter Backbone-Router hinzugefühgt werden.

Kapitel 6

Tools

In diesem Kapitel sollen nun einige Programme vorgestellt werden, welche im Zusammenhang mit dieser Diplomarbeit entstanden sind. Dabei ist zu erwähnen, dass die nötige Erweiterung des RSVP-Daemons mit der tatkräftigen Mithilfe von F. Baumgarter und der verwendete Bandwidth-Broker von I. Kahlil zur Verfügung gestellt wurde. Zusätzlich wird auf tc, von A. Kuznetsov, eingagange, das zur Steuerung der Traffic-Control des Linux-Kernels verwendet wird. Die einzelnen Tools werden nach dem gleichen Schema beschrieben. Zuerst folgt eine kurze Beschreibung der Funktionalität. Anschliessend werden die Schnittstellen definiert.

6.1 RSVP DiffServ Gateway (RDG)

Der RDG ist ein erweiterter RSVP-Daemon, welcher ankommende Reservierungsanfragen an den Bandwidth-Broker (BB) weiterleitet. Je nach Entscheid des BB setzt er dann eine Reservierung auf oder nicht. Zur Kommunikation mit dem BB ist die folgende Schnittstelle definiert.

6.1.1 Definition der Schnittstelle

Zur Kommunikation wird eine TCP-Verbindung aufrecht erhalten. Dabei werden die folgenden Kommandos gebraucht:

hereis: Mit diesem Befehl wird eine Verbindung zum BB hergestellt, ein Passwort übergeben, um den RDG zu identifizieren und eine minimale Sicherheit zu gewährleisten. Die folgenden Parameter sind erlaubt:

- **-h:** Name des Hostes, auf dem der RDG läuft.
- **-p:** Passwort, um in den BB einzuloggen.

Als Resultat sind die folgenden Werte definiert:

- **ok:** Erfolgreicher Verbindungsaufbau.
- **no:** Zugang verweigert.
- **error:** Fehler, eventuell einen Parameter vergessen.

newflow: Mit newflow wird versucht, Bandbreite zu reservieren. Dieser Befehl wird gesendet, wenn der RDG eine Resv-Nachricht erhält. Erhält der BB eine Anfrage für eine bereits existierende Reservierung, so muss diese alte Reservierung durch die neue ersetzt werden. Die folgenden Parameter sind erlaubt:

- **-d:** Die Empfängeradresse des Datenstroms.
- **-m:** Der Port des Empfängers.
- **-s:** Die Senderadresse des Datenstroms.
- **-n:** Der Port des Senders.
- **-b:** Die Bandbreite in Bytes pro Sekunden.

Als Resultat sind die folgenden Werte definiert.

- **ok:** Die Reservierung ist möglich und wurde in der Datenbank gespeichert.
- **no:** Die verlangte Bandbreite ist nicht verfügbar.
- **error:** Fehler, eventuell einen Parameter vergessen.

delflow: Mit delflow wird der Eintrag in der Datenbank gelöscht und die reservierte Bandbreite freigegeben. Die folgenden Parameter sind erlaubt:

- **-d:** Die Empfängeradresse des Datenstroms.
- **-m:** Der Port des Empfängers.
- **-s:** Die Senderadresse des Datenstroms.
- **-n:** Der Port des Senders.
- **-e:** Notfallmässiges Löschen der Reservierung. Tritt auf, wenn der RDG nach einer Erlaubnis des BB trotzdem nicht in der Lage ist, die Reservierung zu machen. Dies bedeutet ein Synchronistionsproblemen zwischen TC aud RDG und dem BB und sollte eigentlich nie auftreten..

Als Resultat sind die folgenden Werte definiert.

- **ok:** Die Reservierung wurde gelöscht.
- **no:** Die Reservierung konnte nicht entfernt werden.
- **error:** Fehler, eventuell einen Parameter vergessen.

modflow: Mit modflow wird die Bandbreite einer bestehenden Reservierung angepasst. Die folgenden Parameter sind erlaubt:

- **-d:** Die Empfängeradresse des Datenstroms.
- **-m:** Der Port des Empfängers.
- **-s:** Die Senderadresse des Datenstroms.
- **-n:** Der Port des Senders.
- **-b:** Die Bandbreite in Bytes pro Sekunden.

Als Resultat sind die folgenden Werte definiert.

- **ok:** Die Reservierung ist möglich und wurde in der Datenbank gespeichert.
- **no:** Die verlangte Bandbreite ist nicht verfügbar.
- **error:** Fehler, eventuell einen Parameter vergessen.

6.2 RSVP-Bandwidth-Broker

Der Bandwidth-Broker dient dazu auf Anfragen des RDG zu reagieren. Er muss entscheiden, ob eine Reservierung erlaubt ist oder ob diese abgelehnt werden muss. Wenn eine Reservierung genehmigt wird, so muss der BB die Backbone-Router entsprechend konfigurieren. Die Steuerung der Backbone-Routers selber erfolgt durch die Generierung von Skriptdateien, die durch die Backbone-Router selbständig geladen und ausgeführt werden.
Dies führte dazu, dass die ursprünglich vorgesehene Schnittstelle (Abschnitt 6.1.1)angepasst werden musste, was mit dem RDG-BL (RDG-B-Layer) geschah. Die Kommunikation zwischen dem Bandwidth-Broker und dem RDG-BL erfolgt mit zwei getrennten Datenübertragungen, dabei wird bei der ersten die Anfrage gestellt und bei der zweiten die entsprechende Antwort zurückgeliefert. Die entsprechende Schnittstelle zum RDG wurde beibehalten. In Abb. 6.1 ist das Zusammenarbeiten der Komponenten schematisch Dargestellt.

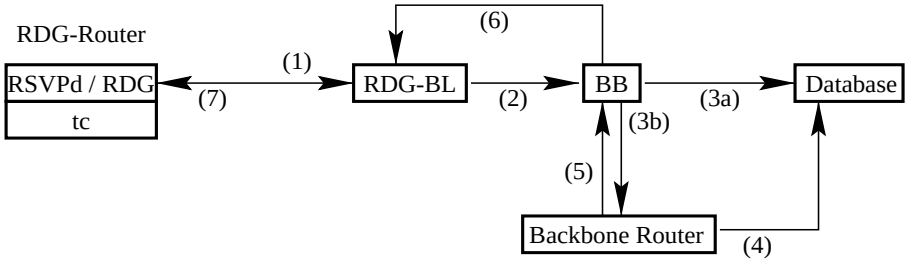

Abbildung 6.1: Das Zusammenspiel der Komponenten für das Aufsetzten einer Reservierung. Die Zahlen geben die Reihenfolge der Anfragen an.

Auf die entsprechende Schnittstelle zu den Backbone-Routern wird hier nicht weiter eingegangen.
Um eine Reservierung zu erhalten, kreiert der BB im Backbone-Router einen IP-Tunnel, über welchen die Daten gesendet werden. Zur Zeit ist eine Aggegration mehrerer Flüsse auf einen Tunnel nicht vorgesehen. Diese Aufgabe des BB ist für eine spätere Version geplant.
Die beiden nächsten Tools wurden entwickelt, weil für Testzwecke geeignete Programme unter Linux nicht zur Verfügung standen. Bevor diese entwickelt

wurden, mussten die nötigen Befehle zur Reservierung per Hand im Debug-Modus des RSVPd eingegeben werden. Diese Befehle waren kryptisch und mussten sowohl auf Empfänger-, als auch auf Senderseite eingegeben werden. Dies legte das Schreiben eines Hilfsprogrammes nahe.

6.3 RSVPServer

RSVPServer ist ein Tool, das eine Kommunikation zwischen einem Programm und dem RSVP-Daemon sicherstellt. Es ermöglicht das Aufsetzen einer RSVP-Reservierung von jedem beliebigen Punkt aus, sofern auf den betroffenen Maschinen auch der *RSVPServer* läuft. Zudem müssen nicht beide bei der Verbindung beteiligten Programme RSVP unterstützen, da die für die Reservierung notwendige Signalisierung von einem Punkt aus geschehen kann. Es ist sogar möglich die Reservierung zentral von einer externen, nicht beteiligten Maschine aus zu machen. Um dies zu erläutern betrachten wir das Beispiele in Abb. 6.2.

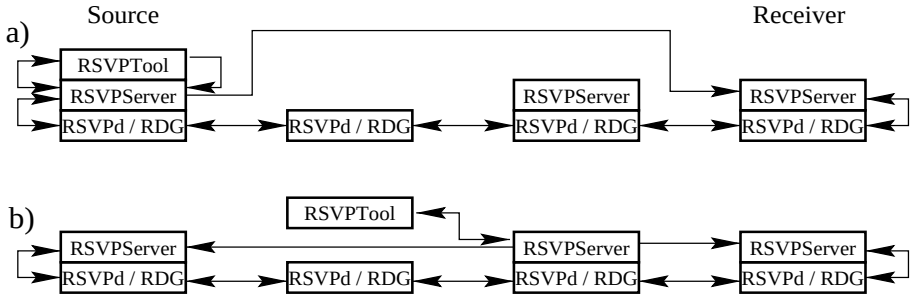

Abbildung 6.2: Das Zusammenspiel der Komponenten für das Aufsetzten einer Reservierung mit dem *RSVPServer*.

In a) ist der Sender, der die Reservierung aufsetzt. In b) ist dies ein Benutzer, der nicht direkt in die Datenkommunikation eingebunden ist. In beiden Fällen sendet der Benützer eine Anfrage an einen RSVPServer, der dann die Reservierung initialisiert, indem er mit den RSVPServern an den beiden Ende der Verbindung Kontakt aufnimmt.

6.3.1 Definition der Schnittstelle

Die Kommunikation zum *RSVPServer* geschieht über eine bidirektionale Verbindung, die auf normalen Unix-Sockets aufbaut. Dabei werden verschiedene Kommandos verwendet, die in drei Gruppen unterteilt werden können. Zuerst werden die Befehle erklärt, die ein Benutzer braucht, um beim *RSVPServer* eine Reservierung zu verlangen. In der Zweiten Gruppe sind die Befehle vereint, die auf Anfrage an den *RSVPServer* reagieren. In der letzten Gruppe sind die Befehle eingeteilt, die nur zwischen den *RSVPServern* erlaubt sind.

Externe Kommandos

Externe Kommandos ermöglichen es einem Benutzer eine entsprechende Reservierung aufzusetzen. Diese Befehle werden auch zwischen den *RSVPServern* verwendet.

resv: Dieser Befehl dient dazu, eine RSVP-Reservierung aufzusetzen. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.
- **-b:** Die gewünschte Bandbreite in KByte.
- **-u:** Die Traffic-Art der Daten:
  - TCP
  - UDP
- **-q:** Die gewünschte Dienstgüte:
  - G entspricht dem Guaranteed Service
  - CL entspricht dem Controlled Load Service.

free: Dieser Befehl dient dazu, eine RSVP-Reservierung aufzuheben. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.

clear: Dieser Befehl dient dazu, eine RSVP-Reservierung komplett zu entfernen. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.

check: Dieser Befehl dient dazu, alle vorhanden RSVP-Reservierungen anzuzeigen, die zu den angegebenen Parametern passt. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.

close: Dieser Befehl dient dazu, eine Verbindung zu beenden. Er hat keine Parameter.

iam: Dient der Identifizierung desjenigen der die Verbindung aufgebaut hat. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-s:** Der Adresse des Initiator.
- **-p:** Passwort.
- **-i:** Eine ID um die Art des Programmes, das die Verbindung macht, anzugeben.
  - 1: *RSVPServer*.
  - 2: *RSVPTool*.

Antworten an Clients

Wenn eine Anfrage an den RSVPServer gestellt wurde, gibt es zwei mögliche Antworten. Entweder es sind Reservierungen bekannt, oder aber es gibt keine passende Reservierung.

checkresult: Dieser Befehl gibt eine passende Reservierung zurück. Der Befehl wird so oft wiederholt, bis alle passenden Reservierungen übermittelt wurden. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.
- **-b:** Die reservierte Bandbreite in KByte.
- **-u:** Die Traffic-Art der Daten:
  - TCP
  - UDP
- **-q:** Die gewünschte Dienstgüte:

  - G entspricht dem Guaranteed Service
  - CL entspricht dem Controlled Load Service.

no: Gibt an, dass keine passende Reservierung vorhanden ist.

all: Gibt an, dass alle passenden Reservierungen übermittelt wurden.

Interne Kommandos

Interne Kommandos werden gebraucht um die Kommunikation zwischen den *RSVPServern* zu ermöglichen. Dabei geht es darum die entsprechenden RSVP-Signalisierungen sicherzustellen.

initpath: Dient dazu den Empfänger aufzufordern, eine Path-Nachricht zu erzeugen. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.
- **-b:** Die gewünschte Bandbreite in Byte.
- **-u:** Die Traffic-Art der Daten:
  - TCP
  - UDP
- **-q:** Die gewünschte Dienstgüte:
  - G entspricht dem Guaranteed Service
  - CL entspricht dem Controlled Load Service.

initresv: Dient dazu den Empfänger aufzufordern, eine Resv-Nachricht zu erzeugen. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.
- **-b:** Die gewünschte Bandbreite in Byte.
- **-u:** Die Traffic-Art der Daten:
  - TCP
  - UDP
- **-q:** Die gewünschte Dienstgüte:
  - G entspricht dem Guaranteed Service
  - CL entspricht dem Controlled Load Service.

clearpath: Dient dazu den Empfänger aufzufordern, eine PathTear-Nachricht zu erzeugen. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.
- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.

clearresv: Dient dazu den Empfänger aufzufordern, eine ResvTear-Nachricht zu erzeugen. Die folgenden Parameter sind erlaubt und/oder erforderlich:

- **-d:** Der Adresse des Empfängers des Datenstroms.
- **-e:** Die Portnummer des Empfängers.

- **-s:** Die Adresse des Senders des Datenstroms.
- **-t:** Die Portnummer des Senders.

6.4 RSVPTool

RSVPTool ist ein grafisches Interface, mit dem man über den RSVPServer eine RSVP-Reservierung aufsetzten kann. Dieses Interface wurde in Java implementiert und benötigt zur Ausführung eine Javainterpreter.

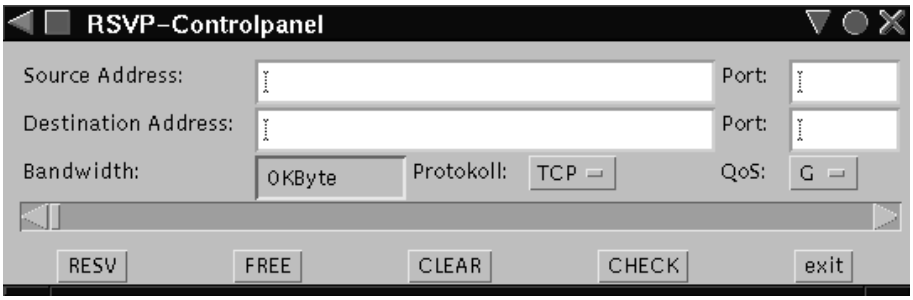

Abbildung 6.3: Screenshot von RSVPTool.

Wie man in Abb. 6.3 sieht, ist die Bedienung recht einfach. Die Eingabefelder haben folgende Bedeutung:

- **Source Address** und **Port**: Die Adresse und Portnummer des Sender der Daten.
- **Destination Address** und **Port**: Die Adresse und Portnummer des Empfänger der Daten.
- **Bandwidth**: Zeigt die gewählte Bandbreite an, die über die Scrollbar eingestellt werden kann.
- **Protokol**: Gibt die verwendete Protokollart an. Zur Auswahl stehen TCP und UDP.
- **QoS**: Dient zur Wahl der Dienstgüte. Zur Wahl stehen Guaranteed Service (**G**) und Controlled Load (**CL**).
- **RESV**: Initialisierung oder Anpassung einer Reservierung.

- **FREE**: Gibt eine reservierte Bandbreite frei. Die RSVP-Session wird aber nicht beendet, d.h. der Path-State bleibt erhalten.
- **CLEAR**: Gibt eine reservierte Bandbreite frei. Dazu wird auch die RSVP-Session komplett beendet.
- **CHECK**: Es wird beim *RSVPServer* angefragt, welche Reservierungen existieren, die zu den übermittelten Daten passen. Als Ergebnis wird eine Fenster geöffnet, das alle passenden Reservierungen anzeigt. Wird eine davon angewählt, so wird diese automatisch übernommen.
- **EXIT**: Beendet das Programm.

Es existiert auch eine Version ohne die Schalter RESV, FREE und CLEAR. Dort wird die Reservierung direkt über die Scrollbar für die Einstellung der Bandbreite gemacht. Dabei wird die Reservierung laufend dem aktuellen Stand der Scrollbar angepasst, was dazu führen kann, dass die Reservierung sehr häufig angepasst werden muss. Dies kann zu Interaktionsproblemen mit den Backbone-Routern führen, wegen der relativ langsamen Umsetzung der Reservierungen.

6.5 tc– Traffic-Control-Steuerung

Mit *tc* lässt sich die Traffic-Control des Linux-Kernels steuern. Leider existiert dazu nur vereinzelte Drafts mit Information über tc, jedoch keine wirkliche Beschreibung von A. Kuznetsov. Hilfreiche Informationen finden sich in den zwei Arbeiten von W. Almesberger [Alm98] und [ASK99].
Betrachten wir die Aufteilung einer CBQ (wie in Abb. 6.4 gezeigt), so fällt auf, dass diese aus drei Grundelementen aufgebaut wird.

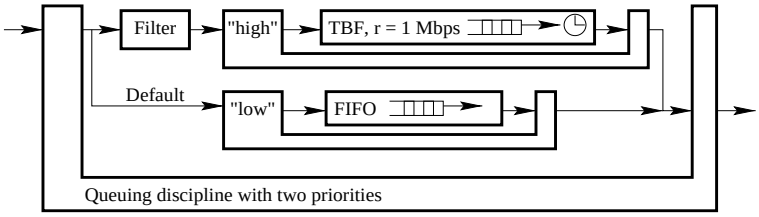

Abbildung 6.4: Der Aufbau einer CBQ, mit ihren seine drei Grundelementen.

Für diese Grundelemente gibt es in tc jeweils einen Schalter, der diese aktiviert. Mit dem Parameter help erhält man eine genauer Beschreibung der Syntax, jedoch ist eine genaue Beschreibung aller Parameter nicht verfügbar. Betrachten wir nun, was die einzelnen Befehle machen.

- **qdisc**: Mit tc qdisc add ... kann man eine neue queueing discipline erzeugen. Die Basis einer CBQ ist immer eine queueing discipline. In dieser kann nun eine Klasse eingebettet werden.
- **class**: Mit tc class add ... kann man eine neue Klasse zu einer queuing Disziplin hinzufügen. Eine queueing discipline kann mehrere Klassen enthalten, wobei eine Klasse wieder eine eigene queueing discipline enthalten kann.
- **filter**: Mit tc filter add ... wird ein Filter hinzugefügt. Ein Filter dient dazu, die Pakete verschiedenen Klassen zuzuweisen. Mehrere Filter können auch auf ein und dieselbe Klasse zeigen.

Zu jedem dieser Befehle gibts es verschiedene Schalter.

- **add**: Damit wird ein neues Objekt hinzugefügt.
- **del**: Damit wird eine existierende Komponente entfernt. Dies geht nur, wenn in der Hirarchie keine Komponente davon abhängt.
- **change**: Damit wird eine bestehende Komponente abgeändert.
- **help**: Gibt die Syntax der verschiedenen Befehle an. Muss wie folgt gebraucht werden: *tc class help*.
- **show**: Gibt den Zustand aller auf die Parameter passenden Komponenten an. Gibt man den Schalter -s an, wird die Beschreibung noch ausführlicher. Mit dem Befehl *tc -s class show dev eth0* erhält man eine Auflistung aller Klassen, die für das Interface eth0 definiert wurden.

Betrachten wir nun die Initialisierung einer TC für das Interface. Dieses Beispiel entspricht der TC die in Beispiel im Abschnitt 7 für das Versuchsnetz gebraucht wurde.

```
#!\bin\sh

tc qdisc add root dev eth0 handle 1: cbq \
bandwidth 100Mbit cell 8 avpkt 1000 mpu 64
```

Auf dem Interface eth0 wird eine queueing discipline eingerichtet, die die volle Bandbreite von 100 Mbit verwalten soll.

```
# root class, for full bandwidth

tc class add dev eth0 parent 1:0 classid :1 est 1sec 8sec cbq \
bandwidth 100Mbit rate 100Mbit bounded isolated allot 1514 cell 8 \
weight 10Mbit prio 8 maxburst 50 avpkt 1500
```

Es wird eine Basis-Klasse eingerichtet, um die ganze Bandbreite von 100Mbit zu verwalten. Die Klasse hat die ID 1:1.

```
# classes for default traffic

tc class add dev eth0 parent 1:1 classid :2 est 1sec 8sec cbq \
bandwidth 100Mbit rate 800Kbit bounded allot 1514 weight 100Kbit \
prio 6 maxburst 50 avpkt 1000 split 1:0 defmap ff3d
```

Es wird eine Klasse eingerichtet, die für normalen Datenverkehr vorgesehen ist. Sie ist eine Unterklasse der Klasse 1:1. In dieser Klasse wird sowohl der UDP, als auch der TCP-Verkehr umgeleitet. Mit dem Parameter *rate 800Kbit* wird die verfügbare Bandbreite auf 800 Kbit beschränkt. Durch den Parameter *bounded* wird verhindert, dass die Klasse auf nicht vergebene Bandbreite zurückgreift. Mit *maxburst 50* wird die Queue auf 50 Pakete beschränkt.

```
# class, rsvp need this
```

```
tc class add dev eth0 parent 1:1 classid 1:7FFE cbq \
rate 20Mbit bandwidth 100Mbit bounded isolated allot 1514b avpkt 1500 \
weight 500Kbit prio 8 maxburst 20 cell 8

# Reclassified realtime traffic

tc class add dev eth0 parent 1:7FFE classid 1:7FFF  est 4sec 32sec cbq \
rate 15Mbit bandwidth 100Mbit allot 1514b avpkt 1000 weight 10Kbit \
prio 6 maxburst 10 cell 8 split 1:7FFE defmap ffff
```

Diese zwei Klassen werden von RSVPd benötigt. Wobei die zweite Klasse eine Unterklasse der ersten ist, die wiederum von der Klasse 1:1 abgeleitet wurde. Auch hier wird die zur Verfügung stehende Bandbreite durch den Parameter bounded beschränkt. Mit dem Parameter *isolated* wird zudem verhindert, dass andere Klassen, die dem RSVPd zugeteilte Bandbreite benutzen können. Dies bedeutet, dass diese 20Mbit von keiner anderen Queue gebraucht werden kann. Mit dem Script kann man folglich TC starten, um eine maximale RSVP-Reservierung von 20Mbit zu machen und den normalen Datenverkehr auf weniger als 1Mbit zu beschränken. Für eine faires System mit normalem Datenverkehr würde man die Beschränkungen, die mit bounded und isolated erzeugt wurden, natürlich entfernen.

6.5.1 Probleme mit RSVPd und tc

Da die verfügbare Dokumentation zu tc nicht komplett ist, treten bei der Benutzung automatisch Probleme auf. Deshalb werden hier einige Effekte vorgestellt, die bei den Tests auftraten.

- Wie oben angedeutet kann man mit bounded und isolated die Bandbreite sehr genau beschränken. Jedoch traten bei Tests mit ttcp einige Probleme mit UDP-Traffic auf. Befindet sich der Sender auf der gleichen Maschine

wie das Queuing-System, so wird der UDP-Strom überproportional ausgebremst. Während mit TCP-Daten dieses Problem nicht auftritt. Es ist aus nicht nachvollziehbaren Gründen sogar so, dass eine TCP-Verbindung einen höheren Datendurchsatz hat als UDP.

- Wenn man im Ethernet arbeitet, sollte man sicher gehen, dass die erzeugten Datenpakete die MTU von 1500 Byte nicht überschreiten. Ist dies der Fall, so unterteilt das IP-Subsystem des Linux-Router diese Pakete und sendet die Fragmente an den Empfänger. Leider sind in diesen Fragmenten keine Portnummern enthalten. Wodurch TC die Pakete nicht mehr korrekt klassifizieren und den entsprechenden Queues zuordnen kann. Dies ist kein generelles Problem von Linux, sondern tritt auch bei anderen Routern auf.

  Will man eine Reservierung mit einer MTU grösser als 1500 Byte machen, so lehnt dies der RSVPd, genau aus diesem Grund ab.

  Bei einem DiffServ-Router würde dieses Problem nicht bestehen, da das TOS-Field im IP-Header zum Routing verwendet wird und dieses in den Fragmenten enthalten ist.

Kapitel 7

Realisierung

Mit den in den vorangehenden Abschnitten vorgestellten Tools ist es nun möglich, eine Bandbreitenreservation zwischen zwei Endsystemen aufzubauen. Betrachten wir dazu Abb. 7.1.

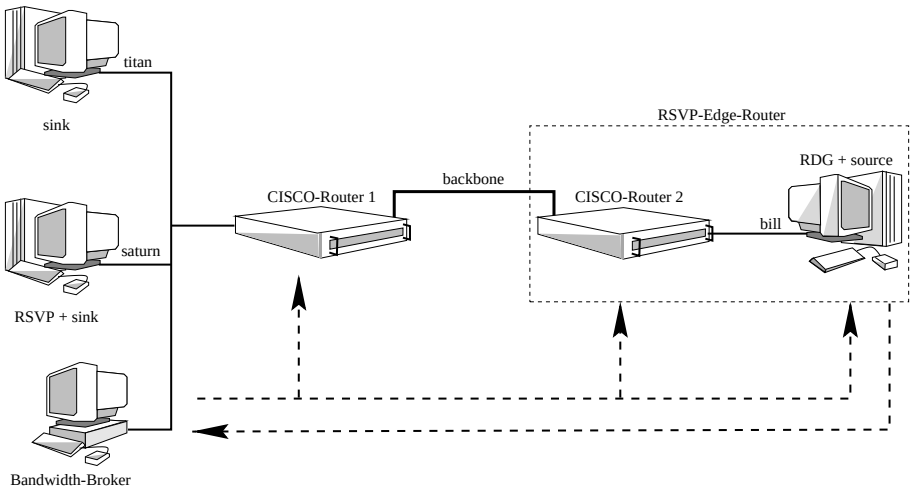

Abbildung 7.1: Netzwerktopologie, die bei mehreren Demonstrationen gebraucht wurde.

Zu Vergleichszwecken sendet die Source *bill* identische Datenströme an die zwei Empfänger *saturn* und *titan*. Dabei kann über RSVP zwischen *bill* (RDG) und *saturn* (RSVPd) eine Reservierung aufgesetzt werden. Hierzu kommuniziert der RDG über den RDG-BL mit dem Bandwidth-Broker. Dieser ist dann für die Umsetzung der Reservierung zwischen den Backbone-Routern zuständig.

Abb. 7.2 zeigt die Kommunikation zwischen den einzelnen Komponenten zum Aufsetzen der Rerservierung.

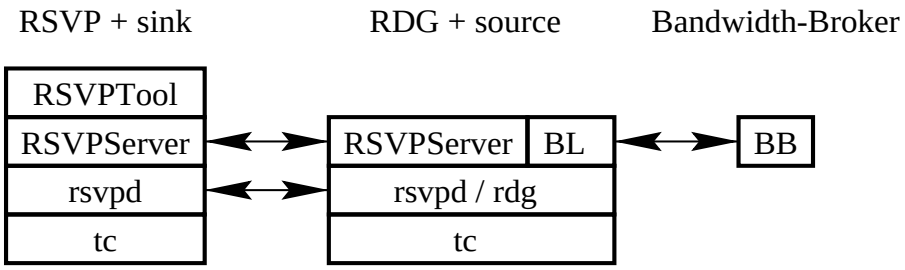

Abbildung 7.2: Kommunuikation zwischen den einzelnen Modulen/Schichten.

Wir sehen die Kommunikation zwischen den einzelnen Schichten. Der Erzeuger der Daten ist dabei nicht aufgeführt, weil er in diesem Fall nicht für die Reservierung der Bandbreite zuständig ist, da dies mit Hilfe des RSVPTools erfolgt. Wenn der Sender oder der Empfänger eine Reservierung selbstständig aufsetzen könnte, so würde dieser an die Stelle des RSVPtools treten. Wenn sogar beide eine RSVP-Unterstützung hätten, könnte sogar der RSVPServer entfallen.
Zwischen den RSVPServern muss eine direkte Kommunikation bestehen, um die gewünschte Reservierung zu initialisieren, während die Kommunikation zwischen dem RSVPd und dem RDG dazu dient, die Reservierung aufzusetzen. In den einzelnen Systemen erfolgt die Kommunikation von Schicht zu Schicht, wobei der RSVPd, bzw. der RDG die Traffic-Control (tc) der Linux-PCs auffordern, die Reservierung zu machen.
Mit dem oben vorgestellten Testnetz ist es möglich, verschiedenen Datenströmen, verschiedene Bandbreiten zuzuteilen. In der folgenden Tabelle 7.1 sind die Ergebnisse eines solchen Versuchs abgebildet.
Beim Versuch wurde *ttcp* als Quelle verwendet. Zusätzlich wurde die verfügbare Bandbreite für normalen TCP- und UDP-Datenverkehr in den Linux-PC's auf 800 Kbit beschränkt. Wie dies erreicht wurde ist in Anhang 6.5 beschrieben.
Betrachtet man das Ergebnis, so fällt auf, dass sobald ein Datenfluss eine Reservierung erhält, der zweite Datenfluss erneut 800 Kbit erreicht. Dies ist aber normal, da der Datenfluss mit der 2000 Kbit Reservierung nicht mehr über die selbe tc-Klasse gesendet wird.

Testlauf	Empfänger	Bandbreite	Beschreibung
1.A	titan	779.34 Kbit/s	Keine Reservierung,
1.B	saturn	797.55 Kbit/s	800 Kbit/s Beschränkung
2	titan saturn	381.65 Kbit/s 396.09 Kbit/s	Keine Reservierung, 800 Kbit/s Beschränkung, beide Flüsse parallel
3	titan saturn	788.44 Kbit/s 1819.43 Kbit/s	2000 Kbit/s Reservierung für Datenfluss bill → saturn, beide Flüsse parallel.

Tabelle 7.1: Ergebnisse eines Tests mit ttcp.

Einen effektvolleren Vergleich erhält man, wenn man zwei Videoströme sendet und die erhaltenen Videoqualität auf beiden Endsystemen betrachtet. Auf dem Host mit der Reservierung erhält man eine Bildrate von knapp 10 Bilder pro Sekunde, was einer flüssigen Wiedergabe entspricht. Hingegegen ist auf dem zweiten Host die Qualität nach wie vor schlecht und es werden 2-3 Bilder pro Sekunde empfangen. Auf eine weitere quantitative Analyse der Videoqualität soll hier allerdings verzichtet und auf die Demonstrationen verwiesen werden. Mit dem derzeitigen Stand der Arbeit steht ein lauffähiges System zur Aufsetzung von Reservierungen zur Verfügung. Im nächsten Abschnitt erfolgt eine Zusammenfassung, sowie ein Ausblick auf weiter Entwicklungen.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Mit Abschluss dieser Arbeit steht ein Ansammlung von Programmen zur Verfügung, die es erlauben eine Reservierung zwischen zwei End-Systemen über ein Backbone-Netz hinweg zu etablieren. Im Speziellen wurden folgende Ziele erreicht:

- Es existiert ein lauffähiges System, um zwischen zwei Endsystemen eine Reservierung aufsetzen kann.
- Auf Linux-Seite des Testnetzes kann mit Hilfe der Traffic-Control im Linux-Kernel eine QoS garantiert werden.
- Der RDG bildet für sich ein abgeschlossenes System.
- Eine abgeschlossenen Umgebung mit dem RSVPServer und dem RSVP-Tool, womit man von jedem Punkt aus eine Reservierung initialisieren kann.
- Mit der Verwendung von speziellen Tunnels ist es möglich die Daten im Backbone-Bereich verschlüsselt zu übertragen.

Die folgenden Probleme sind bis jetzt offen geblieben und sollten in einer späteren Phase behoben werden:

- Die Aggregration von mehreren Datenströmen auf einen Tunnel wurde verschoben, da zur Zeit in einer anderen Diplomarbeit die Architektur für einen neuen Bandwidth-Broker umgesetzt wird.
- Die Datenübertragung erfolgt im Klartext, was ein Sicherheitsproblem sein kann Dies ist jedoch für einen Prototypen ausreichend und erleichtert die Fehlersuche.
- Die mangelnde DiffServ-Unterstützung führt dazu, dass die RSVP-Datenströme mittels IP-Tunnels übers Backbone übertagen wurden.

8.2 Ausblick

Wie oben beschrieben ist das bestehende System in der Lage eine Reservierung aufzusetzen.
Im folgenden soll einige Punkte aufgezeigt werden, in welche Richtung diese Arbeit fortgeführt werden könnte:

- Mit einer DiffServ-Implementation die in Zusammenarbeit der RVS-Arbeitsgruppe und NEC-Heidelberg entstanden ist, könnte man die Aufteilung auf einen Linux-Router und einen Backbone-Router umgehen und alles auf einem einzigen Rechner machen.
- RSVP unterstützt bei der Signalisierung auch Policy-Objekte. Es liegt nahe diese Eigenschaft der Protokolles auszunützten und für die Admission-Control zu nutzen. Dabei sollte man an eine Unterstützung von COPS denken ([BCD+98]).
- Die Diplomarbeit stand im Zusammenhang mit dem CATI-Projekt. Dabei müssen verschiedene Komponenten miteinander kombiniert werrden. Z.B. können charging Komponenten zum RDG hinzugefügt werden, welche dann ermöglichen die einzelnen RSVP-Flows einzeln abzurechnen.

- Die Sicherheit der Tools selber sollte erhöht werden, indem man für die Datenübertrageung Verschlüsselung verwendet.
- Der verfügbare Bandwidth-Broker sollte so ausgebaut werden, dass er eine Aggregration von mehreren RSVP-Flüssen in einen einzigen Tunnel erlauben soll. Zudem sollte der BB fähig sein, einen neuen Tunnel über mehrere ISPs hinweg aufzusetzen und die gewünschte Dienstgüte zuzusichern.
- Die Backbone-Router sollten in der Lage sein, die Daten mit den DiffServ-Mechanismen zu transportieren, so dass man für die Umsetzung der RSVP-Flüsse nicht mehr nur auf Tunnels angewiesen ist.

Anhang A

Traffic-Control unter Linux

In diesem Abschnitt wird kurz erklärt, was im Linux-Kernel vorhanden sein muss, um Traffic-Control im Allgemeinen und RSVP im Speziellen zu unterstützen. Leider sind die Informationen zur TC-Unterstützung spärlich und die Beschreibungen zum Kernel sind teilweise lückenhaft.
Seit Version 2.1.105 ist die Unterstützung für QoS im Linuxkernel vorhanden. Für die Integration und die Test wurde der stabile 2.2.12 Kernel verwendet. Da die Traffic-Control-Mechanismen standartmässig ausgeschaltet sind, müssen sie explizit in den Kernel compiliert werden. Für die Konfiguration siehe das Kernel-Konfigurations-Menu in Abb. A.1.
Für unsere Zwecke brauchen wir mindestens eine Class-Based-Queueing Disziplin. Da die Traffic-Control Unterstützung im Testnetzwerk nicht immer gebraucht wurde, wurde sie nur als Modul in den Kernel gebunden und kann somit dynamisch bei Bedarf geladen werden. Um die Unterstützung von RSVP zu erhalten muss auch der *QoS-Support* und die Option *Packet Classifier API* aktiviert werden. Zum Schluss muss man noch die RSVP-Unterstützung mindestens als Modul in den Kernel integrieren.

Abbildung A.1: Die vorhanden QoS-Unterstützung im Linux-Kernel.

Anhang B

RSVP Definitionen

B.1 Einige funktionale Definitionen

- *Path*-Nachricht:
  <Path Message> ::= <Common Header> [<INTEGRITY>] <SESSION> <RSVP_HOP> <TIME_VALUES> [<POLICY_DATA> ...] [<sender descriptor>]
- *sender descriptor*:
  <sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC> [<ADSPEC>]
- *Resv*-Nachricht:
  <Resv Message> ::= <Common Header> [<INTEGRITY>] <SESSION> <RSVP_HOP> <TIME_VALUES> [<Resv_CONFIRM>] [<SCOPE>] [<POLICY_DATA> ...] <STYLE> <flow descriptor list>
- *flow descriptor*:
  <flow descriptor list> ::= <empty> | <flow descriptor list> <flow descriptor>

- *ResvErr*-Nachricht:
  <ResvErr Message> ::= <Common Header> [<INTEGRITY>] <SESSION> <RSVP_HOP> <ERROR_SPEC> [<SCOPE>] <STYLE> [<POLICY_DATA> ...] <error flow descriptor>
- *WF*-Style:
  <error flow descriptor> ::= <WF flow descriptor>
  <WF flow descriptor> ::= <FLOWSPEC>
- *FF*-Style:
  <error flow descriptor> ::= <FF flow descriptor>
  <FF flow descriptor> ::= [<FLOWSPEC>] <FILTER_SPEC>
- *SE*-Style:
  <error flow descriptor> ::= <SE flow descriptor>
  <SE flow descriptor> ::= <FLOWSPEC> <filter spec list>
  <filter spec list> ::= <FILTER_SPEC> | <filter spec list> <FILTER_SPEC>

B.2 RSVP Header (Common Header)

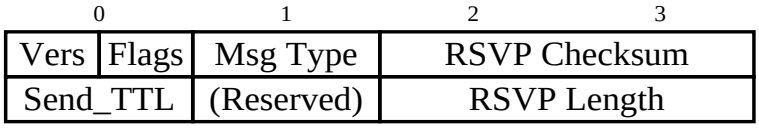

Abbildung B.1: Header, der für jede RSVP-Nachricht gleich ist.

Die Felder im Header (Abb. B.1) sind wie folgt definiert:

- *Vers* (4 bit): Versionsnummer des Protokolles.
- *Flags* (4 bit): Zur Zeit sind keine Flags definiert, aber 0x01 - 0x08 sind reserviert.
- *Message Type* (8 bit): Der Typ der Nachricht, wobei
  1. = Path,

2. = Resv,
3. = PathErr,
4. = ResvErr,
5. = PathTear,
6. = ResvTear und
7. = ResvConf ist.

- *RSVP Checksum* (16 bit): Einerkomplement der Einerkomplementsumme der Nachricht. Ist die Checksumme = 0, so wurde keine Checksumme übermittelt.
- *Send_TTL* (8 bits): Der IP TTL wert, mit der die Nachricht gesendet wurde.
- *RSVP Length* (16 bit): Die Länge der RSVP-Nachricht in Bytes, einschliesslich des Headers und den folgenden Objekten.

B.3 RSVP-Objekt Formate

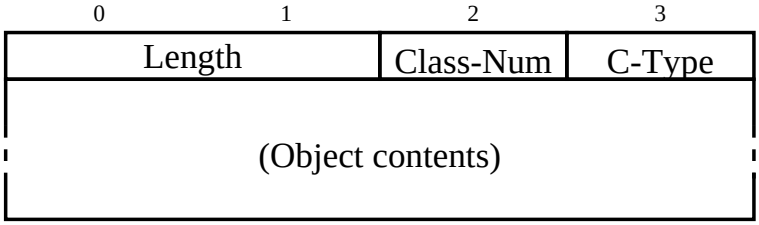

Abbildung B.2: Ein RSVP-Objekt.

Die Felder eines Objektes (Abb. B.2) sind wie folgt definiert:

- Lenght (16 bit): Gibt die totale Länge des Objektes in Byte an.
- Class-Num (8 bit): Gibt die Objekt-Klasse an. Die folgenden Klassen sind vorhanden:
  - **NULL:** Ein NULL-Objekt wird von jedem Empfänger ignoriert.

- **SESSION:** Enthält die IP Empfänger Adresse (DestAddress), die IP Protokoll ID und eine Art generalisierten Zielport, um die Sitzung zu definieren. *Muss in jeder RSVP-Nachricht enthalten sein.*
- **RSVP_HOP:** Enthält die IP-Adresse des RSVP-fähigen Knotens, der diese Nachricht gesendet hat, sowie die logical outgoing interface handle (LIH).
- **TIME_VALUES:** Enthält den Wert der refresh Periode R, die vom Erzeuger der Nachricht gebraucht wird.
- **STYLE:** Definiert die Reservierungsart, sowie spezifische Daten dazu, die nicht in *FLOWSPEC* oder *FILTER_SPEC* enthalten sind. Muss in jeder *Resv*-Nachricht enthalten sein.
- **FLOWSPEC:** Definiert die verlangte *QoS* in einer *Resv*-Nachricht.
- **FILTER_SPEC:** Definiert eine Untermenge von Sitzungspaketen, welche die verlangte *QoS* erhalten sollen.
- **SENDER_TEMPLATE:** Enthält die IP-Adresse des Senders und eventuell einige weiter demultiplex Daten, um den Sender zu identifizieren. Muss in der *Path*-Nachricht enthalten sein
- **SENDER_TSPEC:** Spezifiziert den Datenstrom des Senders. Muss in der *Path*-Nachricht enthalten sein.
- **ADSPEC:** Enthält *OPWA*-Daten in einer *Path*-Nachricht.
- **ERROR_SPEC:** Spezifiziert einen Fehler in *PathErr*-, *ResvErr*- oder *ResvConf*-Nachricht.
- **POLICY_DATA:** Enthält Daten, die es lokalen Kontrollmodulen ermöglicht zu entscheiden, ob eine Reservierung erlaubt ist. Kann in *Path*-, *Resv*-, *PathErr*- oder *ResvErr*-Nachrichten vorkommen.
- **INTEGRITY:** Enthält Daten zur Verifizierung dieser RSVP-Nachricht.
- **SCOPE:** Gibt eine explizite Liste von Host an, an welche die Nachrichten weitergeleitet werden sollten. Kann in *Resv*-, *ResvErr*- oder *ResvTear*-Nachrichten vorkommen.

- **RESV_CONFIRM:** Gibt die IP-Adresse des Empfängers an, der eine Bestätigung wünscht.

- *C-Type* (8 bit): Objekt Typ, nur mit Class-Nummer.

- *Object contents*: Der Inhalt des Feldes darf höchstens 65528 Bytes gross werden. Zudem ist die Grösse immer ein Vielfaches von 4 Bytes.

B.3.1 SESSION Objekt: Class = 1

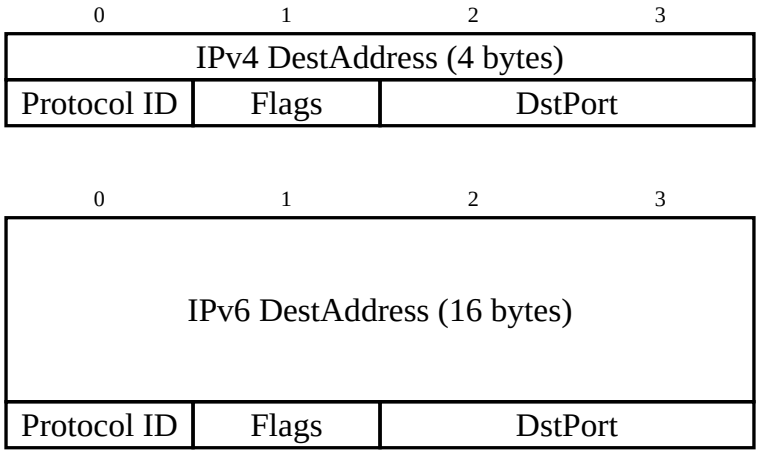

Abbildung B.3: SESSION Objekt für IPv4 und IPv6.

In Abb. B.3 sind die beiden möglichen SESSION Objekte für IPv4 (C-Type = 1), bzw. IPv6 (C-Type = 2) angegeben. Dabei sind die Felder wie folgt definiert:

- **DestAddress:** Adresse des Empfängers.

- **Protocol ID:** IP Protokoll Identifikator für den Datenfluss. Diese Feld darf nicht 0 sein.

- **Flags:**

  - 0x01 = E_Police: Wird gebraucht, um den Bereich festzustellen, wo traffic-policing durchgeführt werden kann. Wenn der Host nicht traffic-policing-fähig ist, so setzt er dieses Flag. Der erste Knoten der traffic-policing kann, schaltet dieses dann aus.

- **DstPort:** Den UDP/TCP Empfänger Port des Datenflusses. Null impliziert, dass kein spezifischer Port vorhanden ist.

B.3.2 RSVP_HOP Objekt: Class = 3

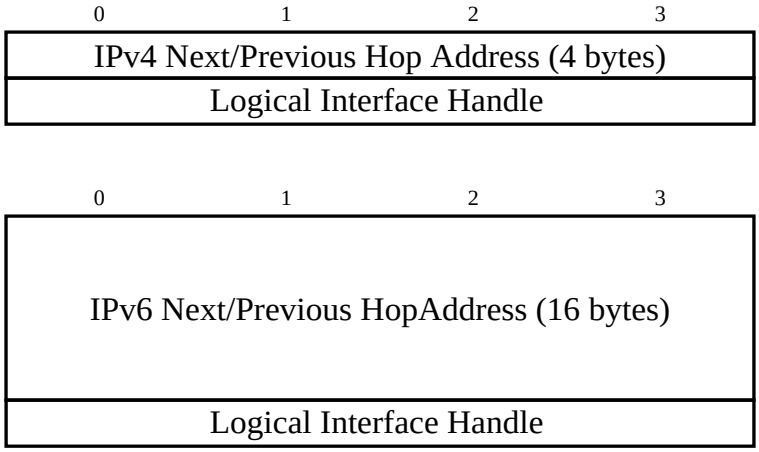

Abbildung B.4: RSVP_HOP Objekt für IPv4 und IPv6.

In Abb. B.4 sind die beiden möglichen RSVP_HOP Objekte für IPv4 (C-Type = 1,) bzw. IPv6 (C-Type = 2) angegeben. Dabei sind die Felder wie folgt definiert:

- **Next/Previous Hop Address:** IP-Adresse des Interfaces, über das der letzte RSVP-Hop diese Nachricht gesendet hat.
- **Logical Interface Handle:** Zum Bestimmen von logischen Interfaces (z.B. bei Tunnels).

B.3.3 TIME_VALUES Objekt: Class = 5

Abbildung B.5: TIME_VALUES Objekt.

In Abb. B.5 ist das TIME_VALUES Objekt (C-Type = 1). Dabei ist das Feld wie folgt definiert:

- **Refresh Period:** Refresh Periode, die verwendet wird, um eine Reservierung zu aktualisieren (in Millisekunden).

B.3.4 ERROR_SPEC Objekt: Class = 6

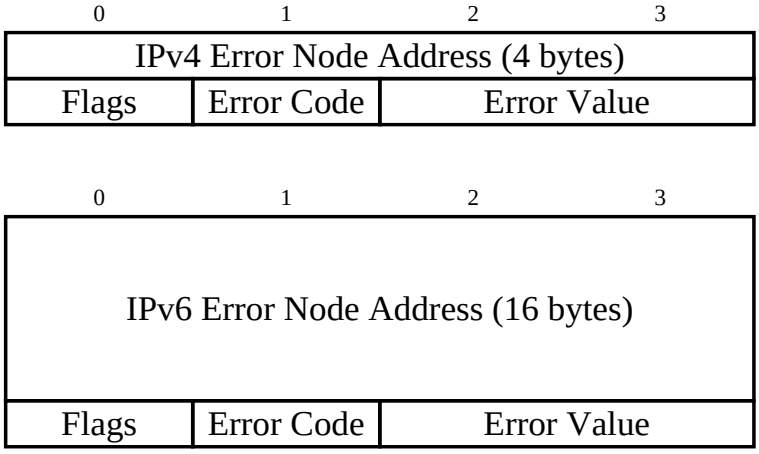

Abbildung B.6: ERROR_SPEC Objekt für IPv4 und IPv6.

In Abb. B.6 sind die beiden möglichen ERROR_SPEC Objekte für IPv4 (C-Type = 1), bzw. IPv6 (C-Type = 2) angegeben. Dabei sind die Felder wie folgt definiert:

- **ERROR Node Address:** Adresse des Hosts, der den Fehler entdeckt hat.
- **Flags:**
  - 0x01 = InPlace: Im Host ist trotz des erkannten Fehlers eine vorherige Reservierung beibehalten worden.
  - 0x02 = NotGuilty: Der fehlererzeugende FLOWSPEC ist in allen Komponenten grösser als diejenige, die vom Empfänger gesendet wurde.
- **Error Code:** Fehlerbeschreibung.
- **Error Value:** Zusatzinformationen zum Fehler

Error Codes und Error Values

Die unten folgende Liste gibt einige für diese Arbeit relevante Error Codes und deren Error Values an. Für die komplette Liste wird auf [BZB+97] verwiessen.

- Error Code = 00: ...
- Error Code = 01: Reservierung musste wegen fehlenden Resourcen abgelehnt werden. Die 16 bit des Error Value werden in diesem Fall wie folgt definiert: ssur cccc cccc cccc.
  - ss = 00: Die 12 bits (c) enthalten einen global definierten Sub-Code. (Werte siehe weiter unten.)
  - ss = 10: Die 12 bits (c) enthalten einen Organisations-Spezifischen Sub-Code. RSVP kann diese Werte nicht auswerten.
  - ss = 11: Die 12 bits (c) enthalten einen Service-Spezifischen Sub-Code. RSVP kann diese Werte nicht auswerten.
  - u = 0: RSVP verwirft die Meldung, ohne lokale Werte anzupassen.
  - u = 1: RSVP kann die Meldung verwenden um die lokalen Werte anzupassen.

  r: Reserviert, sollte 0 sein.
  - Sub-Code = 1: Delay bound kann nicht gesetzt werden.
  - Sub-Code = 2: Verlangte Bandbreite nicht vorhanden.
  - Sub-Code = 3: MTU in FLOWSPEC grösser als MTU des Interfaces.
- Error Code = 02: ...

B.3.5 STYLE Objekt: Class = 8

In Abb. B.7 ist das STYLE Objekt (C-Type = 1). Dabei sind die Felder wie folgt definiert:

- **Flags:** Zur Zeit nicht definiert.

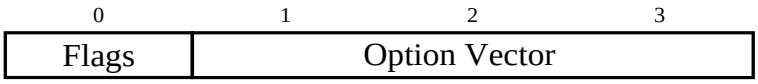

Abbildung B.7: STYLE Objekt.

- **Option Vector:** Satz von bit Feldern, welche die Reservierungsart definiert. Dabei sind diese wie folgt (von links) definiert:
  - 19 bits: Reserviert
  - 2 bits: Sharing control:
    - 00b: Reserviert
    - 01b: Einzelne Reservierung
    - 10b: Gemeinsame Reservierung
    - 11b: Reserviert
  - 3 bits: Sender selection control:
    - 000b: Reserviert
    - 001b: Wildcard
    - 010b: Explicit
    - 011b - 111b: Reserviert

B.3.6 FLOWSPEC Objekt: Class = 9

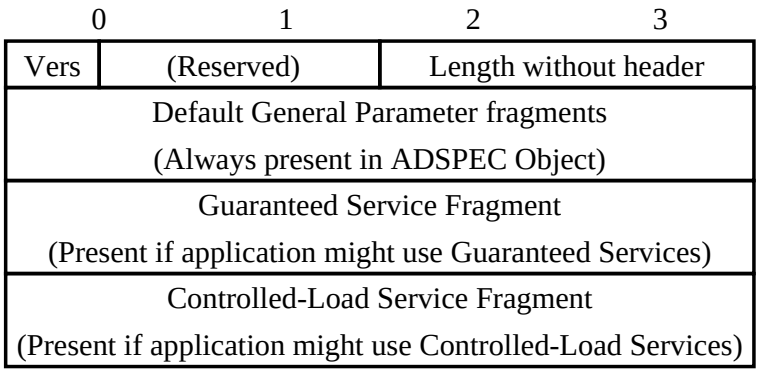

Abbildung B.8: Grundgerüst für FLOWSPEC, ADSPEC und SENDER_TSPEC Objekte.

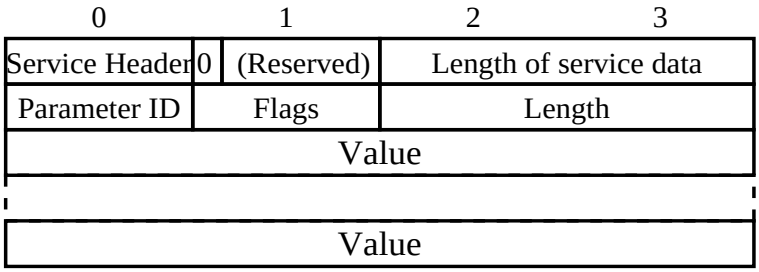

Abbildung B.9: Aufbau eines Parameter eines FLOWSPEC, ADSPEC und SENDER_TSPEC Objekt.

In Abb. B.8 ist der Grundaufbau eines FLOWSPEC Objekte (C-Type = 2, → IntServ Flowspec) gezeigt. In Abb. B.9 der Aufbau eines Parameters. Dabei sind die Felder wie folgt definiert:

- **Vers:** Version des Formates: Zur Zeit 0.
- **Length without header:** Länge ohne Header (Controlled-Load → 7, Guaranteed → 10).
- **Service Header:** Art des Services (Controlled-Load → 5, Guaranteed → 2).
- **Lenght of Service Data:** Länge des Service Packetes (Controlled-Load → 6, Guaranteed → 9).
- **Parameter ID:** ID des Parameters (Controlled-Load → 127 (Token Bucket TSpec), Guaranteed → 127 und 130 (Guaranteed Service RSpec)).
- **Flags:** Zur Zeit nicht definiert.
- **Lenght:** Länge des Parameters ohne Header(Token Bucket TSpec → 5, Guaranteed Service RSpec → 2).
- Für den Parameter 127 existieren folgende Felder:
  - **Token Bucket Rate:** Paketrate des erzeugten Datenstroms. (32-bit IEEE floating point number)

  - **Token Bucket Size:** Paketgrösse des erzeugten Datenstroms. (32-bit IEEE floating point number)
  - **Peak Data Rate:** Maximale Rate, die erreicht wird. Ist keine Ratenkontrolle vorhanden, so ist dieser Wert unendlich. (32-bit IEEE floating point number)
  - **Minimum Policed Unit:** Minimale Grösse der Pakete, die zu berücksichtigen sind. Wert gleich 0 ist ungültig. Kleine Werte erzeugen grössere Reservierung von Bandbreite, da worst-case angenommen wird. (32-bit integer)
  - **Maximum Packet Size:** Kleinster Path-MTU der angetroffen wird. So wird eine Fragmentation von Paketen verhindert.

- Für den Parameter 130 existieren folgende Felder:
  - **Rate:** (32-bit IEEE floating point number) → RFC2212.
  - **Slack Term:** . (32-bit integer)

B.3.7 FILTER_SPEC Objekt: Class = 10

In Abb. B.10 sind die möglichen FILTER_SPEC Objekte für IPv4 C-Type = 1), IPv6 (C-Type = 2) und IPv6 Flow-label (C-Type = 3) gezeigt. Dabei sind die Felder wie folgt definiert:

- **SrcAddress:** Adresse des Senders.
- **SrcPort:** UDP/TCP Source Port.
- **Flow Label:** 24 bit Flow Label, definiert in IPv6. Kann gebraucht werden, um in einem IPv6 Netz die Pakete zu identifizieren.

B.3.8 SENDER_TEMPLATE Objekt: Class = 11

- C-Type = 1: IPv4 SENDER_TEMPLATE Objekt Analog IPv4 FILTER_SPEC Objekt (siehe B.3.7).

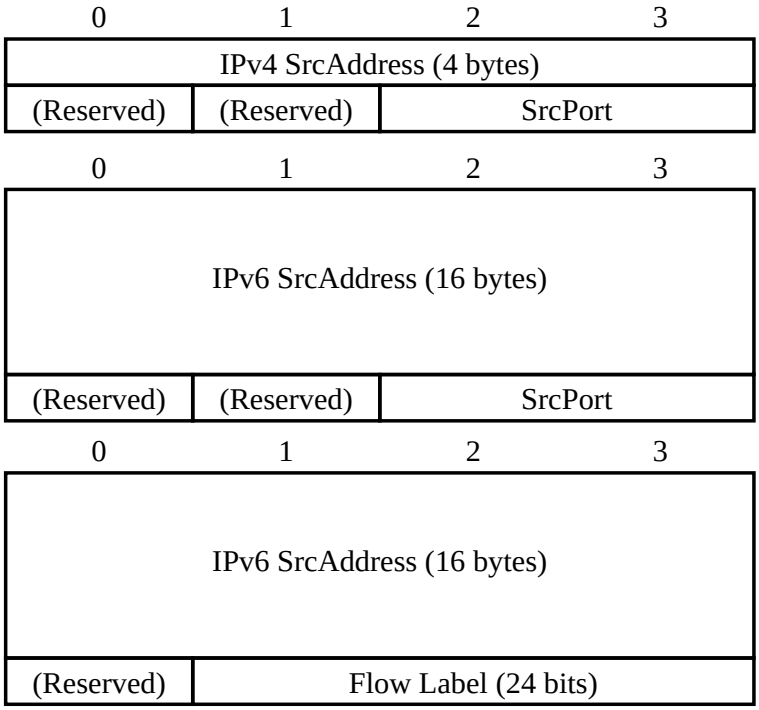

Abbildung B.10: FILTER_SPEC Objekte für IPv4 und IPv6.

- C-Type = 2: IPv6 SENDER_TEMPLATE Objekt Analog IPv6 FILTER_SPEC Objekt.
- C-Type = 3: IPv6 Flow Label SENDER_TEMPLATE Objekt Analog IPv6 Flow Label FILTER_SPEC Objekt.

B.3.9 SENDER_TSPEC Objekt: Class = 12

Gleich wie FLOWSPEC (Abb. B.8 und Abb. B.9). Definition der Felder siehe B.3.6

B.3.10 ADSPEC Objekt: Class = 13

Wie in Abb. B.8 zu sehen, wird das ADSPEC Objekt (C-Type = 2) aus einem Header und drei Fragmenten der Art wie in Abb. B.9 (Default General Parameters: Service 1, Guaranteed Service: Service 2 und Controlled-Load Service: Service 5) gezeigt. Dabei sind die Felder wie in B.3.6 definiert. Zusätzlich sind für das Default General Parameter Fragment folgende Parameter (PID 4, 6, 8,

10) definiert:

- Für den Parameter 4 existieren folgende Felder:
  - **IS hop count:** (32-bit unsigned integer).
- Für den Parameter 6 existieren folgende Felder:
  - **Path b/w estimate:** (32-bit IEEE floating point number).
- Für den Parameter 8 existieren folgende Felder:
  - **Minimum path latency:** (32-bit integer).
- Für den Parameter 10 existieren folgende Felder:
  - **Composed MTU:** (32-bit unsigned integer).

Anhang C

Abkürzungen

BB	**B**andwidth **B**roker
CD	**C**onfiguration **D**aemon
CBQ	**C**lass **B**ased **Q**ueueing
DSCP	**D**iff**S**erv-**C**ode-**P**oint
DSR	**D**iff**S**erv-**R**outer
ISB	**I**nternet **S**ervice **B**roker
ISI	USC **I**nformation **S**ciences **I**nstitute
ISP	**I**nternet **S**ervice **P**rovider / Internet Zugangs Vermittler
QoS	**Q**uality **o**f **S**ervice / Dienstgüte
RDG	**R**SVP-**D**iffServ-**G**ateway
RDG-BL	**R**SVP-**D**iffServ-**G**ateway-**B**-**L**ayer
RED	**R**andom **E**arly **D**iscard
RIO	**R**ED with **I**n and **O**ut
RSVP	Ressource **Re**ser**v**ation Setup **P**rotocol
RSVPd	Ressource **Re**ser**v**ation Setuo **P**rotocol Daemon
SLA	**S**ervice **L**evel **A**greement
TC	**T**raffic **C**ontrol
VPN	**V**irtual **P**rivat **N**etwork
WFQ	**W**eighted **F**air **Q**ueueing

Literaturverzeichnis

[Alm98] W. Almesberger. Linux traffic control - imolementation overview. Technical Report SSC 1998/037, EPFL, November 1998.

[ASK99] W. Almesberger, J. H. Salim, and A. Kuznetsow. Differentiated services on linux, February 1999. draft-almesberger-wajhak-diffserv-linux-00.txt.

[BBC+98] S. Blake, D. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services, December 1998. Internet RFC 2475.

[BCD+98] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry. The copps (common open policy service) protocol, August 1998. Internet Draft.

[BCS94] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview, June 1994. Internet RFC 1633.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol, September 1997. Internet RFC 2205.

[Cla88] D. Clark. The design philosophy of the darpa internet protocols. ACM SIGCOMM 88, August 1988.

[CW97] D. Clark and J. Wroclawski. An approach to service allocation in the internet, July 1997. Internet Draft: draft-clark-diff-svc-alloc-00.txt.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. In *IEEE/ACM Transactions on Networking Vol.1 N.4*, pages 397–413, August 1993.

[FJ95] S. Floyd and V. Jacobson. Link-sharing and resources management models for packet networks. In *IEEE/ACM Transactions on Networking Vol.3 N.4*, August 1995.

[GB99] M. Guenter and T. Braun. Evaluation of bandwidth broker signalingarequipa: Tcp/ip over atm with qos ... for the impatient. Technical Report 99-002, IAM, May 1999.

[RG97] S. Rampal and R. Guerin. Flow grouping for reducing reservation requirements for guaranted delay service, July 1997. Internet Draft: draft-rampol-flow-delay-service-01.txt.

[SPG97] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service, September 1997. Internet RFC 2212.

[Wro97] J. Wroclawski. Specification of the controlled-load network element service, September 1997. Internet RFC 2211.