QoS Signalling and Charging in a Multi-service Internet using RSVP

Dissertationsschrift in englischer Sprache
vorgelegt am Fachbereich Informatik
der Technischen Universität Darmstadt von

Diplom-Wirtschaftsinformatiker Martin Karsten

geboren am 10.07.1971 in Kempen-Hüls, jetzt Krefeld

zur Erlangung des Grades eines
Doktor-Ingenieur (Dr.-Ing.)

Darmstadt 2000
Hochschulkennziffer D17

Vorsitz:	Prof. Dr. Wolfgang Henhapl
Erstreferent:	Prof. Dr. Ralf Steinmetz
Korreferent:	Prof. Dr. David Hutchison (Lancaster University, UK)

Tag der Einreichung:	15. Mai 2000
Tag der Disputation:	5. Juli 2000

Abstract

Because of the high level of uncertainty about the future traffic mix and the heterogeneous nature of *Quality of Service* (QoS) technologies and service contracts, it is very likely that a future commercial multi-service Internet requires a general service signalling architecture to reliably deliver end-to-end QoS. In this thesis, the suitability of the *Resource Reservation Protocol* (RSVP) for this purpose is investigated and assessed. The thesis presents the following contributions: A flexible QoS signalling architecture is described, based on an extended version of RSVP. A new protocol engine has been designed and implemented, containing a number of design and algorithmic improvements over previous work. This implementation has been used to experimentally verify the technical suitability of RSVP. Additionally, new protocol elements have been developed to allow for flexible charging mechanisms of service invocations. Based on several calculation models for service invocations, these protocol elements are assessed to cover a variety of potential usage cases, namely cost-based pricing, auction-based pricing and advance service invocations. As a result, it is concluded that an extended version of RSVP is indeed a good candidate to form the basic building block for an overall commercial QoS signalling architecture.

Table of Contents

List of Figures

List of Tables

Abbreviations

ALTQ	Alternate Queuing
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BA	Behaviour Aggregate
BGP	Border Gateway Protocol
BGRP	Border Gateway Reservation Protocol
CBQ	Class-Based Queuing
CIDR	Classless Inter-Domain Routing
COPS	Common Open Policy Service
CPU	Central Processing Unit
DiffServ	Differentiated Services
DSCP	Differentiated Services Code Point
ECN	Explicit Congestion Notification
FF	Fixed Filter
FlowSpec	Flow Specification
HeiBMS	Heidelberg Buffer Management System
HeiRAT	Heidelberg Resource Administration Technique
HeiTS	Heidelberg Transport System
HFSC	Hierarchical Fair Service Curve
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IntServ	Integrated Services
I/O	Input/Output
IP	Internet Protocol
ISP	Internet Service Provider
ISSLL	Integrated Services over Specific Link Layers
NBMA	Non-Broadcast Multiple Access

OIatPSB	Outgoing Interface at Path State Block
OutISB	Outgoing Interface State Block
PC	Personal Computer
PHB	Per-Hop Behaviour
PHopSB	Previous Hop State Block
PNNI	Private Network-Node Interface
PSB	Path State Block
QoS	Quality of Service
RAM	Random Access Memory
RSB	Reservation State Block
RSpec	Reservation Specification
RSVP	Resource Reservation Protocol
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SE	Shared Explicit
SLA	Service Level Agreement
ST2	Internet Stream Protocol Version 2
TCSB	Traffic Control State Block
TSpec	Traffic Specification
UNI	User-Network Interface
VCM	Virtual Circuit Management
VC	Virtual Circuit
VPN	Virtual Private Network
WF	Wildcard Filter
YESSIR	Yet another Sender Session Internet Reservations

Trademarks

3Com	3Com is a trademark of 3Com Corporation
FreeBSD	FreeBSD is a copyright of FreeBSD Inc.
Gigabyte	Gigabyte is a trademark of Gigabyte Technology Co., Ltd
Linux	Linux is a trademark of Linus Torvalds
OPNET	OPNET is a trademark of OPNET Technologies, Inc.
Pentium	Pentium is a trademark of Intel Corporation.
Solaris	Solaris is a trademark of Sun Microsystems, Inc.
SparcServer	SparcServer is a trademark of Sun Microsystems, Inc.

Other company, product and service names may be trademarks or service marks of others.

Chapter 1: Introduction

The relevant technology that is used to run the Internet is developed and defined within the *Internet Engineering Task Force* (IETF) working groups. The standardization process of the *Resource Reservation Protocol* (RSVP) [BZB+97] and the *Integrated Services* (IntServ) architecture [Wro97a] has created significant expectations about the migration of the Internet towards an integrated multi-service network. Afterwards, objections against the resulting signalling and data forwarding complexity have led to the establishment of a new working area, called *Differentiated Services* (DiffServ) [BBC+98], in which much simpler solutions are sought. Furthermore, numerous other proposals (see Chapter 3) have been made to deliver predictable *Quality of Service* (QoS) in packet-switched networks. However, it is unlikely that a single QoS technology can satisfy the diverse requirements for a future multi-service Internet. Additionally, it is highly questionable whether any such technology will be uniformly deployed on a global scale. In this thesis, it is argued that building predictable end-to-end network services requires a signalling architecture that integrates diverse data forwarding technologies and allows for service requests for various topological scopes. The only relevant standard proposal for a flexible signalling protocol is given by RSVP. While there might also be reluctance against deploying uniform signalling mechanisms, such mechanisms have an important integration role to simplify inter-operation between different autonomous systems and to integrate varying technologies and strategies, similar to e.g, the *Border Gateway Protocol* (BGP) [RL95] for routing in the current Internet. Furthermore, they will serve as the principal interface to incorporate charging information. It is clear that the technical potential to discriminate service requests must be accompanied by corresponding economic calculations. Accounting of service requests and appropriately charging for them is essential to protect a multi-service Internet from arbitrary service requests and to create a funding mechanism to extend network capacity at the most desired locations at the expense of those users that actually use these resources. The main objective for such work is flexibility to support all reasonable employment scenarios. On the other hand, it seems unsuitable to precipitously invent new signalling mechanisms, before the full potential of existing (yet maybe extended) proposals has been investigated and exploited.

1.1 Goals

In this thesis, it is intended to show how stringent decoupling of service interfaces and protocols from service instantiation (as e.g. initially intended for RSVP and IntServ) can create a realistic point of view on the issue of service signalling. The main goal for this work is to assess the applicability of RSVP to realize a flexible QoS signalling architecture, incorporating the necessary charging mechanisms for commercial exploitation. To do so, it is necessary to develop an overall signalling architecture and carry out the suitability assessment of RSVP both conceptually and technically. The other important goal for this thesis is to investigate the issue of price calculation and charging in a multi-service Internet. At the same time, it is an explicit goal for this work to adhere to existing standardization proposals as much as possible.

It is not a goal of this thesis to design a detailed and overall optimal technology to deliver QoS in the Internet or to suggest particular schemes for pricing and charging network services. Corresponding to the well-known paradigm to distinguish between mechanisms and strategy, this thesis investigates and proposes flexible yet efficient mechanisms, leaving it open for future work to answer the question about the optimal strategies to employ them.

1.2 Structure

This thesis is structured as follows. In Chapter 2, the general topic for the thesis is motivated, as well as the specific choice of its focus and methods. Chapter 3 presents and discusses existing results and related work in the area of QoS and economics for packet-switched networks, since these general topics constitute the foundation of this work. The first step to investigate the topic of this thesis is described in Chapter 4, in which the design for a flexible QoS signalling architecture is presented, based on abstract reasoning about its fundamental building blocks. In Chapter 5, the technical feasibility of RSVP is evaluated by describing an innovative implementation design, its realization and examining the performance of the resulting software. The last step to fulfil the overall goals of this thesis is carried out by developing calculation models for a multi-service Internet and charging mechanisms for RSVP-based service invocations. Both are evaluated to conceptually demonstrate the commercial applicability of this approach. This work is described in Chapter 6. Finally, Chapter 7 concludes the thesis and summarizes the major contributions. An outlook to future work items is given. Some particular aspects are described in the appendices. For reasons of clarity and brevity, they are not part of the main body of the thesis.

Chapter 2: Motivation

In this chapter, the scope, goals and methods for this thesis are presented, motivated and discussed in further detail. The motivation begins with describing a very broad vision and general challenges for this vision. The chapter continues by narrowing down the focus to more detailed aspects, which resemble the relevant topics for this thesis.

2.1 Vision of an Integrated Multi-service Internet

The current Internet offers a packet-based, best-effort service via the *Internet Protocol* (IP) [Pos81] and provides no assurances about the quality of transmission, other than the best effort of all participating entities. Packets can be garbled, lost, duplicated or arbitrarily delayed. This particular technology is well-suited and very effective to serve the application mix it has been initially designed for. This can be directly concluded from the current success of the Internet in number of users and increase in topological size. At the same time, there are other communication networks, like the telephone and cable-TV network, which deliver a certain transmission quality for exactly one application.

A network that can carry many different applications seems highly desirable to support the large diversity of communication applications that already exists and will be created in the future. In fact, it can hardly be imagined to build new and separate networks to support new applications.

A single network infrastructure is economically more efficient than the current situation, even if it cannot be as optimized as a dedicated network [She95]. This is due to economies of scale that can be achieved both in the human resources sector as well as in the equipment sector, if only one technology has to be supported. The apparent need for interaction and combination of multiple applications, for example, between human tele-conferencing and collaborative applications, even further supports this point of view. Finally, there is a potential for resource savings through multiplexing gains in both the short and medium time-scale. As an example for medium time-scale multiplexing, consider that the telephone network usually reaches is peak load during the day, whereas the TV network records it peak demand during the evening. The

existence of short time-scale multiplexing gains for a heterogeneous applications mix that includes variable transmission rate applications, is a well-known fact that can be deduced directly from basic queuing theory [Kle75].

The vision behind the work for this thesis is the eventual realization of a truly global and ubiquitous communication infrastructure, supporting the largest diversity of applications conceivable, e.g., ranging from text-based email to tele-surgery.

2.2 Challenges for an Integrated Multi-Service Internet

Numerous proposals have been made to enhance the basic service model of the Internet and provide the capabilities for predictable QoS characteristics (see Chapter 3). However, a number of issues, which highly influence the suitability of any particular technology, are often not clearly expressed.

2.2.1 Demand Expectations

Two important but often confused aspects to assess different alternatives for QoS provisioning in the Internet are given by the time scale one considers and the expectations about the demand for different types of applications, i.e., the traffic mix that is carried at that time. An insightful classification is given in [She95] to distinguish different traffic types, especially elastic from inelastic applications. Adaptive applications can be considered to have an inelastic lower bound, thus eventually being inelastic.

If one assumes that future traffic patterns are mainly determined by elastic applications, the current best-effort service of the Internet might be very appropriate (with only slight modifications) for a future multi-service network. The conclusion that a relatively small amount of inelastic traffic can be carried with high performance can be drawn quite obviously. Resources will be plentiful for traffic from inelastic applications, if it is simply marked to receive a higher priority when packets are forwarded. On the other hand, if inelastic applications, especially long-lived flows, will form the major part of overall demand, one might choose to favour ATM-like technology as main interconnection layer, because it is optimized for such applications. Elastic traffic could then be carried over a meshed IP topology on top of the underlying ATM network and the impact on overall resource provisioning would be negligible.

If, however, the traffic mix is roughly evenly distributed between both types of applications, it seems favourable to have a connectionless packet-switched interconnection layer, which is augmented by special mechanisms to accommodate and appropriately carry inelastic traffic without wasting too many resources. It is actually imaginable that ‘roughly evenly’ covers a quite significant part of the overall distribution scale. In [She95] it is argued that in this case, an integrated network is economically more efficient than having two distinct network infrastructures. This point of view is supported in this thesis and furthermore, it is argued that because of the fundamental uncertainty about future demand patterns, a system should be built to support any possible traffic mix, yet not necessarily each potential traffic mix at the absolutely lowest cost. It can be optimized at any time, if the demand patterns actually stabilize.

It should be noted here that demand and technology in fact have a bidirectional dependency. Technological choices determine the lowest possible price at which certain services will be offered, which in turn influences overall demand. This cyclic dependency makes it even harder to predict any particular traffic mix.

2.2.2 Commercial Environment

When considering a communication network that is eventually designated to serve in a commercial environment, the most prevalent requirement is given by the need for clear and precise, yet expressive, semantics of service invocations. This is especially important when it comes to the upper and tighter end of performance characteristics. Such properties do not only have to apply to the interface between end-users and network providers, but instead, they are equally important for interfaces between peering provider networks. The main reason for this assessment has no direct technical background, but is deduced from the necessity of end-customers’ trust in an advanced communication infrastructure. So far, product liability and customer satisfaction has been paid rather little attention in the area of computer software and Internet technology, opposite to almost every other industry branch (even the traditional telephony industry after deregulation). In order to create a similar trust as regular users currently have in their residential telephones, a new era of clear and precise responsibility for the value-chain of creating an advanced commodity product ‘integrated services network’ is very likely to come forth. For such commercial reasons, it is also very important that both service provision and service accounting are intimately tied together. It does not make sense to invent new mechanisms to de-

liver a high-end service without presenting a commercially sound service interface and vice versa.

2.3 Towards an Integrated Multi-service Internet

The chain of arguments in the previous sections has its background in both technical and non-technical areas. On the one hand, the are indications for the technical (and consequently economical) superiority of integrating all communication applications over a uniform, robust, packet-switched interconnection layer. On the other hand, it can be expected that some of the underlying principles of current Internet network service provision, as e.g., cooperation, implicit traffic contracts, and the assumption of every party's goodwill, cannot be carried over to the commercialized future of an integrated multi-service Internet. Instead, explicit service negotiations will be necessary for a large spectrum of topological scopes and time scales of service duration.

2.3.1 QoS Signalling

The need for an overall and integrative QoS signalling architecture is stated in the most recent draft of the Internet Architecture Board [Hus00]. Such a general architecture is needed to integrate the prevalent variety of QoS subnet technologies and to provide meaningful end-to-end services [Hus00]. In general, a receiver-oriented model to request network services can be adopted for such service signalling [Hus00]. In order for a communication service to make sense, both sender and receiver must have reasonable interest in establishing it. Only at the receiver's end-system, there is knowledge what kind of network transmission performance is desired and reasonable, therefore, it seems plausible to delegate service requests to receivers. A sender determines how much traffic it emits and announces this to the network and receivers, however it does not make sense to have the sender request a certain network performance, if e.g., a receiver has not enough interest in any particular transmission performance (or transmission in the first place) and discards incoming packets anyway.

One particular goal of such a QoS signalling architecture, besides flexibility, is given by the consideration that requests for tight per-flow service guarantees will present the relatively highest challenge to a future multi-service network, both in terms of signalling and data forwarding complexity. In this thesis, it is therefore argued to optimize the QoS signalling architecture for that case. The development of a QoS signalling architecture is a highly constructive task, which

requires a significant understanding of existing QoS proposals and application requirements in both theory and practice. The suitability and even superiority of any particular proposal can hardly be proven, other than by experiments with real users in a real large-scale network. In order to evaluate the specific architecture that is proposed in this thesis, a usage case evaluation is carried out in Section 4.5 to at least back up the claim of its flexibility.

2.3.2 Employing Standardization Proposals

Besides the protocol work for QoS signalling that has been carried out within the IETF, numerous other publications have proposed specific, often called ‘lightweight’, signalling protocols. On the other hand, the general acceptance of QoS mechanisms for the Internet has been rather slow, so far. This slow acceptance can be explained as follows. Providing Internet connectivity in the first place has evolved from an academic niche to an increasingly emerging market. Producers of this ‘good’ consequently have only little incentives to differentiate their products, because satisfying the increasing number of users already ties up their capacities. This situation might dramatically change in the near or middle-term future, if the market for basic connectivity begins to saturate. However, deployment of Internet QoS technology can only occur incrementally, but also only makes sense, if eventually the provision of true end-to-end service commitments is possible. Therefore, in order to alleviate the realization of QoS-enabled Internet communication, it is important to adhere to a minimum of standardized inter-operation capabilities. Consequently, it seems inefficient to precipitously develop (or even standardize) new signalling mechanisms, before the full potential of existing (yet maybe extended) proposals has been investigated and exploited. These arguments do not apply to short-term solutions for urgent congestion problems, but for any middle- and long-term activity to develop a flexible end-to-end QoS solution.

2.3.3 RSVP

The specification of RSVP [BZB+97] is currently the only major IETF standardization proposal for a QoS signalling protocol. An older approach is given by the *Internet Stream Protocol* (ST2) in its latest version ST2+ [DB95], but because of no significant acceptance, it can be considered largely historic. It is not as flexible and robust as RSVP, because it employs a sender-oriented setup scheme and hard state in intermediate systems. Furthermore, it represents a dual-stack approach in the network layer, such that two entities of internetworking protocols have to

co-exist in intermediate systems. A comparison between both protocols from either point of view can be found in [DHVW93] and [MESZ94].

Since its invention, the applicability of RSVP is under discussion, because of scalability concerns about the required amount of state and additional computation overhead for periodic soft state refreshes. However, RSVP is designed to be extremely flexible and robust and provides a number of features, which seem hard to achieve without this resulting complexity. The overhead introduced by message refreshes can be arbitrarily traded off against robustness by modifying the refresh period. Its receiver-oriented service model fits quite well with the motivation for a QoS signalling architecture as presented in Section 2.3.1. Additionally, through this receiver-oriented service model, RSVP suits the well-known challenges stemming from IP multicast and heterogeneous service requirements from multiple receivers. For those reasons, RSVP basically resembles the only currently existing choice for such a highly flexible signalling mechanism.

In this thesis, the approach to investigate the technical feasibility of RSVP is carried out in multiple directions. First, a few RSVP extensions are presented, which enable the protocol to more efficiently handle requests for aggregated service invocations and thus, increase overall flexibility. As with other work items of this thesis, it is impossible to prove the absolute truth of certain assumptions. While the theoretical complexity of RSVP is not debatable, it is highly interesting to relate the theoretical knowledge to practical performance figures in order to gain additional insight about its applicability. This work is described in Chapter 5 and is based on a new and innovative internal design and a number of optimizations. It forms a large part of the overall thesis contribution.

2.3.4 Charging

The transition of the Internet towards a commercially funded and used integrated services network raises, among others, the question about how network usage can be charged appropriately. Clearly, the current charging schemes (mainly flat-fee access-based or time/volume-based) will not be sufficient in the presence of multiple service classes, resource reservation and discrimination between different usage requests [MMV97]. It is obvious that if network traffic is discriminated by QoS mechanisms, some negative feedback is needed to prevent users from arbitrarily allocating resources. On the other hand, a market and competition mechanism is needed to provide users with the best and most inexpensive level of service, while creating in-

centives for network providers to supply more resources when there is sufficient demand. Therefore, charging mechanisms are needed to compensate for the allocation of scarce resources. Furthermore, they are needed to create a funding mechanism to extend network capacity at the most desired locations at the expense of those users that actually use these resources. A QoS signalling architecture is in principle independent of charging models built on top of it, however, both have to be related, to achieve clear responsibility for service delivery when service requests induce financial consequences. For example, the legal aspects of service charging might require that a network provider can present detailed accounting records for high-end services upon request. A signalling architecture must therefore allow for the collection of such data.

Several aspects have to be taken into account, when designing charging mechanisms for a multi-service Internet. Primarily, there is a need to ensure technical inter-operation between charging mechanisms and network service signalling on the one hand and accounting and actual network service on the other. Furthermore, cost-based price calculation resembles an important input to capacity planning and sales price calculation. In this thesis, work is presented to develop calculation models and charging mechanisms for an RSVP-based QoS signalling architecture and thus, to assess the suitability of RSVP as a general basic signalling mechanism as a whole. Similar to a QoS signalling architecture, the resulting system proposal can hardly be proven to be optimal for real world employment. The charging mechanisms that are proposed in this thesis, have been experimentally implemented to realize a proof-of-concept about their suitability. Additionally, they are analytically assessed in order to verify their applicability to a variety of cost and price calculation scenarios. In order to do so, certain calculation methods have been developed, as well. These calculation models have been designed, but not solved or optimized. They are intended to serve as basis for economic research to improve and apply them.

Chapter 3: Fundamentals of QoS and Economics

In this chapter, the existing fundamental knowledge about QoS for packet-switched networks and economics of communication networks is presented and related work is revised. As well, a brief overview about RSVP is given.

3.1 QoS - Related Work

An insightful overview and evaluation of various QoS technologies for the Internet is presented in [MEH00]. In the following sections, a brief overview about the relevant state of the art is given.

3.1.1 QoS Provision

QoS-enabled network services can be provided by a spectrum of different technologies which can be mainly distinguished by their trade-off between complexity and capabilities. Those technologies are built upon different models of QoS. A *QoS model* consists of the following macroscopic facets (related to [Bra97]):

- **Scope** defines the logical distance over which a service model is provided.
- **Granularity** defines the smallest unit which is treated individually by a service model.
- **Time Scale** defines the granularity in time on which services are being provided.
- **Control Model** defines the entities which exert control over the network and how they do it. As extreme cases, control could either be located exclusively in the network or in the end-systems, with a continuum of hybrid forms in between.

QoS models apply different *QoS tools* in order to achieve their respective goals:

- **Network Design and Engineering** deals with the proper setup and maintenance of network equipment based upon experience, expert knowledge, heuristics or formal optimization methods [Ker93]. Sometimes this is also called *provisioning*.
- **Traffic Engineering** is concerned with distributing the incoming traffic for a given network among potential transmission paths by mechanisms as, e.g., explicit routing capabilities [RVC99] or QoS-based routing schemes [AWK+99].

- **Signalling and Admission Control** is an integrated set of mechanisms which builds upon a session paradigm where users of the network signal their requirements explicitly and the network consults admission control modules to grant or reject those requests. Examples of proposed signalling protocols for the Internet are RSVP [BZB+97] and ST2+ [DB95], while proposed admission control procedures are either parameter- [PG93] or measurement-based [GK97,BJS99]. Without per-flow admission control, only statistical QoS assurances can be given on a per-packet basis.
- **Packet Scheduling** is concerned with the decision which packet to send next on a given link if there is a number of packets waiting for service [SV98]. This decision of course has a major impact on the QoS experienced by a packet since the queuing delay usually constitutes a large portion of the total end-to-end transfer delay.
- **Traffic Policing/Shaping** deals with forming traffic to an either negotiated or at least advertised level at the edges of networks or between distinct network elements in order to ensure a controllable load of the network. Example mechanisms in this area are the well-known leaky or token bucket traffic envelopes [Tur86,Cru91].
- **Adaptiveness** is the capability of end-systems to react upon congestion in the network by evaluating signals from the network. These signals can be either implicit, e.g., the loss behaviour of the network, or explicit, e.g., by a so-called *Explicit Congestion Notification* (ECN) [RF99]. Dynamic and possibly congestion-based pricing of network services is also a form of network signal proposed for managing QoS [MMV95,GK99b].

These tools have different time-scales and, thus, not all of these tools are suitable for all different kinds of QoS models. In general a combination of those tools is needed to implement a certain QoS model. For example, proper network design and engineering is certainly a prerequisite to the successful operation of any QoS model. The QoS models mainly differ in how much emphasis is put on each element in their combination of tools, which reflects different assumptions on how powerful the different components are assessed.

Proposed QoS models in the Internet arena are listed below and classified according to the above mentioned aspects.

IntServ. This model [BCS94] is composed of RSVP as a per-flow signalling protocol and service classes defined by the Integrated Services (IntServ) architecture [Wro97b,SPG97]. Its scope is to provide end-to-end services, and its granularity is determined on a per-flow basis, i.e., it is very fine-grained. Through RSVP, the concept of a session is introduced which deter-

mines the unit of time-scale of that model. The control loop of the IntServ model tends to be network-centric in the sense of offering fairly advanced services inside the network, which can be requested by applications. A usual counter-argument against this QoS model is based on the resulting complexity for network elements. The most important tools to implement the IntServ model are signalling and admission control obviously, but it also depends upon a sensible network design and engineering in order to keep the blocking probability for sessions low. Furthermore, policing and shaping is required to keep reserved traffic in its negotiated borders.

DiffServ. While still evolving, the Differentiated Services (DiffServ) [BBC+98] model's scope is rather inter-domain, based upon the peering between network domains. It outlines a framework which allows for bilateral contracts by *Service Level Agreements* (SLA) at such borders. Currently, the DiffServ proposals only deal with different flavours of forwarding behaviours inside network elements, so-called *Per-Hop Behaviours* (PHB) [HBWW99,JNP99]. It is assumed that by concatenating PHBs, it is possible to build sensible services, thereby allowing for an end-to-end scope in result. However this is not a necessary direction of evolution for DiffServ. The PHBs operate on traffic aggregates, so-called *Behaviour Aggregates** (BA) [NC00], and thus its granularity is fairly coarse-grained. Similarly, the unit of time-scale is expected to be long-termed since SLAs should be rather static, although with the addition of dynamic SLAs by the introduction of signalling protocols, the time-scale could decrease [FNM+99]. As with the IntServ model, the control model is rather network-centric, but for some of the PHBs defined, it is necessary for end-systems to adapt themselves to the network state. Recent results [Cha00,Bou00,SZ99] have shown that only by providing static service level agreements (SLA), the theoretical worst-case performance guarantees for providing per-flow services might exhibit a larger conflict with the objective to utilize resources as efficient as possible, than often assumed. Therefore, it can be concluded that building end-to-end services out of DiffServ PHBs forwarding classes will not be fully sufficient to satisfy the diverse requirements for a future multi-service Internet. The tools upon which DiffServ builds are mainly network design/engineering and traffic policing or shaping. However, an introduction of dynamic SLAs will make it move towards an emphasis of signalling and admission control mechanisms, as well.

Over-provisioned Best-effort. This model argues for a continuation of the current operation of the Internet in a best-effort manner. The underlying assumption is that over-provisioning net-

* Recently, the IETF DiffServ working group decided to change the terminology to *Per-Domain Behaviour*

work resources is both sufficient and possible to sustain the single service nature of the current Internet. The scope of that model is end-to-end in nature since all the intelligence is located in end-systems. Since there is no state in the network and all traffic is treated at the same granularity, it is as coarse-grained as possible. The time-scale of this model is very large and essentially equal to the length of one capacity planning cycle. Since end-systems are the only intelligent units in the network, the control model is end-system-centric. Certainly, the most important tool applied by that model is that of network design/engineering in order to always provide for a super-abundance of network resources. However, in periods of scarcity of resources this model relies on the adaptiveness of end-systems to such presumably transient situations, and hence, it is intrinsically targeted to elastic applications.

Price-controlled Best-effort. This is not a single proposal, but a notion of several authors [MMV95,KMT98,CP99] who feel that pure over-provisioning is not sufficient without additional means of signalling besides packet loss. This additional signal is a per-packet price that may depend on the internal state of the network, i.e., its congestion level. However, some authors even propose a semi-static approach with fixed but differentiated prices per packet [Odl99]. With respect to its properties this model is very similar to the pure over-provisioned best-effort model, however, its time-scale is related to the frequency of price announcements and, due to the ability to set prices, the network is not as passive as for that model. With regard to the tools that are applied by that model it also has to be noted that it heavily relies on the combination of network design/engineering and the adaptiveness of end-systems, and hence, it is intrinsically targeted to elastic applications. Furthermore, it is crucial for correct operation that the end-systems' or users' sensitivity to pricing signals can be estimated. In order to provide a flat-fee best-effort service in combination with price-controlled best-effort, it is important that traffic for both classes can be distinguished by routers, hence a DiffServ-like marking scheme is required to distinguish service classes. Price marks must be set only with respect to the congestion level of this traffic class. Furthermore, if the price-controlled service class is supposed to offer a higher transmission quality than flat best-effort, a single tail-drop queue might not be sufficient to appropriately favour packets from this service class over best-effort packets. Additionally, there are open questions about potential time-gaps between pricing signals and adaptation at the end-systems, as well as the prerequisite of global consensus between all participants in such a model.

3.1.2 RSVP

RSVP is designed to carry reservation requests for packet-based, stateless network protocols such as IP. In essence, it is aimed at combining the robustness of connectionless network technology with flow-based reservations by following a so-called *soft state* approach. State is created to manage routing information and reservation requests, but it times out automatically, if it is not refreshed periodically. In the RSVP model, senders inform RSVP-capable routers and receivers about the possibility for reservation-based communication by advertising their services via PATH messages. These messages carry the sender's *traffic specification* (TSpec) and follow exactly the same path towards receivers as data packets, establishing soft state in routers. Receivers initiate reservations by replying with RESV messages. They contain a TSpec and a *reservation specification* (RSpec) and also establish soft state representing the reservation. RESV messages are transmitted hop by hop and follow exactly the reverse path that is formed by PATH messages.

RSVP treats reservation requests (TSpec and RSpec) as opaque data and hands them to complementary local modules, which are able to process them appropriately. Being tuned to support large multicast groups, RSVP uses logic from these modules to merge reservation requests that share parts of the transmission path. Merging takes place at outgoing interfaces by merging requests from different next hops that can be satisfied by a single reservation at the same interface. As well, reservation requests that are transmitted towards a common previous hop are candidates for merging. The amount of merging possible is determined by the filter style, which is requested by receivers. For shared filter style, all reservations for the same interface and all reservations towards the same previous hops are merged, respectively. When distinct filter style is requested, only reservations that specify the same sender are being merged. Furthermore, filter styles are classified by whether applicable senders are wildcarded or listed explicitly. The (potentially empty) list of senders is called *FilterSpec*. The following filter styles are currently defined:

- FF (fixed filter): single sender, distinct reservation
- SE (shared explicit): multiple senders, shared reservation
- WF (wildcard filter): all senders, shared reservation

All these filter styles are mutually exclusive and a session's filter style is determined from the first arriving RESV message. The combination of TSpec and RSpec is called *flow specification* (FlowSpec). The combination of FlowSpec and FilterSpec is referred to as *flow descriptor*.

The initial description of RSVP [ZDE+93] explains many details and the design philosophy of the protocol. A detailed overview about RSVP operation in combination with the IntServ architecture is given in [WC97].

3.1.3 QoS Signalling

A number of proposals have been made for lightweight signalling protocols or signalling protocols that are dedicated for certain application scenarios. In this section, some of the recent proposals for packet-switched networks are briefly presented and discussed in comparison to RSVP.

In [PS99], the lightweight reservation protocol *YESSIR* is presented. This protocol is tied to the employment of RTP [SCFJ96] as transport protocol and exploits RTCP reports to establish end-to-end resource reservations. Reservations are requested from the sender and thus, the protocol does not support heterogeneous requests from multiple receivers in a multicast group. It provides a very elegant and efficient way to carry out resource reservation for certain application scenarios. Its processing overhead is reported to be less expensive than that of RSVP by a factor between 2 and 3. However, this speed-up has to be traded off against the loss of flexibility. Furthermore, only limited details about the performance tests are given in [PS99]. For example, the testing scenario did not burden the YESSIR protocol engine with the ability to support various service classes. Since the software is not publicly available, so far, it is impossible to verify these results or fill the gaps in the test. As a conclusion, while being an interesting approach, this proposal does not provide the flexibility of RSVP and its performance gain must be subject to further study.

The *Boomerang* project [FNM+99] defines a limited set of protocol primitives to establish quantitative QoS parameters for end-to-end IP flows. It requires no immediate support from end-systems, because it is proposed to be embedded into ICMP messages. However, this is not a very clean system design. In a sense, it can be considered as a very useful and efficient subset of RSVP's specification, but, as the authors of [FNM+99] note, it cannot be considered to replace a protocol like RSVP.

An alternative suggestion has been made in [PHS99] by the development of *BGRP* to enable resource management for inter-domain trunk reservations. This proposal borrows from the specification of BGP [RL95] and considers backbone reservations along the sink tree to each destination. It addresses some problems of the current version of RSVP for this scenario. This

work also presents a very interesting and useful analysis of the resulting protocol overhead. However, as shown in Section 4.5.3 and Section 5.1.2, these improvements can be quite easily carried out by slightly modifying RSVP and without losing the general scope of RSVP, instead of defining a completely new protocol.

While all the proposals mentioned above contain very important insight and present valuable properties, they are, at the same time, limited to support certain application scenarios. None of them is as flexible as RSVP. However, approaching the overall problem with a single homogeneous protocol seems to be clearly advantageous when compared to using multiple protocols for different services and scopes, because a single protocol eliminates functional redundancy.

Very interesting work has been carried out in the area of *Open Signalling* [CLSS98]. However, the focus of this work goes much beyond the understanding of signalling for this thesis, in both effort and goals. It is targeted towards creating programmable interfaces employing active networking nodes. In that sense it can be considered more heavy-weight and less evolutionary as compared to the traditional protocol-based approach.

In comparison to proposals how to specifically realize the inter-operation of certain QoS mechanisms, the work of this thesis concentrates on the interface role of a QoS signalling protocol. Work as described in [LR98,BYF+00,TKWZ00] can be considered as complementary, in that low-level detailed aspects of inter-operation are examined and solved.

3.1.4 Integrated QoS Architectures

Besides ATM and the already mentioned IntServ approach, several research groups have developed comprehensive QoS architectures, consisting of both QoS models and the respective signalling mechanisms. These projects have created enormous insight into the challenge of QoS provision in the Internet. However, all these research architectures have in common that they cannot be applied directly to the global Internet, because of their overall complexity and the tight relation between individual components. All approaches focus on reservation-based services and thus, employ similar QoS tools as the IntServ architecture. The following three architectures can be considered as the most influencing ones.

Tenet

The core building block of the Tenet approach [BFM+96] is given by a real-time channel, which is defined as a simplex unicast end-to-end connection with performance guarantees and

restrictions on traffic shapes. Two versions of the Tenet protocol suite have been implemented. The network service is provided using a connection-oriented channel administration protocol in combination with a separate network protocol and two dedicated transport protocols for continuous media and real-time message delivery, respectively. Thereby, the Tenet approach employs a dual-stack model of employing two network protocol entities in intermediate systems. In this project, issues like channel setup, multi-party communication and flexible service interfaces have been studied, among others. The need for pricing as back-pressure method against greedy service requests is recognized, but no corresponding mechanisms have been integrated. With respect to QoS signalling, the protocol suite is less flexible than RSVP in the areas of multicast communication and connection establishment, mainly because it employs sender-oriented connection setup and hard-state at intermediate systems.

HeiTS

This QoS architecture, developed as part of the *HeiProjects* [Her92], uses the concept of stream handlers as the core building block, to which QoS assurances are applied. The end-to-end data path is described as a concatenation of stream handlers, including resources in both end systems and network. The network part of this overall architecture is given by HeiTS and consists of essentially ST2 [DB95] as network protocol and a dedicated transport protocol. Additionally, aspects of media scaling and resource reservation in advance have been investigated in this project. Further modules of the overall system are especially given by HeiRAT [VHN93] for resource administration and HeiBMS for buffer management. The HeiProjects are a very comprehensive approach to QoS for packet-switched communication. However, when considering QoS signalling, this architecture inherits the problems of ST2 as discussed in Section 2.3.3.

QoS-A

The QoS-A architecture [CCH94] defines the most complete and comprehensive QoS architecture. There is a distinct architectural model consisting of layers and planes, similar to the ATM model. The network layer is based on ATM technology and QoS-A is focused on service contracts between the user and the network. A service contract consist of a variety of QoS parameters, for which the level of service commitment can be independently chosen, i.e., instead of a fixed set of service classes, each parameter combination is allowed. A cost parameter is integrated into the service contract, as well, to express the system’s effort to deliver certain services. Furthermore, the service contract specifies a QoS maintenance policy, which describes how the system should react upon variations in the actual delivered QoS. As with the HeiProjects,

this project also investigated resource management within end systems. One particular work item is given by integrating QoS control functionality in the Chorus micro kernel [CCR+95]. There has been no separate QoS signalling protocol developed in this project, but the architecture's flow manager is designed similar to ST2, however, with extensions supporting fast connection establishment and advance reservation service.

3.2 Economics - Related Work

There are two economic perspectives which can be applied to the Internet. Work in the area of welfare economics seeks to optimally allocate resources between competing users, subject to certain criteria. From a business economics point of view, a network provider's goal to maximize its profit is adopted and cost and price calculation is carried out appropriately. In a theoretical perfect market, both perspectives lead to the same results, however, the intermediate path to this result differs. Furthermore, both approaches have to model a very large and complex system and deal with a lot of uncertainties. Consequently, the respective work can only analyse a particular fragment of a real situation and thus, both approaches typically lead to different inaccuracies.

3.2.1 Welfare Economics

So far, significant research work has been published on the issue of pricing in telephone networks (see e.g. [MV91] and references contained herein). In the area of packet-switched multiservice networks, approaches to find welfare-optimizing price models can be found in e.g. [MMV95,MM97,KVA98,PSC98,SFY95,GK99b]. A good overview about the fundamental economic theory is given in [Var96]. There are mainly three basic concepts that are used to assess the operation of a communication network:

- **Utility** refers to a user's valuation of the successful transmission of a packet or a flow of packets.
- A **utility curve** describes the type of an application in terms of the user's utility depending on the amount of resources allocated to transmit his network traffic.
- **Marginal cost** denotes the additional cost that is caused by the additional transmission of a packet or a flow of packets.

It has been established by many researchers [MMV97,WPS97,SFY95] that pricing based on real marginal cost is not sufficient for communication networks. For example, the marginal cost

of transmitting an additional packet through a network is basically zero. Consequently, congestion costs are often used as marginal costs, i.e., consumption of resources prohibits other users to use them and lowers their utility. The goal of a pricing algorithm is then to provide the optimal resource allocation with respect to the users' utility function. In this section, the most influencing proposals are briefly presented. Three game-theoretic criteria are usually used to asses the characteristics of any pricing algorithm:

- **Nash equilibrium** defines a state in which no participant can increase its individual lot through unilateral actions, i.e., a local optimum for each participant.
- **Pareto optimality** defines a state in which no lot can be increased for any participant without decreasing the lot of others.
- **System optimality** defines a state in which the sum of all individual lots is maximized.

In [MMV95], the concept of a *Smart Market* for packet-switched networks is presented. This is a conceptual system, which is not intended to be a real engineering proposal, but to illustrate some economic principles. The Smart Market is created by carrying out *Second-Price* auctions [Vic61] at each router for all packets. Packets contain a representation of an amount of money, which is used as a bid and serves to reveal the user's true valuation of the service, i.e., essentially his utility curve. This system leads to a system-optimal allocation of resources. In [MM97], the Smart Market proposal is extended to cover flow-based services, as well, however, the overall practicability is still questionable, because it relies on independent auctions at each router.

A completely different proposal has been made in [Odl99] to design pricing and QoS in the Internet according to the old pricing system of the Paris Metro. This model is intended to be extremely simple by statically partitioning the network into two classes and differentiating both classes by price. For economic reasons, it can be concluded that the load situation would usually be better in the high-price class and thus, this service class would deliver superior service. However, there is little flexibility in this approach and no intention to provide real service guarantees.

Realizing the respective shortcomings of both previous alternatives, a sophisticated model for pricing Internet services and analysing the resulting effects has been developed and presented in [GK99b]. It aims to combine the simplicity of [Odl99] with the economically powerful proposal in [MMV95]. Packets are statistically marked similar to the ECN proposal for the Internet [RF99] and it can be statistically shown that the resulting system converges to a system-

optimal state as long as all utility curves are strictly concave. This proposal is definitely the most elaborated model to price packet transmission in the Internet by a simple mechanism and employing economic insight. However, its application to the real Internet remains open, because of the prerequisite of strictly concave utility curves. Furthermore, it is not clear, whether the theoretical results hold in the presence of real timing and delay effects at the scale of a global communication network.

Opposite to the proposals presented above, there are publications [PSC98,VLK99] indicating that stability and optimality criteria are not necessarily fulfilled by very simplistic proposals. Specifically, in [PSC98] a careful investigation of game-theoretic results in the presence of multi-dimensional QoS vectors and burstiness of network traffic is reported. It is argued that these desirable economic characteristics are guaranteed, if there are enough resources in the network to potentially accommodate all users requirements, but not necessarily in any other case, especially if utility curves are not concave.

3.2.2 Business Economics

Little work can be found in the area of business economics to find provider-oriented calculation models for packet-switched multi-service networks. From a business economics point of view, communication services are characterized by:

- availability of a non-storable resource (network capacity)
- high fixed costs & low variable costs

In business economic theory, these characteristics, which are similar to traditional telephony, electricity, aircraft seats, etc., are dealt with by using a management technique called *Yield Management* [Lei98]. When Yield Management is used, calculation is based on *profit contribution* and *opportunity costs*, instead of using full-cost or variable-cost calculation. The concept of profit contribution means that each resource unit is sold for a price higher than its marginal cost. The difference of both contributes to the overall revenue, which must exceed the overall investment for the appropriate business cycle. Opportunity costs describe the fact that selling a resource unit prohibits using it for another business transaction. Under given price-demand patterns, prices can be optimized to maximize the overall revenue.

In the context of communication networks, granting a service request is profitable as long as the charge for this request is higher than its marginal cost. However, to reach the optimum profit, opportunity costs must be added to the marginal costs, i.e., a service request might prohibit

using the resources for another request with a potentially higher revenue. In fact, opportunity costs dominate marginal costs by far, since variable costs are negligibly low. The main task is to optimize capacity and prices according to a given price elasticity (i.e. demand per price), such that the overall revenue is maximized. A very general, complete and hence complex price calculation model is presented in [WPS97], although it lacks full applicability to multiple service classes. Particularly, details about the relation of prices to resource-oriented admission control are not considered in [WPS97]. These aspects are further discussed in Chapter 6.

In [Lei98], it is attempted to describe a complete cost- and price-model for a typical Internet Service Provider (ISP). Then, it is examined how the introduction of a new service offering, namely IP telephony, in combination with a flat-fee pricing model influences the ISP's revenue. The author shows, how this change negatively affects the provider's profit and clearly points out that offering new services in combination with the traditional flat-fee pricing model of current Internet access service directly leads to this result.

One interesting theoretical business aspect must be mentioned at this point. Similar to other emerging electronic businesses, providers of Internet service face the risk of high customer fluctuation. Electronic markets might create an almost friction-free economy for commodity products. In such an economy, however, it is clear that price is the dominating and potentially only factor for a consumer's decision. In theory, this leads to a price-war between competitors trying to drive each other out of business to realize economies of scale. Prices are then expected to decrease until and below production cost. Eventually, a monopoly situation can be established by the winner. To avoid this scenario, it might be necessary for ISPs to vertically integrate their business with providing application-level services and to realize so-called *economies of scope*.

3.3 Conclusions

When considering the diverse proposals for enabling QoS in the Internet in combination with the economic background information, it becomes clear that all this work provides extremely useful insight into the problem and its potential solution. However, there is currently no clearly superior technology that has the ability to supersede all others. Because of the different applicability scenarios and the natural heterogeneity that will result from this diversity of QoS technologies, it can be concluded that a controllable and flexible interaction layer is needed to integrate all these approaches. Such interaction can be carried out by a uniform mechanism for

signalling service requests. Other proposals than RSVP are generally limited in their application scenarios and thus, do not provide the same flexibility. In analogy to the open question about superior technical mechanisms for providing QoS, no single and completely sound business model can be foreseen at this time. Therefore, there is also need for flexible charging mechanisms, which allow for highly differentiated pricing and accounting strategies. An early overview about charging for Internet communication has been published in [KSWS98b]. As well, the relation of QoS and charging for communication networks has been surveyed in in [KSSW00].

Chapter 4: QoS Signalling Architecture

As motivated in Chapter 2 and further discussed in Chapter 3, it is rather unlikely that a future commercial multi-service Internet can provide end-to-end QoS completely without service signalling. In this chapter, the scope for developing a signalling architecture is an arbitrary combination of packet-switched networks, some having end-systems attached, some might be pure transit networks, all of them are running IP as the basic interconnection layer protocol. The main focus of this work is to define a useful global architecture for service interfaces at traffic exchange points between peering networks (including end-systems/end-subnets). It is not an immediate goal of this architecture to describe the most efficient packet-forwarding technology to actually build end-to-end services, since that is beyond the scope of this thesis. Furthermore, it is not a goal to precisely define the inter-operation of potentially different QoS technologies at such service exchanges points. However, a usage case evaluation is described in Section 4.5 in order to conceptually show the applicability of this architecture.

4.1 General Taxonomy of QoS Signalling Architectures

It is important to clarify fundamental roles of building blocks for a QoS architecture. In this section, a taxonomy is given along the well-known distinction between mechanism and strategy/ algorithm.

4.1.1 Distinction of Service Interface Mechanism from Distributed Algorithm

One must clearly distinguish two roles of a signalling protocol like, e.g. RSVP. It has been initially designed as a distributed algorithm to enable multiple entities to cooperatively deliver a certain service, i.e., multiple routers creating a reservation-based, end-to-end transmission service. On the other hand, it can be considered as an interface specification to request services, regardless of how the service is technically constructed. In this role as an interface mechanism, it can be used to negotiate related administrative information, such as prices and payments, as well. Much of the current QoS debate does not clearly distinguish between both roles, thus creating significant unnecessary misunderstanding. If any single and coherent protocol can pro-

vide for multiple incarnations of both roles, it can be considered superior to having multiple distinct protocols for different services, because a single protocol eliminates functional redundancy. In that sense, certain extensions for RSVP are described in Section 4.4, which enable the resulting protocol architecture to serve as both concise interface mechanism and, if desired, distributed algorithm. RSVP can be considered as being a basic mechanism, whereas the traditional IntServ approach of establishing per-hop and per-flow reservations can be described as one particular example of a strategy employing the RSVP mechanism.

4.1.2 Basic Alternatives for QoS Signalling Architectures

To illustrate the distinction between distributed strategies and interface mechanisms, a somewhat instructive comparison can be made by comparing ATM's separation between UNI [ATM96b] and PNNI [ATM96a] with the Internet architecture, as depicted in Figure 1 (a) and (b). Along the dimension *distributed algorithm*, two types of functionality of a communication

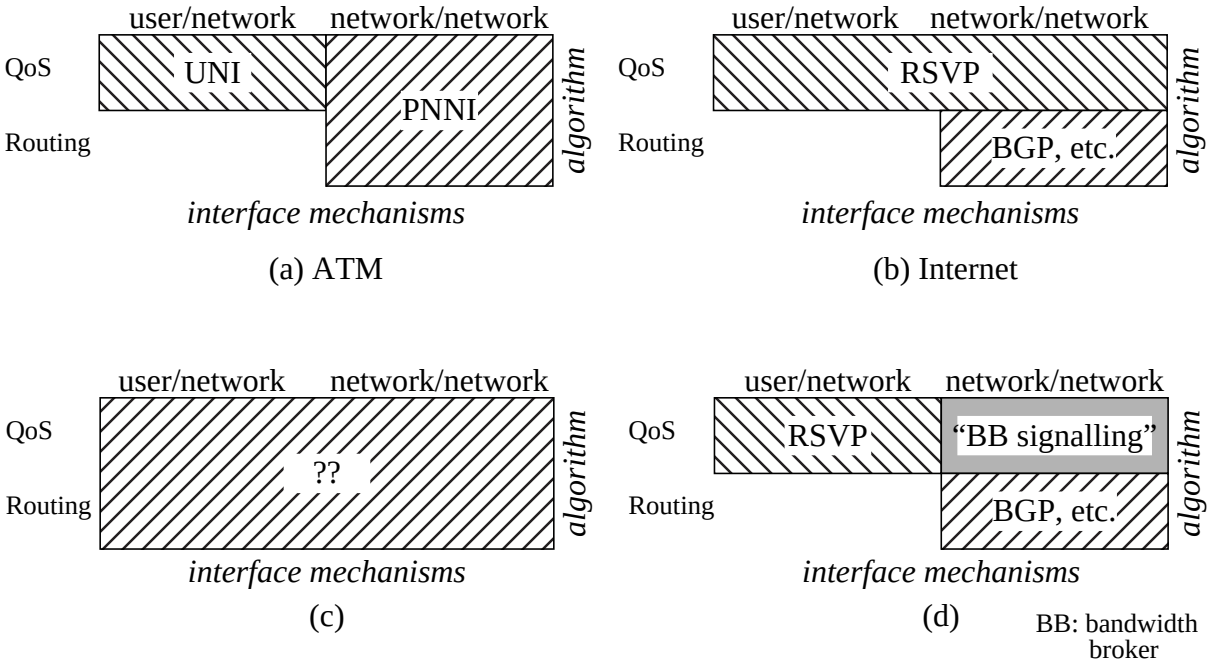


Figure 1: Basic Alternatives for QoS Signalling Architectures

network are shown: *Routing*, which establishes basic connectivity between nodes and *QoS*, which refers to performance characteristics applying to specific flows between sets of nodes. QoS is negotiated between participating entities. The second dimension is given by *interface mechanisms* and populated by two instances, as well. In both architectures, end users are not involved in the global routing mechanism, hence, in both scenarios the remaining problem space

is split in a certain way. In ATM, this separation is done along interface mechanisms. With the advent of RSVP, the Internet community developed a separation along the two algorithmic tasks. However, there are proposals to decouple basic connectivity from QoS provision in ATM signalling, as well [RHV99]. It is important to note that at least either of these splits is inevitable for a modular and concise design of the overall architecture, since otherwise, an interface to global network-level routing would have to be presented to end users. This choice is depicted in Figure 1 (c), whereas part (d) of Figure 1 shows the alternative of further subdividing the problem space, as suggested in e.g. [NJZ99]. In the light of these issues, the different points of view in the current debate about how to provide QoS in the Internet can be classified by which of those four basic alternatives turns out to be superior for a future multi-service Internet. Thereby, although not complete in terms of technology choices, these architectural alternatives establish a minimal taxonomy for competing QoS proposals for the Internet.

4.2 Proposed Architecture

The most important requirement to consider when assessing the basic architectural alternatives, is to consider interfaces (especially interfaces to end-users) as stable and hard to change. Therefore, service interfaces must be chosen carefully to be very flexible, robust and compatible with future developments. On the other hand, service interfaces must not inhibit the performant realization of services. The best way to accommodate these goals is to make interfaces as lean yet expressive as possible.

4.2.1 Concept

This proposal for an overall QoS signalling architecture conceptually consists of three layers as depicted in Figure 2. With respect to the taxonomy in Section 4.1, it is assumed that a basic connectivity mechanism exists, which is given by a routing protocol and packet forwarding nodes

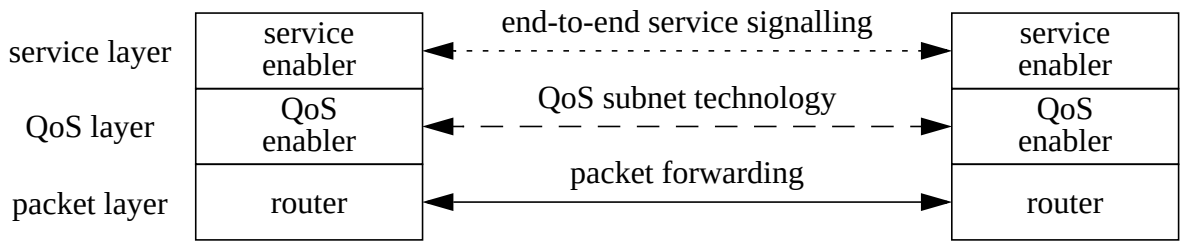


Figure 2: QoS Signalling Architecture - Conceptual View

called *router*. This is described as *packet layer* in the picture. The actual QoS technology is represented by an intermediate *QoS layer*. An entity that, besides carrying out router functionality, also performs active load management by policing, shaping or scheduling, or marking packets for a certain scheduling objective, is called *QoS enabler*. A pure QoS enabler, however, does not participate in end-to-end signalling. Advanced end-to-end services that allow for specification of performance characteristics are realized using a complementary interface on the *service layer*. The entities of this layer, which handle service signalling and potentially flow-based load control (admission control) are denoted as *service enabler*. A service enabler can also perform the role of a QoS enabler, but not vice versa. Of course, in a future QoS-enabled Internet, further open issues, such as QoS routing have to be addressed, as well. However, their eventual precise definition is currently beyond the scope of a QoS signalling architecture.

The focus of this work is to design a flexible service layer that allows for integration of a variety of QoS layers. In the conceptual architecture, the layers can be considered as roles. Compared to previous work, the role or functionality of each layer is not bound to certain nodes in the network topology. Instead, it depends on a network operator's particular choice of QoS technology and furthermore, on the service class, which node carries out the role of a certain layer. Detailed usage cases are presented in Section 4.5. While this concept increases flexibility, the resulting architecture is not as completely defined as in earlier proposals like those presented in Chapter 3.

4.2.2 Topological View

To illustrate the application of the conceptual architecture, an example network topology is shown in Figure 3. The picture shows a sender host S and a receiver host R which are connected through an access network each (A_S and A_R) and two backbone networks, (B_1 and B_2). Border routers are shown as entities R_S, R_1, R_B, R_2, R_R in this figure. Service signalling takes place between at least those border routers. Depending on the service class and the particular QoS technology, intermediate routers (not shown in this picture) might participate in the signalling, as well. Furthermore, subnets might employ bandwidth brokers [NJZ99], depicted as BB, to carry out resource allocation for the complete subnet for other service classes. In this case, service requests are either forwarded from border routers to the bandwidth broker or alternatively, the bandwidth broker somehow intercepts the respective service requests. All nodes are classified as either *service-aware*, *partially service-aware* or *service-unaware* as depicted in Table 1. In

case of partially service-aware nodes, these nodes have to distinguish whether to process or just forward a service request. The main criterion for this distinction is very likely to be the service class. This aspect is further illustrated by the usage cases in Section 4.5.2.

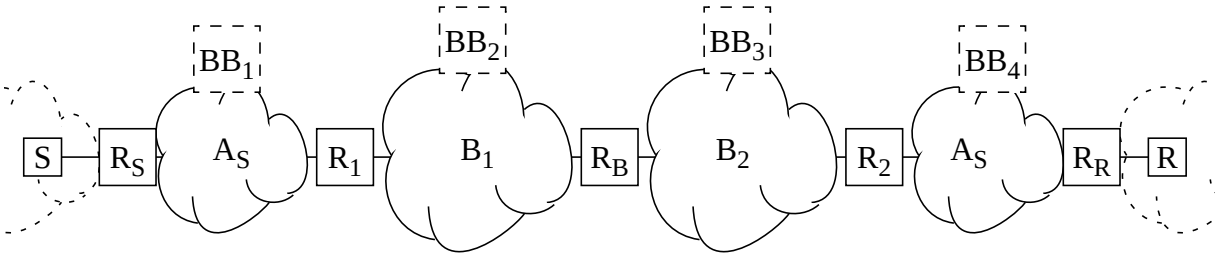


Figure 3: QoS Signalling Architecture - Topological View

Table 1: Service Awareness of Network Nodes

Service Awareness	Description
service-aware	*signalling-capable, support for all service classes*
partially service-aware	*signalling-capable, support for some service classes*
service-unaware	*not signalling-capable*

4.2.3 RSVP as General Signalling Mechanism

In order to satisfy both goals of flexibility and optimization for highly demanding services when realizing a service layer, as discussed in Section 2.3, a solution is given by a uniform extended RSVP interface for advanced services. Using such an interface as service layer entity at each traffic exchange is both sufficient and effective to realize the conceptual architecture for multiple topological scopes and QoS technology alternatives and to create meaningful end-to-end services. With respect to the taxonomy in Section 4.1, this design represents the choice to carry on with the Internet service architecture and to employ RSVP (including the extensions presented in Section 4.4) as the primary signalling mechanism, especially for inter-domain signalling. Initially, it can then be used as a service interface between bandwidth brokers (particularly for dynamic DiffServ SLAs or other coarse-grained QoS technologies).

However, the main motivation is given by the advantage that a future migration to employ RSVP in its initially intended style as distributed algorithm to request and provide per-flow and potentially per-node service guarantees will be alleviated, if the basic mechanisms are already

in place. In such a future scenario, RSVP then acts as a signalling mechanism between each node, as well. Consequently, it is intended that a router employing this extended version of RSVP can efficiently handle both per-flow and aggregated service invocations of multiple service classes. The alternative to invent different signalling mechanisms for per-flow and aggregated service requests or different mechanisms for end-to-end and backbone signalling seem clearly inferior, especially if RSVP can be applied beyond its initial scope without introducing a large overhead. In Chapter 5, it is demonstrated that even the application of RSVP in its initial way of operation does not necessarily exhibit as much of a performance problem, as usually assumed, if the software is well implemented.

In order to emphasize the generalized application of RSVP, the terms *service request* and *service invocation* are used throughout the rest of the thesis to describe the general interface mechanism, despite RSVP's initial designation for actual end-to-end reservations. Only when referring to reservation-based services, the terms *reservation request* and *reservation establishment* are used. To keep the relation with traditional RSVP, protocol messages are nevertheless referred to as PATH and RESV messages.

4.3 Service Classes

A general QoS architecture should be flexible to support arbitrary service classes and be open for new alternatives. To this end, the following services classes are considered to be supported by the architecture. These services classes represent the set of currently discussed methods to provide various QoS assurances for Internet communication:

- *best-effort*

  It is highly desirable for any kind of future multi-service network to provide a flat-fee best-effort service class in order to continue the extremely successful operation of the current Internet and support elastic applications like web-browsing. Such a service provides a low threshold entry, especially for private users, and accommodates a large number of applications and usage scenarios. If a flat-fee service is not offered by technical means, it is essential to develop a sound business model to provide it on top of other service classes.
- *ECN*-priced best-effort*

  ECN-priced best-effort service in combination with 'admission control' gateways is a rather new proposal, presented in [GK99a], that uses ECN-marking [RF99] at routers to signal

* Explicit Congestion Notification

congestion to end-systems and furthermore, argues that ECN-priced best-effort leads to stable resource allocation. Although the applicability of these results might be debatable, ECN-priced best-effort could be considered as a separate service class and provide differentiated service for elastic applications.

- *DiffServ PHB concatenation*
  While currently the quantitative precision for end-to-end services that are provided employing a concatenation of DiffServ PHBs cannot be fully assessed, it seems clear that such services can be built very soon within the Internet and that there will be demand, e.g., in the area of *Virtual Private Networks* (VPN).
- *IntServ*: Guaranteed, Controlled Load, and Guaranteed Rate
  The well-known IntServ service classes represent the upper end of the service range by providing guaranteed per-flow service to applications like high-quality videoconferencing, distributed games or even future tele-medicine.

It is important to distinguish the different motivations that might lead to the provision of a particular service class. The natural motivation is given by the assumption of appropriate demand and price-elasticity to eventually recover the service's cost and operate at a profitable level. Another source of motivation might be given by existing excess resources or by the fact that a service class promises a very high resource utilization. Then, such a service class can be offered at a very competitive price, which in turn stimulates the necessary demand to create an overall profitable business.

4.4 RSVP Extensions

There are mainly two shortcomings in the currently specified version of RSVP, which aggravate its application as a general service interface:

- Traffic flows are either identified by host or multicast addresses, i.e., the specification of subnets as source or destination address is not possible.
- Path state information has to be stored for each service advertisement in order to ensure correct reverse routing of service requests.

In order to appropriately extend RSVP's functionality, existing ideas [Boy97,PHS99] have been taken up and augmented to design a general processing engine for a lean and flexible service interface. The major goal for this work is to achieve a high expressiveness for service interfaces. The extensions are mainly dedicated for, but not restricted to, unicast communication

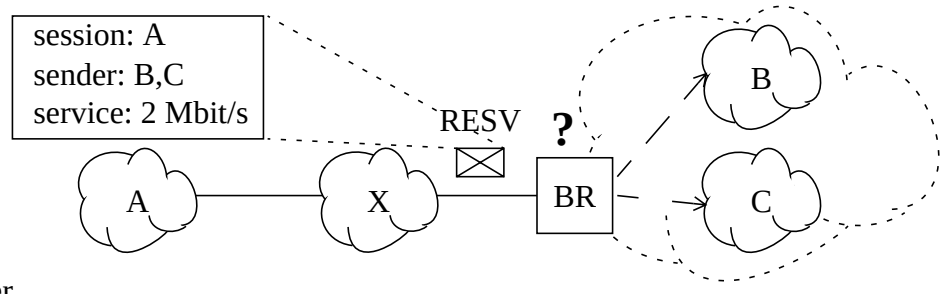
(including communication between subnets) and cover cases where the per-flow model of traditional RSVP signalling, which in the worst case exhibits quadratic state complexity [PHS99], seems inefficient, because the requested transmission performance characteristics do not require flow isolation at each intermediate node. In that sense, the extensions are targeted to aggregated service requests on the control path. This has to be distinguished from the issue of aggregating flows on the data path. For the latter, careful network and traffic engineering, e.g. using MPLS [LR98], is required or alternatively, strict performance guarantees might be given by applying network calculus to multiple flows [SKWS99]. For both multicast in general and non-aggregated performance-sensitive (i.e. inelastic) unicast communication, the current version of RSVP can be considered as very well-suited, especially if recent proposals to increase the overall efficiency of RSVP operation [BGS+00] are implemented.

4.4.1 Compound Addresses

The current specification of RSVP supports only host and multicast addresses. In order to specify service requests for traffic aggregates between subnets, the notion of addresses has to be extended to cover network addresses. A respective proposal to include the *Classless Inter-Domain Routing* (CIDR) extension into RSVP has been made in [Boy97]. In this thesis, the term *generalized address* is used to refer to either a host or a network address, including CIDR prefixes and the special address 0.0.0.0 denoting complete wildcarding. Additionally, it might be necessary to specify several of such addresses within a single session or sender description, thus the notion of a *compound address* is introduced, which consists of a set of generalized addresses. Of course, a dedicated node must exist within an end-subnet to receive and respond to such service advertisements. In principle, any node can emit requests as long as they are authorized.

In order to employ the full flexibility of compound addresses, it is inevitable to introduce a further generalization to specify their handling at certain nodes. During transmission of RSVP messages targeted to a compound address, the border router towards the specified subnet(s) will be hit. In that case, it has to be decided whether the message is forwarded towards multiple destinations or not. If the message is not forwarded, then the resulting service essentially covers only a portion of the end-to-end path. If however, the message is forwarded into multiple subnets, it is not immediately clear how to interpret any quantitative expression of performance characteristics. The term *scoping style* is used to describe the alternatives that such a message

is forwarded to multiple next hops (*open scope*) or not (*closed scope*). To this end, it is an open issue whether the scoping style should be chosen by the node issuing a request or whether it is determined by the network provider depending on its local policy how to provide certain services. As this is a matter of strategy and not mechanism, it is beyond the scope of this work to extensively investigate the detailed aspects of this question. Nevertheless, some usage case examples are given in Section 4.5. In Figure 4, an example RESV message is shown to illustrate the choice between both alternatives.



BR: border router

Figure 4: Compound Addresses and Scoping Style

If RSVP's addressing scheme is extended to include compound addresses, new challenges are presented to the data forwarding engine of a router. In order to support flows targeted to or sent from an end-system at the same time as a session involving the subnet of this end-system, a longest-prefix match on both destination and source address might be necessary to distinguish which packets belong to which session. However, it can be expected that any service establishing performant communication for traffic aggregates between subnets is going to be built using a packet marking scheme, as e.g. the DiffServ model. In the DiffServ architecture, such a case is already considered and alleviated by the fact that only edge-routers are expected to do the full classification to isolate aggregate service contracts from individual flows. In the core of the network, traffic belonging to aggregates is forwarded according to its DiffServ marking and individual flows requiring total isolation can be appropriately serviced using a dedicated DiffServ mark and full packet classification. The same marking scheme can be applied to RSVP messages themselves, such that per-flow request messages are transmitted to the appropriate end-subnet, but not processed by nodes along a trunk flow. This allows for transparent end-to-end signalling, even in the case that a flow is mapped onto trunk service along a part of the path.

A somewhat different treatment of port numbers is necessary to incorporate compound addresses into RSVP. It might be useful to specify a port number, if e.g., the resulting service is used for a single application which can be identified through the port number. In any other case, the port number should be set to zero and effectively denote wildcarding. Analogous to the description in the previous paragraph, a classification challenge exists, which will be alleviated by employing a DiffServ-like marking scheme.

A scheme of compound addresses in combination with the choice of scoping style is more appropriate for service requests between subnets than the initial approach of CIDR addressing of RSVP messages [Boy97], because it overcomes the limitations induced by restricting source and destination to a single address prefix each. Furthermore, the scoping style provides a controllable way to deal with the resulting flexibility. Thereby, it is well-suited to especially provide a signalling mechanism and interface between bandwidth brokers which control the establishment of dynamic SLAs that are eventually provided to traffic aggregates by means of DiffServ PHBs.

4.4.2 Hop Stacking

To reduce the quadratic amount of state that might have to be kept by routers in case of traditional RSVP signalling, it is quite trivial to extend its specification similar to [PHS99]. Traditionally, PATH messages are sent along the same path as the data flow and state containing reverse routing information is kept at each node to allow forwarding of a RESV message along the exact reverse path towards the sender.

In order to alleviate this effect for intermediate nodes, a mechanism termed *hop stacking* can be incorporated into RSVP. From a node's point of view, hop stacking provides a transparent method to employ other approaches for QoS provision without per-flow state at intermediate nodes, e.g., RSVP over DiffServ-capable networks [BYF+00]. However, from an overall system's point of view, hop stacking defines a generic mechanism to carry out RSVP signalling without PATH state at each node. It can be used for trunk signalling or tunnelling, and it provides for an open interaction with traffic and network engineering. In that sense, slightly more freedom is taken to extend the existing RSVP specification than other approaches.

Each router has the option to replace the RSVP_HOP object by its own address and store appropriate state information from PATH messages (traditional operation). Alternatively, the address of the outgoing interface is stored as additional RSVP_HOP object in front of existing

ones. During the service request phase, the full stack of such hop addresses is incorporated into RESV messages and used at respective nodes to forward the service request to previous hops, if no PATH state has been stored. On the way upstream, such a node removes its RSVP_HOP object and forwards the message to the next address found in the stack. This mechanism allows for installation of state information for service requests without the necessity to keep PATH state for each service announcement. This specification introduces even further flexibility as compared to other approaches in that stacking of hop addresses is optional and can be mixed with traditional processing within a single session. A node might even remove the full stack, store it locally together with the PATH state, and insert it into upstream RESV messages, such that the next downstream node does not have to deal with hop stacking at all. Figure 5 illustrates the flexibility of hop stacking. In this picture, nodes C and D perform hop stacking instead of storing local state whereas node E removes the full stack and stores it locally, such that node F does not realize the existence of stacked hops at all. An according RESV message travelling along the reverse path, can find its way back to the sender by local state or stacked hop information.

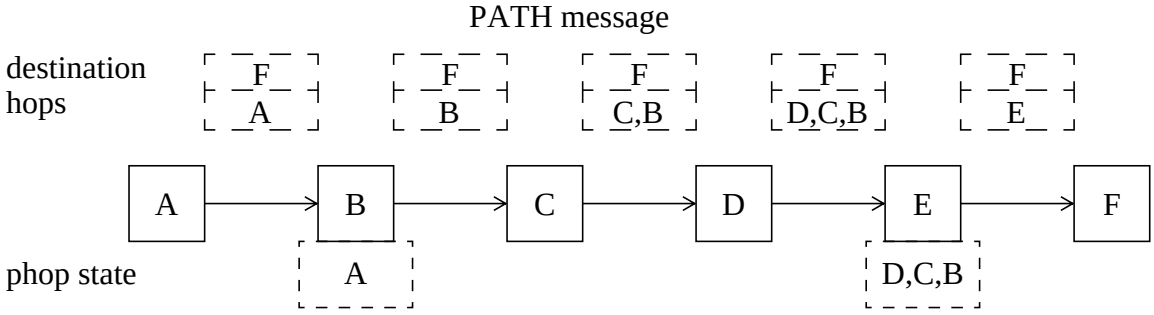


Figure 5: Hop Stacking for RSVP Messages

4.4.3 Interface Semantics

While the extensions presented above form the procedural part of this proposal, it is important to define coherent semantics at a service interface. The inherent meaning of accepting a traditional RSVP message is to adhere to the distributed algorithm, i.e., to appropriately process and forward the request, establishing an end-to-end resource reservation. In this architecture, the semantics are changed such that the meaning of accepting a service request is a (legal) commitment to deliver this service, regardless of its actual realization. For example, compound

addressing provides an interface to transparently incorporate IP tunnels as presented in [TKWZ00]. Similarly to the notion of *Edge Pricing* [SCEH96], this creates a notion of *edge responsibility* for the end-to-end service invocation. Effectively, an application's data flow might be mapped onto several consecutive network flows in the notion of traditional RSVP. In that sense, intermediate nodes carrying out that mapping might actually be considered as 'RSVP gateways' or 'service gateways'.

4.5 Usage Case Evaluation

A collection of usage cases is described in this section to conceptually show the flexibility of the RSVP-based signalling architecture to integrate diverse QoS technologies and create a variety of service scenarios. The usage cases focus on the mechanisms of service layer signalling between service enablers. In case of multiple alternatives, it is left open to further work to determine the optimal strategies to map service requests onto the underlying QoS technology.

4.5.1 Supporting Diverse Subnets

Below, it is briefly presented how various QoS subnet technologies can be integrated by this QoS signalling architecture. Most of these are well-known and treated (together with link layer technologies) by the IETF ISSLL working group (see [ISS00] for a list of documents). However, there's an additional scenario explained below, which supports the proposal of providing QoS by employing ECN marking in a certain way.

IntServ Signalling for Per-Hop, Per-Flow Service

A per-flow service invocation for an IntServ service class can be handled precisely as described in [BZB+97,Wro97a,Wro97b,SPG97] and thus, is not described here in further detail.

Service Signalling across DiffServ Subnet

The details of per-flow service invocations for a service class, which is mapped onto a *DiffServ Code Point* (DSCP) within a subnet, are described in [BYF+00]. Besides the possibility that partially service-aware nodes ignore service requests because of the service class, another option is to bypass them, if the respective DiffServ SLAs are realized as tunnels [TKWZ00].

Service Signalling across ATM Subnet

As has been proposed in numerous publications [SMMT97,BG97,CBB+98,SA98b], RSVP signalling can be carried out across ATM subnetworks and exploit the QoS capabilities of an ATM subnet. Further details on how to map QoS parameters can be found in [SKS00b].

Service Signalling across ECN-priced Subnet

A somewhat speculative proposal to provide QoS has been made in [GK99a]. It is based on intermediate nodes carrying out statistical ECN-marking, which are interpreted as small charges at edge systems. It is claimed that the resulting economic system provides a stable resource allocation which could then be considered to resemble a certain QoS. In order to mimic the all-or-nothing characteristic of regular admission control, the ingress of the subnet acts like a risk broker and decides whether to accept or reject a service invocation. This risk broker subsequently undertakes the economic risk of guaranteeing the accepted service even in the presence of rising congestion and thus, charges. Another option is for the ingress node to adapt the sending rate to the current congestion situation. Since the ECN mechanism is an end-to-end mechanism and usually requires a transport protocol to carry the feedback from the receiver back to the sender, it is not immediately obvious how such an approach should be realized for a partial path in the network. However, if RSVP signalling is employed between the edge nodes of such a partial path, the periodic exchange of RSVP messages can be used by the egress node to provide at least some kind of feedback to the ingress node.

4.5.2 Flexible Service Signalling Techniques

The following scenarios present a variety of service invocations that can be supported using the RSVP-based QoS signalling architecture. Note that all the scenarios presented below can be carried out at the same time in the same infrastructure.

Reduced State Service Signalling in Backbone Networks

In this scenario, a backbone network is assumed, which allows for establishment of trunk reservations between edge nodes, which are dynamic in size and routing path. Because of a potentially large number of edge nodes that advertise services to each other, it may be inappropriate to potentially keep state for each pair of edge nodes at routers. Furthermore, the service class does not provide precise service guarantees, but rather loosely defined bandwidth objectives. RSVP signalling can be carried out between each pair of nodes including the hop stacking extension. Path state is not stored at intermediate nodes and reservations towards a common send-

er are aggregated at each node. Consequently, the worst-case amount of state that has to kept at each router is linear to the number of nodes, instead of quadratic. This example resembles the basic state reduction technique of BGRP [PHS99].

Service Mapping of Flow Service to Trunk Service

The notion of compound prefix addresses allows for expression of service mappings of individual flows into aggregated trunk services. Individual flow requests that arrive at the ingress end of the trunk service are incorporated into a single service request, which is described by a compound prefix address and transmitted to the other end of the trunk. In Section 4.4.1, it is discussed how to distinguish trunk traffic from other packets which might be exchanged between the corresponding end systems. Alternatively, a tunnel might be established for the aggregation part of the data path [TKWZ00] and eligible packets are encapsulated into the tunnel. Nevertheless, it is useful to have a notion to describe the aggregate traffic flow, such that signalling can be carried out across multiple autonomous systems.

Lightweight Service Signalling

One might even go one step further and consider an RSVP PATH message as service request, while RESV messages only confirm the currently available resources. In that case, the end-systems keep track of the network state along the data path and no state information is stored at intermediate nodes. Such a scenario can be realized by a specific service class instructing each intermediate node to report its current load situation and service commitments, but without carrying out any particular activity for this request. PATH messages record their way through the network by hop stacking and RESV messages are initiated by receivers including the amount of service that this receiver requests. On their way back to the sender, the RESV message is used to collect the information whether this service is currently possible. Intermediate nodes are free to store as much state information as necessary and feasible to report best-effort estimates of the current load situation.

4.5.3 Application Scenarios

In addition to the simple techniques described in the previous section, the following examples describe more complete application scenarios which employ these techniques.

Service Signalling for Dynamic Virtual Private Networks

Consider a corporate Internet user wishing to establish a VPN between multiple locations. Each of these locations operates an IP network with a different subnet address prefix. Furthermore, it is deemed important to dynamically adapt the requested VPN capacity according to each location's current demand. In this example, it is examined how the resulting service requests are handled by a backbone network B, which is crossed by traffic from multiple locations. The scenario is illustrated in Figure 6. The corporate subnets are denoted with S_1, S_2, S_3 and S_4. The edge routers are depicted as E_1, E_2 and E_3. Each corporate subnet emits service advertisements (e.g. from a bandwidth broker or dedicated gateway) towards the other subnets, either separately or bundled with a compound destination address. The corresponding service requests might be treated separately or also be aggregated at certain nodes and targeted towards a compound sender address.

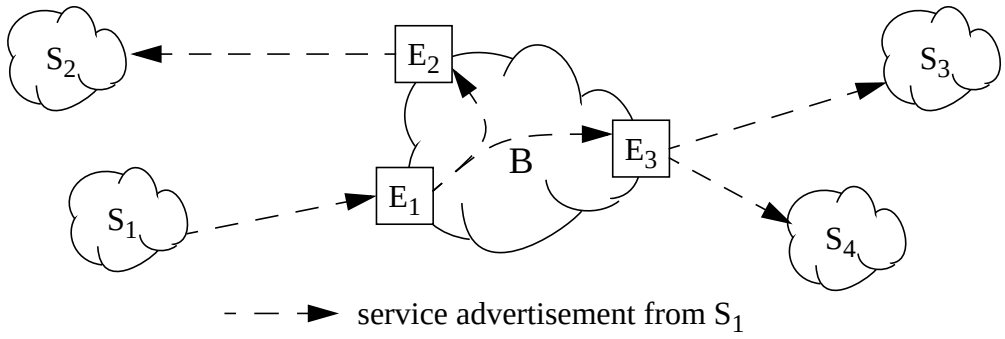


Figure 6: Virtual Private Network Scenario

As an example, S_1 advertises a certain total amount of traffic towards the other subnets, hence there is no specific service description for each subnet. The advertisement is processed by E_1 and forwarded to the other edge devices. If the backbone QoS technology is given by a combination of static SLAs and a bandwidth broker, E_1 obtains the information about multiple egress edge devices from the bandwidth broker and splits up the request accordingly. If intermediate nodes also act as service enablers, the advertisement is forwarded as a bundle, until an intermediate node contains two routing entries for the different destination subnets. This is similar to multicast distribution and applies the service mapping technique described in the previous section. The correspondent service requests from S_2, S_3 and S_4 traverse back to S_1

establishing the subnet-to-subnet service. Because of the dynamic nature of RSVP signalling, the dimensioning of the VPN service can be adapted over time.

Inter-Domain Service Signalling

A scenario of inter-domain trunk reservation signalling has been described and carefully analysed in [PHS99]. The same advantages as reported for BGRP can be obtained by employing the reduced state signalling technique described in the previous section. If combined with a recent proposal to bundle and reliably transmit refresh messages [BGS+00], RSVP provides a functionally equivalent solution having the same complexity as described there. However, there is no completely new protocol needed.

4.6 Summary of Results

In this chapter, some fundamental aspects to distinguish different proposals for QoS architectures have been addressed and, as a result, a minimal taxonomy has been presented, based on the separation of mechanism and strategy. Then, a general QoS signalling architecture has been developed, which employs an extended RSVP as its major building block and covers a variety of service classes. This architecture is more flexible than previous approaches, because the conceptual roles are not statically bound to certain nodes in the topology. Certain protocol extensions have been designed to enable RSVP to serve as general and flexible QoS signalling protocol. The architecture and protocol extensions will be published in [KSBS00]. The approach is aligned with the most recent QoS architecture draft of the Internet Architecture Board [Hus00] in that it supports the integration of heterogeneous QoS technologies and allows the provision of end-to-end services. This conclusion is conceptually shown by a usage case evaluation.

Chapter 5: Implementation and Evaluation

Besides the ideas presented in Section 4.4, various proposals have been published, which describe useful extensions to the basic version of RSVP (see Section 5.1.2 for details). The goals of these extensions are mainly to complete RSVP's specification in the areas of security and reliability and furthermore, to improve certain characteristics which are identified as currently limiting overall performance. On the other hand, little attention has been paid to the implementation of the core protocol engine itself. As a result, RSVP is often assessed as having a poor performance, however, those judgments are usually not based on reasonable and solid data. Therefore, the internal design structure and algorithms, as well as the overall protocol performance, have been subject to careful investigation in this work.

The chapter is structured as follows. First, existing work is described and evaluated with respect to the goals of this thesis. Then, an innovative design and implementation of an RSVP engine is described, which forms one of the main contributions of this thesis. Certain improvements over previous work are discussed in detail in a separate section and the current implementation status is briefly presented. Finally, performance tests are described and their results discussed in order to gain insight about the technical feasibility to employ RSVP signalling in a large-scale scenario.

5.1 Existing Work

Apart from the work that is described in this thesis, there is only one publicly available implementation of RSVP, namely the *ISI rsvpd* package from USC ISI [ISI99]. The ISI rsvpd package is regarded as the reference implementation, and it is also the basis for numerous other versions for different platforms [GF98]. This package is very complete and useful, however, it does not appear to be the optimal platform for examining and testing protocol characteristics or even modifications for anyone not belonging to the original developer team. This assessment stems from the following observations:

- Because of its innovative nature, it is likely that the initial implementation design has been done with only little experience of the protocol. The code grew historically and it seems that no significant design reshaping has been done.
- Throughout the implementation, design and coding style is that of system-level C code.
- Portability is aggravated because system-dependent code (e.g. system calls) is distributed across the implementation.
- The implementation is distributed without design documentation relating general concepts to implementation details. The only documentation available [BZ97] is outdated and contains errors. A new attempt to describe message processing [LBZ99] is incomplete and has not been updated so far.

Furthermore, this software package is currently not suitable for performance testing, because it contains several fatal bugs, which basically prohibit testing scenarios which involve the deletion of multiple sessions. An investigation of this problem revealed at least one non-trivial error in the memory management to be responsible for this situation. It is quite easily possible to fix the most prevalent problem, such that the software does not crash too often, but as a result, memory leaks prohibit reasonable operation.

5.1.1 Performance Evaluation

Some work has been reported to assess the performance of commercial RSVP implementations. In [NCS99], a technical framework for carrying out such tests is presented. The performance figures for a midrange commercial router show that it cannot deliver the delay objective for 450 flows requesting a small bandwidth. Furthermore, from the numbers given in this paper, it can be deduced that RSVP flow setup scales significantly worse than linear. These results indicate that this implementation is in a rather early and immature development stage.

Other published work describes the implementation of an RSVP-capable switch-router in [BBD+99], but the reported performance figures are targeted towards the fundamental capability of the system to deliver QoS objectives in the first-place, rather than performance of the signalling capabilities in a large scale.

In [PS99], interesting performance figures are reported for RSVP message processing on a commercial router platform. However, these performance figures are somewhat debatable, because it is not mentioned under which load conditions they were taken. Additionally, because these numbers are not the central focus of the work in [PS99], not many details about the exper-

iments are given. Because of these observations and because the code which was used is not publicly available, these numbers can serve as a basic indication about RSVP's processing overhead, but they cannot be considered to be the final statement in the discussion about RSVP.

5.1.2 Proposed Improvements

A number of protocol improvements have been suggested to increase the performance characteristics of RSVP operations. An initial proposal to speed up the service establishment time in the presence of occasional packet loss has been made in [PS97]. One of the problems with this approach is the requirement to introduce a reliable confirmation message into RSVP. An improved approach has been described in [MHSS99], which also deals with the general issue of reliability of RSVP messages, e.g., in case a service invocation is torn down. Instead of refreshing all the state information, neighbouring RSVP nodes only need to exchange 'heartbeats' denoting their liveness. A slightly different suggestion addressing the same issue even more precisely is currently developed within the IETF RSVP working group [BGS+00]. This mechanism addresses further details, such as how to discover a very short-timed node failure.

It is beyond the scope of this thesis to rate these different techniques, however, it can be clearly deduced that they create the potential to drastically reduce RSVP's processing requirements, especially for steady-state refresh signalling. This eliminates one of the major disadvantages of the current RSVP specification.

Other RSVP extensions, which are in the process of being standardized, encompass diagnostic messages [TBVZ00], inter-operation with IP tunnels [TKWZ00], cryptographic authentication [BLT00] and user identity representation [YYP+00].

5.2 Improved Design of the Protocol Engine

Because of the generally limited software quality of the only available free implementation, its questionable internal structure and the restricted value of the few available published performance figures, it was decided to develop a new protocol engine from scratch obeying the following design goals:

- structured (object-oriented) design and implementation
- portability for multiple platforms, including simulators
- clear and concise representation of RSVP's concepts in the code
- suitability for performance testing

Another explicit goal for this work was to create and publish an experimental platform which allows other researchers to test and examine modifications of RSVP with a reasonable effort. This implementation is called *KOM RSVP engine* [Kar00b], or *KOM rsvpd* for short.

While at a first glance RSVP seems to be straightforward and easy to understand, the details of an implementation are somewhat puzzling. One reason for the slow acceptance of RSVP might actually be given by the very limited amount of published work on implementation internals, leaving room for myths and rumours about very high implementation complexity and poor performance.

In this section, an innovative design for an RSVP engine is presented together with an overview of the corresponding message processing rules. The design is based on relational design and object-relationships between state blocks. A rigorous approach for modelling RSVP would begin by representing state information as relations and identifying functional dependencies between them. Then, well-known normalisation algorithms could be applied to create the highest possible normal form and message processing could be expressed using relational algebra. Intuitively, this is often done to some extent by software designers and programmers.

While not following the strict method in this work, state information is explicitly modelled as relations which in turn are considered as state blocks to create object-relationships between them. The initial relational model is deduced from the relevant standardization documents [BZB+97,BZ97,LBZ99] and personal reasoning about the protocol. Additionally, experiences made during design and implementation of the software have been a source of insight into protocol operations. The details of relational representation are omitted here for reasons of brevity and can be found in Appendix B. Some basic relations are equivalent to the respective RSVP objects as specified in [BZB+97] and listed in Table 2.

Table 2: Mapping of RSVP Message Objects to Basic Relations

RSVP message object	Relation
`SESSION`	*SessionKey*
`FILTER_SPEC`	*Sender*
`RSVP_HOP`	*Hop*

5.2.1 Conceptual Design

A significant part of RSVP message processing consists of finding appropriate state blocks for certain operations. For a normal implementation (i.e. without using a dedicated database for state information), state blocks and object-relationships can be considered to be more expressive and efficient than directly implementing the relational model. The relationships between objects are explicitly stored when knowledge is available, instead of recalculating them through relational rules whenever they are needed. The algorithmic description in [BZ97,LBZ99] exhibits a relational style, but without being rigorous. Opposite to that approach, the processing rules in this work are based on object-relationships between state blocks. A subset of state blocks is similar to those described in [BZ97,LBZ99], but semantics and lifetime are occasionally modified. Additional relations are designed to express useful state information. Eventually, these are represented as state block objects as well, to efficiently accomplish protocol operations as outline above.

State Blocks - Overview

State information is stored as objects containing relationships to other objects. The contents of a PATH message are stored in a *Path State Block* (PSB) whereas contents of a RESV message are stored in a *Reservation State Block* (RSB). As an example for relationships, each PSB has a relationship to a *Previous Hop State Block* (PHopSB) representing the hop from which this PATH message has been received. Information concerning a reservation at an outgoing interface is stored in an *Outgoing Interface State Block* (OutISB) and the relationship between reservation state and PSB objects is modelled as separate object *Outgoing Interface at PSB* (OIatPSB) in order to internally represent an N:M relationship by 1:N relationships (which simplifies structure and implementation). This design introduces a more detailed representation of state information, by introducing the separate state blocks PHopSB and OIatPSB.

State Blocks - Details

From the initial relations, corresponding state block objects and state block relationships are deduced. Although this is not done rigorously (i.e. by using normalisation algorithms), certain optimizations have been applied to avoid redundancy of information and to suit an efficient implementation. The result can be expressed as an entity-relationship model and is shown as diagram in Figure 7. As shown in the diagram, all entities except Session are weak entities, i.e., they cannot uniquely be identified without the respective session key. Furthermore, OutISB is

indirectly identified through the set of OIatPSB objects it is related to, although the cardinality ratio implies the opposite direction. In RSB, there is no information about the outgoing interface stored and the list of senders is not used for identification. Thereby, it is a weak entity depending on the key of OutISB. For both RSB and OutISB, instead of storing the set of applicable senders, a relationship to PSB is maintained (indirectly in case of OutISB). The cardinality ratio of each relationship is shown in the diagram in Figure 7. A brief description of each state block is given below.

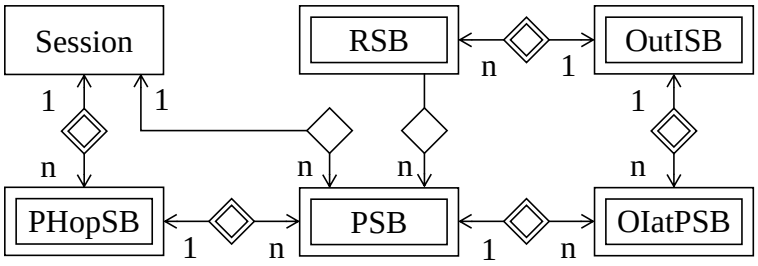


Figure 7: Entity-Relationship Diagram for State Blocks

Session. For each RSVP session, the Session state block bundles all relevant information and the session's destination address and port is saved there. Relationships are kept to those state blocks that are needed to fully access all information during early stages of message processing. All Session objects are identified by a SessionKey and bound to a single RSVP object, representing an RSVP router.

Path State Block (PSB). A PSB holds all relevant information from a PATH message, i.e., the sender's address and traffic specification, routing information, etc. It also stores blockade information, which is needed to alleviate the effects of RSVP's so-called *killer-reservation* problem [BZB+97].

Reservation State Block (RSB). An RSB represents a reservation requested from a next hop, particularly by holding the reservation specification, i.e., the FlowSpec, which determines the amount of resources that are requested, depending on the service class. It is identified by its outgoing interface, available from OutISB, and the next hop's address.

Outgoing Interface State Block (OutISB). This state block represents the merged reservations from multiple RSB objects applying at a certain outgoing interface. It is roughly comparable to the *Traffic Control State Block* (TCSB) in [BZ97,LBZ99]. However, different from

those processing rules, the TCSB is split into a general (OutISB) and a specific part in this new design. The nature of the specific part depends on the particular traffic control module, which in turn depends on the corresponding link layer medium behind the network interface [BZB+97, SA98a,SWKS99]. An instance of OutISB is constructed immediately upon creation of the first contributing RSB.

Outgoing Interface at PSB (OIatPSB). For each outgoing interface that is part of the routing result for a PATH message, an instance of OIatPSB is created. A relationship to an OutISB object expresses that an actual reservation is active at this interface. The introduction of this state block allows the split of the N:M relationship between PSB and OutISB into two 1:N relationships.

Previous Hop State Block (PHopSB). The concept of an explicit PHopSB is new to an RSVP description. It is used to hold information about reservations that are merged at a certain incoming interface towards a previous hop, as well as the resulting reservation request that is sent to this hop. A PHopSB is identified by the previous hop's IP address and the incoming interface, at which traffic from this hop arrives for the destination address of a session. Analogous to OutISB and RSB, a PHopSB object is created as soon as the first PATH message arrives from a certain previous hop.

Relationships

Each relationship is presented below, including the necessary key to traverse it, if it is a multi-object relationship. The respective keys applying to these relationships are often smaller than the full key of each state block. This is due to inherent identification through the relationships. However, the implementation of this model is done by directly storing the relationships. Furthermore, all Session objects are bundled into a global container. This could be considered as a special relationship to a unique object representing the RSVP router. An illustrative example for relationships between state block objects relative to the data flow belonging to a single RSVP session is presented in Figure 8.

Session $\leftarrow_{1}\!\diamond\!\rightarrow_{n}$ **PSB.** key: Sender, Incoming Interface and Previous Hop (R1)
In a PSB object, information about incoming interface and previous hop are not stored directly, instead this information can be extracted from the corresponding PHopSB (see (R3) below).

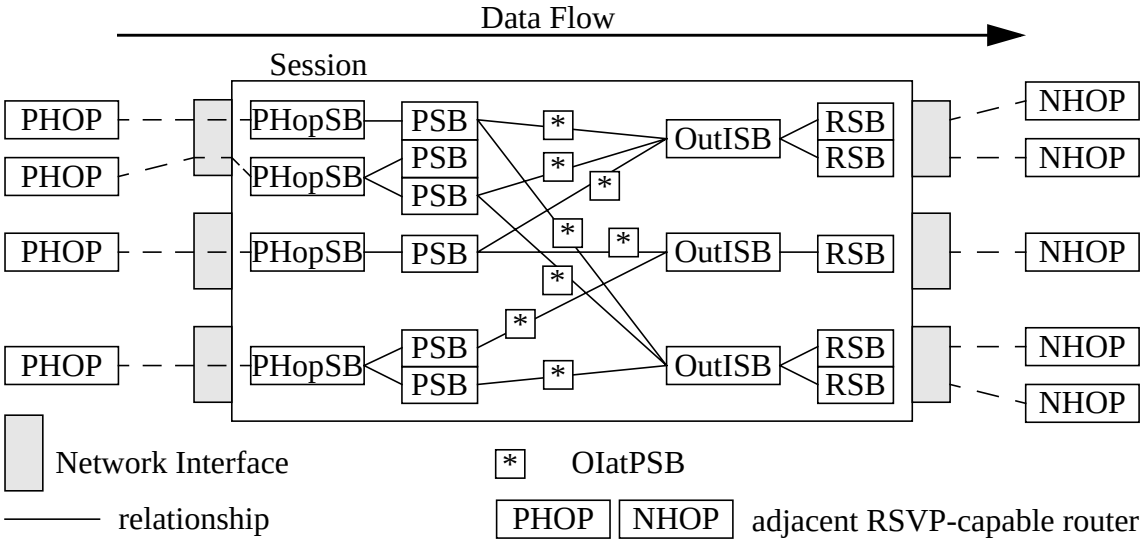


Figure 8: Example for Relationships between State Blocks in a Session

Session $\leftarrow_1 \diamond_n \rightarrow$ **PHopSB.** key: Incoming Interface and Previous Hop (R2)

While it seems redundant to store relationships from Session to PSB and PHopSB, both relationships are needed in order to efficiently deal with routing changes and process a PATH refresh message that arrives from a different previous hop. Details are given below.

PHopSB $\leftarrow_1 \diamond_n \rightarrow$ **PSB.** key: Sender (R3)

Each PSB is logically connected to the PHopSB representing its previous hop. This relationship is mainly used when reservation requests are created for previous hops. Information obtained through this relationship from the PHopSB (hop address and incoming interface) is used to distinguish PSB objects in (R1). In PHopSB, the merged FlowSpec of all PSBs from this previous hop is maintained in case of a shared reservation style.

PSB $\leftarrow_1 \diamond_n \rightarrow$ **OIatPSB.** key: Outgoing Interface (R4)

This relation expresses that a PATH message is routed to a set of outgoing interfaces. In case of unicast sessions, it is effectively reduced to a 1:1 relationship. In PSB, the shared flowspec of all applicable reservations is maintained.

OutISB $\leftarrow_1 \diamond_n \rightarrow$ **OIatPSB.** key: Sender (R5)

A merged reservation, installed at an outgoing interface, applies to a set of senders. This is expressed by storing relationships to the respective OIatPSB objects, instead of directly storing information about all applicable senders. In the other direction, this relationship in combination

with (R4) expresses the existence of a reservation at a certain outgoing interface for a particular sender.

OutISB $\leftarrow\!\diamond\!\rightarrow_{1\ n}$ **RSB.** key: Next Hop (R6)

Each OutISB is related to those RSB objects that are merged together at an outgoing interface. Because an RSB only contributes to one specific OutISB, partitioning the set of RSB objects along all OutISB objects creates a complete and disjoint decomposition. Because the information about the applicable outgoing interface is always available for an RSB, a relationship between Session and RSB is not necessary to access RSB objects from a Session object.

RSB $\rightarrow\!\diamond\!\rightarrow_{n}$ **PSB.** key: Sender (R7)

A reservation applies to a set of senders, either by explicit selection (SE or FF filter style) or implicit association (WF filter style). Instead of storing a list of all sender addresses, a relationship to the respective PSB objects is maintained from the RSB.

Operations Overview

In the following paragraphs, the core operations of the RSVP engine are explained with respect to the relationships between state blocks. The presentation is divided into four parts, which together form the central protocol operations:

- State Maintenance
- Outgoing Interface Merging
- Incoming Interface Merging
- Timeout Processing

In general, if a message or timeout triggers a modification of internal state, all relationships are updated immediately during State Maintenance or Timeout Processing, except the relationship between OIatPSB and OutISB. For Outgoing Interface Merging, the ‘old’ state of this relationship has to be available to appropriately modify the filter setting at the underlying traffic control module. Afterwards, this relationship is updated, as well. If the contents of an RSB change, Outgoing Interface Merging is invoked. If during Outgoing Interface Merging, the contents of an OutISB are changed, Incoming Interface Merging is triggered. Only if the resulting upstream reservation request (stored in PSB/PHopSB) changes, a new RESV message is created and sent to the previous hop immediately.

The basic claim of this work is that maintaining relationships imposes no significant additional overhead during analysing an incoming message and updating state from it. However,

when it comes to merging reservations and timeout processing, existing relationships can be exploited instead of recomputing them every time, especially under stable conditions. In this section, only the basic operations are described. Whenever the term *interface* is used in the following text, it might also denote a local connection to an instance of the *Application Programming Interface* (API), i.e., a local application communicating with the RSVP engine.

State Maintenance

Arriving RSVP messages are decomposed into components and processed depending on the type of message. During processing, appropriate state blocks have to be located, created and/or modified and relationships between them have to be updated. Below, a pseudo-algorithmic description of the processing rules is given for each message type. Although it is not mentioned explicitly for most of the message types, usually the appropriate Session object has to be determined first.

PATH. Find a Session and check for conflicting destination ports. If no Session exists, create one. Find a PSB for this sender through (R1) and check for conflicting source ports. If none exists, create a new PSB. When creating a PSB object, create the relationship to the corresponding Session object (R1). If the PSB is new and no appropriate PHopSB can be found, create a new PHopSB and its relationship to Session (R2). Set a relationship between PSB and PHopSB (R3). If the session address is multicast and the incoming interface differs from the routing lookup result, mark this PSB as local to an API session. Update all information in the PSB and in case of relevant changes, trigger an immediate generation of a PATH message and potentially invoke Outgoing Interface Merging.

RESV. Process each flow descriptor separately, i.e., each pair of FilterSpec and FlowSpec. Match (i.e. consider the intersection of) the filter specification (in case of FF or SE) or the address list determining the scope (WF) against all existing PSB objects that route to the outgoing interface through (R1). Find or create an appropriate OutISB for one of the resulting PSB objects through (R4) and (R5) and find or create an RSB using (R6). When creating new objects, set the corresponding relationships. Update the RSB and invoke Outgoing Interface Merging, if relevant content has changed, i.e. FilterSpec or FlowSpec.

PTEAR. Find a PSB through (R1). If found, forward the message to the PSB’s outgoing interfaces, remove the PSB, clear its relationships and invoke Outgoing Interface Merging.

RTEAR. Process each flow descriptor separately. After looking up the matching PSB objects (R1), find an RSB through (R4), (R5) and (R6). If found, remove the filters that are listed in the message and invoke Outgoing Interface Merging. If the RSB's filter list is empty, remove the RSB and clear its relationship to OutISB.

PERR. Find a PSB through (R1). If found, forward the message to the previous hop, which is accessible through (R3).

RERR. Find a PHopSB for the previous hop address from the message through (R2). If found, find all PSB objects that match a filter from the message through (R3). If the error code indicates an admission control failure, set a blockade FlowSpec at those PSB objects. Find all OutISB objects that have a relationship to any of the PSB objects through (R4) and (R5) and do not belong to the incoming interface of the message. Forward the message to all next hops which are represented by RSB objects that have a relationship (R6) to these OutISB objects. In case of admission control failure, forward the message to only those next hops which sent a reservation containing a FlowSpec not strictly smaller than that of the received error message.

RCONF. Forward the message to the outgoing interface that results from a routing lookup for the message's destination address.

Outgoing Interface Merging

During the merge operation at an outgoing interface, all applicable PSB and RSB objects have to be collected to access their TSpecs and FlowSpecs. Precise operation depends on the nature of the underlying link layer and appropriate algorithmic descriptions can be found for point-to-point or broadcast media in [BZ97,LBZ99] and for non-broadcast multi-access media (e.g. ATM) in [SA98a,SWKS99]. Outgoing Interface Merging operates on a certain OutISB. Relationships to those PSB objects that are relevant and route to this interface as well as RSB objects that contribute to the merged reservation state are given by (R4), (R5) and (R6). They can be traversed directly, instead of recomputing them. Therefore, no special (filter style dependent) rules have to be given on how to find those state blocks. The rules to calculate the appropriate merged FilterSpec and FlowSpec settings are given in [BZ97]. The result is stored in the OutISB and, if the merged FlowSpec or the FilterSpec has changed, the appropriate PSB and PHopSB objects (accessible through (R4), (R5) and (R3)) are marked for Incoming Interface Merging. Certain policing flags have to be passed to traffic control, which can be derived from accessible information, as well. To determine whether this reservation is merged with any other

reservation that is not less or equal, the least upper bound of all merged FlowSpecs from all OutISB objects (at different interfaces) for all PSB objects can be calculated by traversing (R4) and (R5). If at the end of Outgoing Interface Merge the OutISB has no relationship (R5) to any OIatPSB object (which must coincide with having no relations (R6) to RSB objects), the OutISB is removed. Further details about the invocation of traffic and policy control modules are discussed in Section 5.3.1.

Incoming Interface Merging

After a single or multiple (in case of RESV message processing) invocations of Outgoing Interface Merging, all PHopSB and PSB objects that are marked for update are subject to Incoming Interface Merging. During this sequence, it is again possible to traverse relationships, instead of collecting state blocks. The details of this merging operation depend on the filter style for the session. In case of distinct reservations (FF), each marked PSB that relates to the PHopSB is considered separately. All OutISB objects accessible through a PSB through relationship (R4) and (R5) are inspected and the FlowSpecs of all RSBs accessible through (R6) are merged. A flow descriptor is created, containing the PSB's sender address and the merged FlowSpec. For shared reservations, all FlowSpecs of all RSB objects accessible through (R4), (R5) and (R6) from any of the PSB objects are merged. The resulting flow descriptor contains the set of all sender addresses and the single merged FlowSpec. In case of relevant changes, a RESV message is created and stored at the PHopSB. The PHopSB manages the periodic refresh of this message.

Timeout Processing

According to the soft state paradigm, each state block is associated with a timer and deleted upon timeout. Periodic refresh messages restart the timer. Timers are directly connected to the object they apply to and the actions resulting from a timeout are similar to those when receiving a PTEAR or RTEAR message. The only difference is, that no PTEAR or RTEAR message has to be forwarded.

5.2.2 Software Design

Given the objectives of the project, the following goals have been set for an implementation:

- Message handling (creation/interpretation) should be clear, simple and extensible.
- Protocol processing should be clear and comprehensible, yet efficient.
- The implementation should be portable, but also nicely integrate with system interfaces.

The design that has been chosen is a hybrid form of object orientation and procedural design. Object orientation does not seem to be fully appropriate for implementing state machines like network protocol engines, however, many aspects of an implementation can benefit from data encapsulation, inheritance and polymorphism. C++ has been selected as the programming language of choice to implement such a hybrid design under the given objectives. In the following description, identifiers stemming from the implementation are printed in italic, when they are introduced. Figure 9 gives an overview of the design.

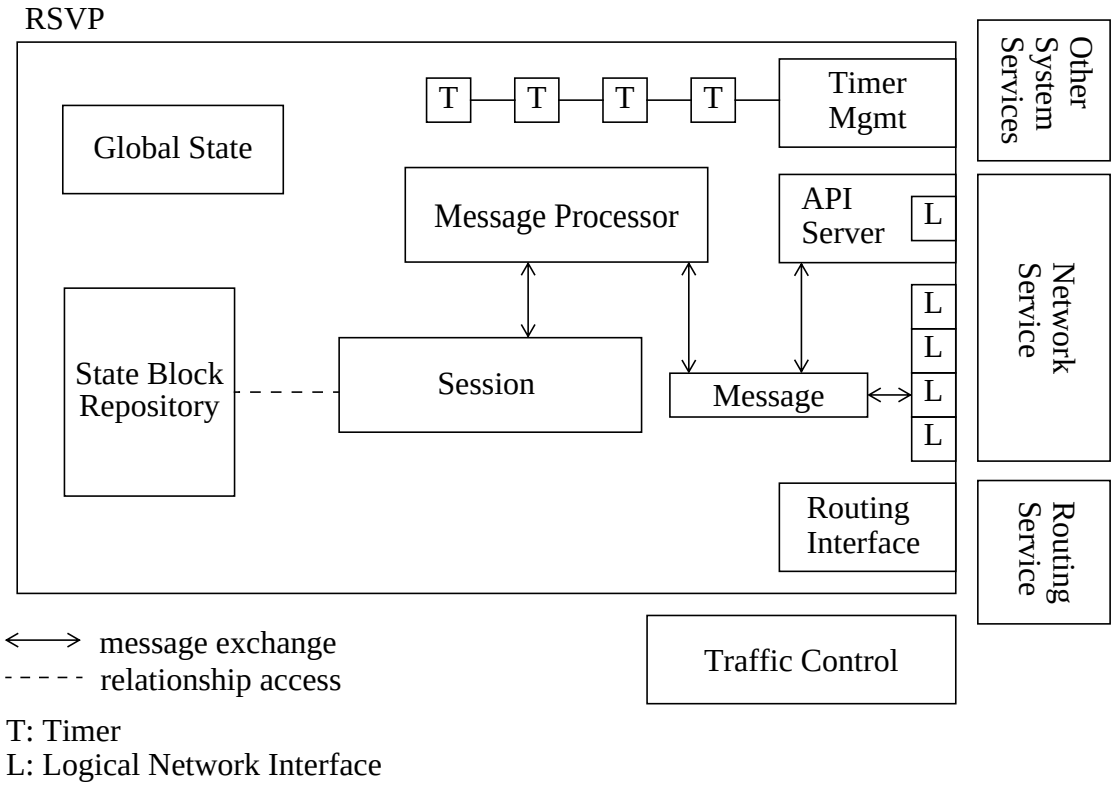


Figure 9: Conceptual Design of RSVP Implementation

In this picture, the main components, which together form the contents of a global *RSVP* object, are shown. An RSVP object represents an RSVP-capable router and interacts through abstract interfaces with system-dependent services like routing, network I/O, traffic control and others. The primary handler for incoming messages and events is denoted as *Message Processor*. Multiple *Session* objects exist, representing currently active RSVP sessions. The *State Block Repository* is an abstract notation for all state block objects which are accessible through the relationships, initially starting from Session. A number of *Logical Interface* objects encapsulate physical and virtual interfaces of the underlying system. Logical interfaces are numbered

and the number is used as LIH (Logical Interface Handle, see [BZB+97] for details). The association with API clients is modelled as a dedicated container object, called *API Server*, containing a special instance of a Logical Interface and all information about currently active API clients. RSVP messages are encapsulated in a *Message* class and passed between Logical Interface, Message Processor and Session objects, potentially involving API Server. Global state is accessible from all other parts and kept separately in the RSVP object, for example, the current message, a PHopSB refresh list, etc. Solid arrows denote the exchange of RSVP message objects and the dashed line represents relationship traversal. Other inter-object communication is not shown.

Message Processing

Each incoming message arrives at the main RSVP object. It is passed to the Message Processor, which carries out preprocessing and updating global state. Afterwards, the message is dispatched to the appropriate Session object for further processing. The initial part of State Maintenance (e.g. finding or creating a Session object) and the majority of Incoming Interface Merging is carried out in the Message Processor object. Another significant part is implemented in class Session. Merging at an outgoing interface is link-layer dependent and consequently, functionality is split up. Further details are given in Section 5.3.1. Incoming Interface Merging takes place when reservation state has changed, that is, if FlowSpec or FilterSpec of an OutISB has been modified. It is implemented in class Message Processor.

Implementation Details

Some specific implementation details are explained below to illustrate the fulfilment of the initial design goals. As a general note, an RSVP implementation on a regular UNIX workstation can only serve as a proof of concept and research platform for future investigations. Therefore, although the design is kept prepared for efficient operation, it is not necessary to implement for outmost efficiency.

Relationship Representation. Relationships are implemented as a set of classes from which state object classes inherit. These relationship classes automatically maintain referential integrity. A single-object relationship is internally represented by a pointer or reference, whereas a multi-object relationship is internally represented by a sorted container of pointers to the respective objects. The order is determined by those object attributes forming the relational key.

Timers. Timer management is logically separated from the rest of the implementation, such that it can be independently optimized without considering other parts of the code. A base class *BaseTimer* exists, from which refresh and timeout timers are derived. They are controlled by their owners, but handled commonly through BaseTimer. In the basic version, all timers are kept in a container, ordered by their expiration time. This design completely hides implementation details between timer management and timer clients.

Container Classes. A simple container library for lists and sorted lists has been implemented, in a style similar to the C++ Standard Template Library [ISO98]. While it is conceptually very advantageous to use common container classes, it seems not necessary to provide the most efficient implementation for them. It is left to the user of this implementation to decide whether outmost efficiency is required when accessing certain containers or not. Because of the encapsulated design, testing of different algorithms and data layouts for containers is possible with relatively low effort.

For example, the container of (R4) is internally realized as an array of pointers, because at most one OutISB exists at each interface and the maximum number of network interfaces is limited, fixed, and can be determined at start-up of the RSVP daemon. The container for the timer system is implemented as a two-layer hierarchy, where its upper layer is given by a an array representing time slots and the lower layer consists of a sorted list of timers belonging to each slot. The session container is implemented in a similar manner, except that the upper layer is accessed through a simple hash function defined on the session's destination address. The size ratio between upper and lower layers of these containers are configurable, hence they allow for trading off memory requirements against processing effort.

5.3 Internal Design and Algorithmic Improvements

Besides developing a new general design for an RSVP engine, a number of internal design and algorithmic improvements have been carried out to generalize RSVP operations and improve message processing performance. These improvements are described below.

5.3.1 Generalized Interface to Traffic Control and Policy Control Modules

The initial specification of RSVP lacks two aspects, which are important for real employment as general signalling mechanism. The interface to traffic control modules, which eventually control and enforce resource reservations, has been specified without taking into account the

specific properties of non-broadcast multiple-access (NBMA) networks such as native ATM. Additionally, no interface to policy control modules has been specified. While the general operation of RSVP in the *Common Open Policy Service* (COPS) framework is described in [HBC+00,Her00a], this does not cover any implementation internals. A policy control module is needed to make and enforce administrative authorizations to use certain resources. As well, inter-operation with a charging system has to be carried out by this module. Several new aspects arise when broadening the point of view on traffic control by NBMA networks and policy control.

Silent Next Hops. Consider the arrival of the first RESV message from a downstream RSVP hop. Suppose that a reservation is already in place at the respective outgoing interface and that the new request carries no new FilterSpec and a FlowSpec which is strictly smaller than the existing one. If NBMA subnets and policy control modules are not considered at this point, no traffic control operation is necessary, because the new request can be served by the existing reservation. However, in case of NBMA networks, a new reservation request conveys a new next hop. This information must be handed over to the traffic control module, because it might be necessary to establish a dedicated transmission channel (e.g. a VC or VC-endpoint in case of ATM) to it. Also, a policy control module that calculates charges and accounts them to next hops must be informed about such a change.

IP Multicast. The interface to a traffic control module of RSVP is specified in [BZB+97]. With respect to IP multicast, it is mentioned in this document that the description “assumes that replication can occur only at the IP layer or ‘in the network’”. This can be denoted as a *broadcast* network. Note that a point-to-point link can be considered as special type of broadcast network. As has been extensively discussed, e.g. in [SA98b] and [CBB+98], there are many aspects of efficiently overlaying RSVP and ATM networks, which mainly result from the NBMA characteristics of ATM and the fact that ATM does not directly support the highly flexible IP/RSVP multicast model. Without extensions, an RSVP engine merges all requests arriving at a single outgoing interface by calculating the least upper bound of all FlowSpecs. In case of NBMA networks, however, the traffic control module itself must be able to decide how to merge reservations. The concept of *merging groups* can be used to express this capability [SA98a]. Because ATM does not support multicast-VCs with heterogeneous QoS parameters, the traffic control module partitions the set of next hops according to the similarity of their QoS

requests into merging groups. Then, a multicast-VC is used for transmission to each merging group. Algorithmic aspects of efficiently building merging groups are studied in [SWKS99].

Atomicity of Operation. Both traffic control and policy control module independently decide about acceptance of a reservation, based on their respective state and configuration. Each operation must be done atomically, i.e., having an all-or-nothing property. Furthermore, for the core RSVP engine, complete acceptance or rejection of a reservation must appear as a single decision, because RSVP has no mechanisms to deal with a reservation that is accepted by only one of both modules. Consequently, admission of a reservation request must be done in one atomic operation from RSVP's point of view. An open issue is to determine which of traffic control and policy control module decides first about admission and which is second. That module doing the first decision must be prepared for a full rollback if the other decision fails. In this work, it has been decided to place this burden on the policy control module for the following reasons.

It is likely to assume that policy control decisions generally consist of an authorization, a validity check and an accounting step. Validity check and accounting might be omitted, if the network operator considers it unnecessary. The validity check might be a test whether the offered payment is sufficient. This in turn requires part of the accounting process to be carried out. 'Raw' resource accounting might be followed by an internal transaction, e.g., debiting an internal account. Internal transactions are periodically cleared by external transactions, i.e., real payments. Because an update of traffic control parameters immediately results in different network conditions, affecting other flows as well, it is favourable to lower the probability of a traffic control rollback over a policy control rollback. In case of a policy control rollback, internal transactions can be reverted without influencing any external entities.

Additionally, from the diversity of subnet technologies and their potential for complexity (e.g., in case of ATM or when employing a *Subnet Bandwidth Manager* [YHB+00]) it can be concluded that the effort for rollback preparation and potential resource wastage makes a case for this design decision. Yet another reason can be given by considering network provisioning. In a well-dimensioned network, traffic control rejections can be expected to be less likely than policy control refusals (e.g., because of overdue bills or empty prepaid billing cards). The same strategy has been chosen in the proposal for interaction between RSVP and COPS [Her00a]. In the following three paragraphs, certain challenges and approaches to them are presented, which result from these considerations.

Partial Rollback of Traffic Control. The scope of a single traffic control update operation is defined by the handling of a single RSVP message or timer expiration per interface. Performing such an update operation consists of several actions to be carried out. Besides installing a new FlowSpec for a reservation, FilterSpecs to identify eligible sender applications might be added or removed. Although unlikely, it is possible at least for a filter insertion operation to fail. Therefore, the traffic control modules are prepared for partial rollback by carrying out the update operation as follows:

- First, the new reservation FlowSpec is installed, then filters are added or removed. Note that the above definition of a scope for a single operation prevents that filters are added and removed within the same update operation. As well, when filters are removed, the installed FlowSpec is never increased.
- If installing a new FilterSpec fails, all previously installed FilterSpecs from this update operation are removed again and the FlowSpec is set to its previous value. Thus, the important all-or-nothing property of a traffic control update operation is guaranteed by internal partial rollback. By appropriately designing the respective software interface (see below), a traffic control module for NBMA networks can easily be integrated into this process to revert any merging group operations (as discussed above).

Full Rollback of Policy Control. In order to integrate a policy control module that has the capability for rollback, the interface has to be split into two parts.

1. The preparation step consists of authentication and validity check. As a result, the request is either accepted or rejected and temporary state is saved within the policy control module.
2. The commit step corresponds to accounting, i.e., handing over the state information for persistent storing and potential external transactions.

If a reservation is accepted and later rejected by the traffic control module, it is sufficient to delete all temporary state information.

Concurrent Execution. Both traffic control and especially policy control operations might involve a certain overhead, so that it seems desirable to execute them concurrently to the core RSVP operations. This however, requires to specify most of RSVP's state information and operations such that concurrent execution is possible. As presented in Section 5.3.3, extending the implementation design for concurrent execution is not too hard.

Interface Design

The traffic control and corresponding modules are modelled as a class hierarchy forming four layers of abstraction. Common tasks are implemented in higher layers whereas more specialized task are implemented in derived classes. A class diagram of this design is given in Appendix C. A common base class *TrafficControl* exists, which provides the main interface to the core RSVP engine. The following interface presentation is restricted to the most relevant methods without showing details like arguments and return types.

```
class TrafficControl {
     virtual updateReservation() = 0;
     virtual redoLastReservation() = 0;
     updateFilters();
     addFilter();
     removeFilter();
     updateTC();
};
```

This base class completely implements the high-level handling of insertion and removal of FilterSpecs. Whenever during message processing a FilterSpec is found eligible for insertion or removal, a call to *addFilter* or *removeFilter* respectively is made. In order to minimize interaction between traffic control module and the system's resources, these actions are buffered within the *TrafficControl* class and executed only when *updateFilters* is called. Common merging logic is implemented in the method *updateTC*. This calling interface takes an instance of OutISB as input parameter. All state that is needed for admission control and updating of the underlying scheduling system is then accessible through relationships starting at OutISB. The methods *updateReservation* and *redoLastReservation* are realized in derived classes and implement the logic for merging of multiple reservations. They are specialized on broadcast or NBMA respectively, depending on the actual type of subnet an interface is attached to. Correspondingly, two classes are derived from *OutISB*: *TCSB_BMA* and *TCSB_NBMA*. Internal state information for a reservation at an outgoing interface is stored in these classes. For example, merging group information for NBMA subnets is stored in objects of type *TCSB_NBMA*.

A separate class *Scheduler* acts as a base class for different flavours of scheduling packages and provides a common interface to them. This interface is basically the same as the traffic control interface in [BZ97]. The public methods of class Scheduler are eventually realized by calling internal virtual methods, which in turn are implemented in derived classes. Furthermore,

this class provides some common mechanisms like logging of events and high-level admission control.

The interface to policy control includes the necessary methods to perform a policy control update in two steps with a potential rollback after the first one. Given that the general design for the protocol engine allows for traversal of object-relationships, it is suitable for those methods to take similar arguments as the traffic control interface. All operations that are carried out when *commit* is called, can be executed concurrently to further RSVP operations.

```
class PolicyControl {
    checkAndPrepare();
    commit();
    rollback();
};
```

Structure of Operation

In order to glue the pieces together, the following pseudo code describes how *updateTC* from class *TrafficControl* utilizes the services of other objects. As can be seen from this pseudo-code, the appropriate design of traffic control and policy control modules and interfaces leads to a very concise and elegant expression of high-level concepts.

```
PolicyControl::checkAndPrepare();
if (success) {
    updateReservation();
    if (success) {
        updateFilters();
        if (success) {
            PolicyControl::commit();
            return;
        }
        redoLastReservation();
    }
    PolicyControl::rollback();
}
```

5.3.2 Fuzzy Timers

By far the largest container in an RSVP implementation is necessary for timer handling. In this implementation, all timers are stored in a hierarchical container. The upper layer is implemented as an array representing time slots and accessed through a hash. The lower layer is implemented as a sorted list. The configuration of this hierarchy, i.e., the amount of time covered by each slot, can be set arbitrary. Such a container is only capable to foresee a limited amount of

time in the future, which should sufficient for RSVP. In order to accommodate the rare event that timers exceed this time frame, an additional sorted list is kept and timers from this list are moved into the respective slot when it becomes available. This concept is known as a *timer wheel* [VL87]. The best possible access complexity of such an implementation using standard hardware is *O(log(n))*, with *n* being the (varying) number of timers in a slot. Consequently, performance of this container can be arbitrarily traded off against memory requirements by choosing the size and number of slots. This data structure design is shown in Figure 10.

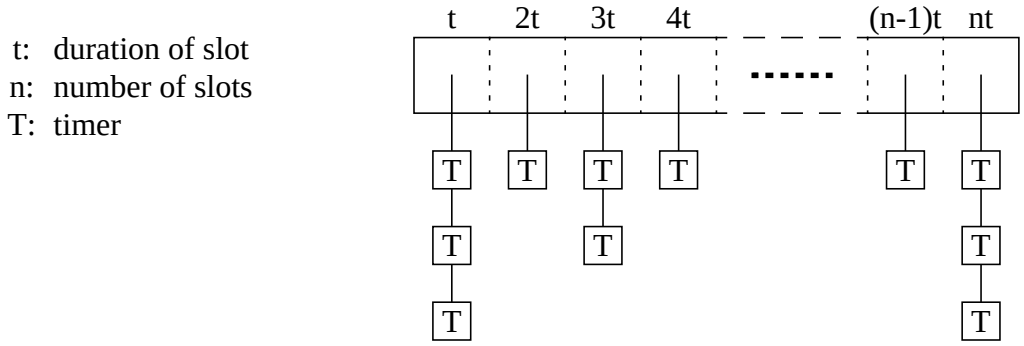


Figure 10: Design of Timer Container

For RSVP messages, this scheme can be optimized even further. RSVP is designed to be robust against varying message transmission times and in fact, a large number of all timers are calculated as random numbers within a certain interval. As a consequence, there is no demand for outmost precision in the scale of a few milliseconds. If the duration of a time slot in the hierarchy becomes small compared to the basic refresh time (e.g. smaller than 100 microseconds when the basic refresh interval is set to 30 seconds), an option to employ *fuzzy timers* is implemented. When enabling it, timers are stored in a simple FIFO list instead of being sorted according to their precise expiration. During each time slot, timers are fired equally distributed according to their location in the simple list. The result is a slight inaccuracy of timers compared to their expiration time. The inaccuracy is bounded by the length of a time slot and can be considered a very reasonable trade-off. This scheme promises a significant performance gain over the plain timer wheel, because the access complexity is reduced to *O(1)*. However, as presented and discussed in Section 5.6.4, only a limited performance gain has been observed so far, on top of what is achievable by tuning the normal timer wheel implementation.

5.3.3 Multi-threaded Message Processing

Employing the new design of the RSVP engine, it was possible to quite easily replace the initial sequential message processing by a multi-threaded protocol engine with an incremental implementation effort of about 4 weeks. The design tries to mimic operation in a high-speed router, such that for each network interface a dedicated thread for message processing is created. Recapitulating the software design from Section 5.2.2, the class Message Processor is combined with a thread of control and an object is created for each network interface in the system. Because of a current lack of system support, certain interactions with the operating system, e.g. the reception of raw IP packets, cannot be performed truly multi-threaded. Therefore, those operations are currently carried out sequentially. As a consequence, in addition to the threads per network interface, there is a dedicated thread to initially receive and dispatch protocol messages. Furthermore, a separate thread is created to handle timer events. Synchronisation points are set at

- access to the central state repository (synchronisation point per session),
- interfaces to traffic control (synchronisation point per interface),
- access to the central timer management (global synchronisation point), and
- access to certain system services (global synchronisation point, see above).

Of course, using multiple threads on a single-CPU workstation cannot be expected to significantly increase performance other than potentially providing improved interaction with any external I/O operation. This design could be further improved. For example, the global lock for the timer system could be replaced by more fine-grained locking for each slot of the timer container. On the other hand, with the fuzzy timer scheme, access to the timer container is not as time-consuming and critical as with a sorted container. To this end, the purpose of this extension is to demonstrate the simplicity and feasibility of parallelizing RSVP operations as a proof of concept. Indicative performance tests have been carried out and are described in Section 5.6.5. The design of multi-threaded message processing is sketched in Figure 11.

It becomes very obvious that the object-relationship design alleviates the task of parallelizing message processing a lot. The reasons are given by the natural encapsulation of data and functionality in an object-oriented design. This allows for easy identification of synchronisation points. Because all state objects are stored and accessed through the session object, no additional locking is necessary for them, besides acquiring a single lock for the session.

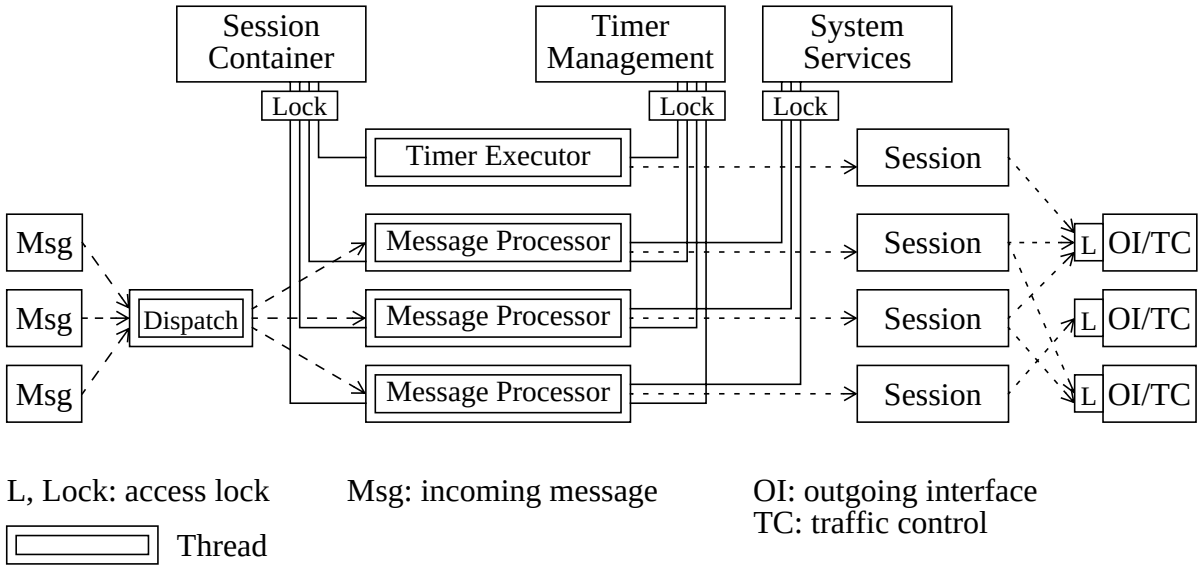


Figure 11: Multi-Threaded Message Processing

5.4 Protocol Extensions

In this section, the message processing rules for the RSVP extensions, which are presented in Section 4.4, are given. This is done separately from basic RSVP processing in order to clearly show the respective overhead compared to the standard RSVP specification. Only fundamental changes in processing are described. For example, allowing CIDR prefixes as address description requires to extend the message format of RSVP messages appropriately, which in turn leads to different parsing of messages. Such obvious details are omitted here. Because these extensions are mainly intended to augment RSVP's capabilities as an interface mechanism, these processing rules should not be considered as exclusive description. On the other hand, this consideration eventually applies to the basic processing rules, as well.

5.4.1 Compound Prefix Addresses

Compound prefix addresses change the addressing of RSVP messages in two ways. First, by allowing network addresses to be used and second, by allowing more than one address to be specified. This applies to session descriptions, i.e., the SESSION object and sender descriptions, i.e., SENDER_TEMPLATE and FILTER_SPEC objects. The changes in message processing are discussed separately for sender and session descriptions.

Sender Description

In case of a PATH, PTEAR or PERR message, the introduction of a network address in a sender description does not change the specification of operations at all. The case of multiple addresses within a single sender description can be handled by creating multiple PSB objects, but it seems efficient to extend the definition of a PSB appropriately to store the respective information in a single object.

As well, the introduction of compound addresses does not change the processing of RESV, RTEAR or RERR messages, except if the node acts as RSVP gateway and has previously combined multiple PATH messages into a single compound message. In that case, the setting of the scoping style determines, whether the respective service request is forwarded to multiple previous hops, or not. As mentioned in Section 4.4.1, it is currently not clear in which scenario which entity is responsible for determining the scoping style.

Session Description

In case a session description contains a network address, this does not change the processing, except at the border router towards the respective subnet, which, depending on the scoping style, has to decide whether to forward a PATH message into multiple subnets, or not. The information about the destination of a PATH message is looked up in the routing table, as usual and the message is forwarded using the session address as IP destination address.

If multiple addresses are given as session address, a similar situation exists. Either the routing lookup for PATH messages delivers a single interface, in which the message is simply forward using any of the addresses as IP destination address. Alternatively, if multiple routing entries direct the message to multiple interfaces, it depends on the scoping style, whether this message is forwarded any further.

In both cases, if a service enabler detects multiple outgoing interfaces for a PATH messages and decides to forward the message through all of them, the resulting session is treated similar to a multicast session at this node. The same applies for PTEAR messages and, in the opposite direction, to PERR messages.

The handling of RESV, RTEAR and RERR messages is completely analogous to the usual message processing, because they are transmitted hop by hop along the reverse data path according to previously stored routing information.

5.4.2 Hop Stacking

Hop stacking requires a change in the message format, as well, and furthermore, a change in the philosophy of message parsing. While a traditional RSVP implementation can, in principle, handle message objects in arbitrary order, the order of RSVP_HOP objects becomes important for the introduction of hop stacking. For handling of a PATH or PTEAR message at a node that employs hop stacking, it is sufficient to perform regular message processing, of course, without creating of or searching for any PSB objects. Upon reception of the corresponding RESV or RTEAR message, such a node can process it normally, but has to remove the first RSVP_HOP object and eventually forward the message to the node denoted by the next RSVP_HOP object.

RERR messages are transmitted hop by hop to those next hops which are stored in RSB objects, such that the processing is unaffected by hop stacking. PERR messages can be transmitted directly to the first hop of a stack. Intermediate hops that perform hop stacking have not stored path state and thus, do not need to be informed about such an error.

Each node that receives an RSVP message containing a stack of hops, has to copy this stack to related outgoing messages. Alternatively, for PATH messages, the stack might be stored in the local PSB object to insert it into corresponding RESV or RTEAR messages. In that case, the full stack can be removed from the message and only the local outgoing interface information is placed as RSVP_HOP object into the outgoing PATH message.

5.5 Implementation Status

In this section, the current implementation status is described in comparison to the RSVP specification. The software package is publicly available [Kar00b].

Basic Information

This implementation is a full implementation of RSVP operations, except for a few limitations given below. It has been tested to correctly inter-operate with the reference implementation. It is developed and tested to compile on Solaris 2.{6,7}, FreeBSD 3.X and Linux 2.X operating systems, using GNU C++ 2.95 and higher. The complete source package consists of approximately 19,000 lines of code. System-dependent code is cleanly separated and consists of about 2,000 lines of code with at most 250 lines dedicated to each system. The implementation effort is estimated to be about 18 person months. To this end, the implementation integrates schedul-

ing packages for CBQ scheduling [FJ95] on Solaris [WGC+95] and CBQ and HFSC scheduling [SZN97] on FreeBSD employing the ALTQ package [Cho98].

In experimental versions, an integration with the VCM package for ATM [SKS00a] on Solaris has been implemented, as well as extensions for embedding charging information into policy objects [Bet99] as described in [KSWS98a]. However, these components are currently not part of the public distribution release.

Limitations

Some of the properties of a fully compliant RSVP implementation with respect to [BZB+97] are currently missing. The main reason for them to be missing is their relative irrelevance with respect to the project goals, compared to the effort necessary to develop and test these features.

- IPv6 [DH95] is currently not supported. Due to the modular and portable design of the software, this should not create too much effort, but of course it would require the effort to be tested then.
- UDP encapsulation as described in [BZB+97] is not supported. It is not planned to support this in the future, because it does not belong to the core of the specification and it is already discussed in the IETF to drop this requirement [Bra98].

Other RSVP parts and extensions, which are currently in the standardization process or just being standardized are currently not supported, either, for example [BLT00] and [BGS+00].

Features

The implementation provides some additional features that are new to an RSVP implementation and rather rare for experimental protocol implementations in general.

The protocol engine can be compiled to execute in an emulation mode, in which multiple daemons execute on the same or differing machines and use a configurable virtual network between them, simulating shared link media and static multicast routing. Without such a feature, examinations of RSVP behaviour in non-trivial network topologies are only possible by using a simulator or by using real systems. In the second case, it is necessary to start multiple processes on multiple machines needing super-user privileges and a suitable infrastructure. The emulation mode allows to experiment without the need for additional software or hardware. A test-suite can be created by writing high-level configuration files, from which detailed configuration files are built. A preconfigured test-suite consisting of 16 virtual nodes and test scenarios is pro-

vided as part of the current distribution package. Furthermore, the emulation mode can be combined with real operation, for example, to test interoperability with other implementations.

An initial attempt to port this software to the OPNET modelling environment [OPN00] has been successfully realized [Met99]. While by far not all system interfaces are supported yet, this shows the general practicability of such a port by modifying only a limited number of modules. Some minor changes have to be done in the current OPNET modelling library (version 6.0.L) in order to enable RSVP support.

For the purpose of experimenting with various protocol aspects and enhancements or creating optimized versions for specific purposes, a large part of the full functionality is compiled optionally and fully configurable at compile time.

5.6 Performance Evaluation

In order to assess the performance of an RSVP implementation and to address the usual concerns against its processing overhead, a number of performance experiments have been carried out. After describing the general setup, these experiments are documented here in detail. It is important to mention at this point that the KOM RSVP engine has not been subject to careful and detailed tuning on the coding level. No specific optimizations have been carried out, other than the general design decisions and algorithmic improvements described earlier in this chapter.

The first series of tests compares the performance of the KOM rsvpd with the ISI rsvpd. The second series investigates the current performance limits of the new implementation and the following experiments analyse the effect of algorithmic improvements that have been implemented. Additionally, an experiment is reported, which investigates the influence of the average flow lifetime on the processing effort. Finally, some experiments have been carried out to obtain additional interesting performance figures, e.g., about the end-to-end setup latency.

5.6.1 Experiment Setup

The experiments were carried out on standard PC-based workstations, which serve as router platform running FreeBSD 3.4. These workstations are equipped as follows:

- single Pentium III processor, 450 MHz, 512KB second-level cache
- point-to-point 100 Mbit/sec Ethernet links, 3Com 3c905C-TX interface cards

- Gigabyte GA-6BXU mainboard, standard hard disk
- 128 MB RAM main memory

The total cost of this equipment as of December 1999 is approximately 600 Euros plus 50 Euros per network interface. For the tests, 6 nodes are connected as depicted in Figure 12. N_5 is used as destination host and N_1 as source host. Multiple unicast sessions are created by specifying multiple port numbers. Since handling of API sessions creates additional overhead at the respective end node, N_2 and N_6 are used as additional source and destination hosts, if a large number of sessions is created. The RSVP refresh interval is set to 30 seconds, as suggested in [BZB+97]. The RSVP daemons exchange basic RSVP messages only, without policy data and integrity objects.

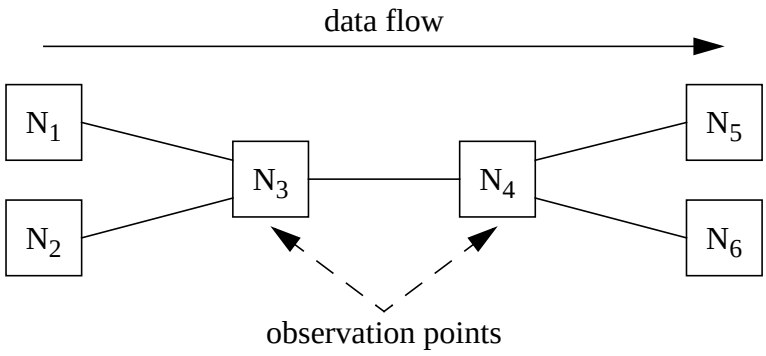


Figure 12: Experiment Setup for Performance Measurements

The load generator at N_1 (N_2) creates sessions and path advertisements with a randomized time interval in between, until a certain number of sessions *n* is reached. The upper bound of this time interval can be chosen for each experiment. When the target number of sessions is reached, the load generator creates and deletes sessions with the same randomized time interval respectively, in a way that the number of sessions is guaranteed to stay in the interval [*n*-10,*n*]. The receiver at N_5 (N_6) responds to each path advertisement by generating reservation requests, which establish the end-to-end flow reservation. All experiments encompass the generation and transmission of confirmation messages.

The observations have been taken at Node N_3 and N_4. Measurements have been done by periodically executing `top` and recording the highest numbers for current total memory consumption and percentage of raw CPU time that is reported for the RSVP daemon on either node.

Note that this kind of measurement introduces some inaccuracies and inherent randomness, which however does not seem to have affected the clarity of the results.

5.6.2 Comparison with Existing Work

For this first series of tests, no specific optimizations have been turned on in the KOM RSVP engine. The timer container has been configured to consist of 20,000 slots covering 50 milliseconds each. Both implementations have been compiled with the same optimization and debugging flags. The hash-based session container does not provide any performance gain for either alternative, because all flows are targeted to the same host.

Because of the restricted capability and reliability of the ISI rsvpd, its performance figures can be considered valid only to a limited extent. It has been chosen to avoid the memory leaks in the existing implementation to at least get some realistic performance figures by avoiding to place the additional burden of administering stale information to the software. As a result, the numbers for the ISI rsvpd can only be considered a lower bound for CPU consumption, because it usually crashes before a stable situation with creation and removal of sessions can be reached. The listed results consequently show the situation just before the crash. With the KOM rsvpd, each test has run for several minutes. The listed percentage of CPU time is the highest number that has been observed during that time. The memory consumption has always stabilized at the reported amount. The results are depicted in Table 3. The average lifetime of a single flow is calculated according to the creation/removal interval to be evenly distributed between zero milliseconds and the given number.

Note that the creation/removal interval is adapted for a small number of flows, such that the average lifetime of flows is not much smaller than the RSVP refresh interval. The influence of the average flow lifetime is further studied in Section 5.6.6. The numbers for the ISI rsvpd cannot be considered as stable as the numbers for the KOM rsvpd, because of the above mentioned reasons. However, it can be derived from these performance figures, that the KOM RSVP engine performs significantly more efficient than the ISI rsvpd. While it is unclear how much of this efficiency gain has to be attributed to a better coding style in general, it can obviously be concluded that the innovative object-relationship design at least does not prohibit performant implementation, however, at the expense of additional memory consumption. The KOM rsvpd consumes almost twice the amount of memory per flow when compared to the ISI rsvpd numbers.

Table 3: Experiment Results: ISI rsvpd vs. KOM rsvpd

experiment settings			ISI rsvpd		KOM rsvpd	
# flows	time interval	avg. lifetime	% CPU	Memory	% CPU	Memory
0	--	--	*0.00*	*1920K*	*0.00*	*2724K*
500	*100 msec*	*25.00 sec*	*2.05*	*2372K*	*1.13*	*3620K*
1000	*50 msec*	*25.00 sec*	*6.18*	*2856K*	*3.56*	*4544K*
1500	*38 msec*	*28.50 sec*	*10.01*	*3296K*	*5.32*	*5472K*
2000	*25 msec*	*25.00 sec*	*14.89*	*3768K*	*7.37*	*6388K*
2500	*25 msec*	*31.25 sec*	*20.51*	*4244K*	*9.91*	*7308K*
3000	*25 msec*	*37.50 sec*	*25.93*	*4728K*	*13.38*	*8236K*
3500	*25 msec*	*43.75 sec*	*33.74*	*5208K*	*16.60*	*9160K*
4000	*25 msec*	*50.00 sec*	*42.53*	*5692K*	*20.26*	*10084K*
4500	*25 msec*	*56.25 sec*	*51.37*	*6168K*	*23.73*	*11008K*
5000	*25 msec*	*62.50 sec*	*60.45*	*6656K*	*27.83*	*11928K*
5500	*25 msec*	*68.75 sec*	*79.69* *	*7140K*	*32.96*	*12848K*

* number of successful reservations: ~ 5400

5.6.3 Performance Limits

The goal of this set of tests is to find the upper limits of reservation requests for a tuned version of the RSVP implementation. The experiment setup and measurements have been done as described above. In the tuned version, the timer container consists of 100,000 slots covering 10 milliseconds each and API processing is disabled at intermediate nodes. Assertion checking and debug output is turned off. Since these tests are carried out in a limited infrastructure with at most two destinations hosts, port numbers are included into the hash calculation for the session container in the tuned version. Because doing so establishes a perfect hash distribution for the test scenario, the session hash index has been restricted to 4096 to simulate a realistic situation. Furthermore, the load generation is distributed between all four end nodes as depicted in Figure 12. The results are listed in Table 4.

The following conclusions can be drawn from this experiment. Tuning the protocol implementation reveals a significant potential for increasing the performance. A router platform based on standard PC hardware can handle the full signalling for 50,000 unicast flows. The higher amount of initially allocated memory for the tuned version can be attributed to the additional memory requirements for the finer-grained timer container. The memory requirements

Table 4: Experiment Results: Performance Limits of KOM rsvpd

experiment settings			basic KOM rsvpd		tuned KOM rsvpd	
# flows	time interval	avg. lifetime	% CPU	Memory	% CPU	Memory
0	--	--	*0.00*	*2724K*	*0.00*	*4724K*
2500	*25 msec*	*31.25 sec*	*9.91*	*7308K*	*4.39*	*9324K*
5000	*25 msec*	*62.50 sec*	*27.83*	*11928K*	*8.50*	*13940K*
7500	*25 msec*	*93.75 sec*	*58.11*	*16548K*	*11.38*	*18560K*
9800	*25 msec*	*122.50 sec*	*93.12*	*20788K*	--	--
10000	*25 msec*	*125.00 sec*	*65.09* *	*21156K*	*14.75*	*23168K*
15000	*25 msec*	*187.50 sec*	--	--	*20.95*	*32396K*
20000	*25 msec*	*250.00 sec*	--	--	*27.73*	*41632K*
30000	*25 msec*	*375.00 sec*	--	--	*40.67*	*60096K*
40000	*25 msec*	*500.00 sec*	--	--	*55.17*	*78556K*
50000	*25 msec*	*625.00 sec*	--	--	*67.99*	*97012K*
40000	*12 msec*	*240.00 sec*	--	--	*56.69*	*78556K*
50000	*10 msec*	*250.00 sec*	--	--	*70.56*	*97012K*

* load generated by 4 nodes, see main text

per flow remain unaffected. Two additional tests are listed, in which the creation/removal interval is set in a way that the average lifetime of a flow is approximately 4 minutes. The resulting load numbers demonstrate that the RSVP engine is indeed able to handle such a load of sessions, even when assuming a realistic average lifetime of calls. In fact, the impact of the lifetime of flows seems to be quite low. Further details are discussed in Section 5.6.6.

One particular detail can be observed when comparing the load numbers for the basic version in Table 4, depending on how many nodes participate in load generation. If four end nodes are used, messages arrive at intermediate nodes at three network interfaces, instead of two. As a consequence, the resulting load is substantially smaller and the performance limit is increased. The explanation of this behaviour is related to the implementation of the timer wheel in combination with the `select` system call, which is used to query for incoming packets. Each switch between timer management and message reception incurs a call to `select`, which must be considered as expensive. It takes at least 10 milliseconds on Linux, Solaris and FreeBSD to perform this system call when any timeout is given. After `select` returns, exactly one message is read from each eligible interface. Now, if messages arrive from more interfaces, more messages are potentially received, before the next invocation of timer management. This

leads to less switching between message reception and timer management and thus, reduces the total number of system calls, which in turn decreases the system load.

Figure 13 shows an overall picture of the experiment results from this and the previous section. The graph depicts the fraction of CPU load as a function of the number of sessions.

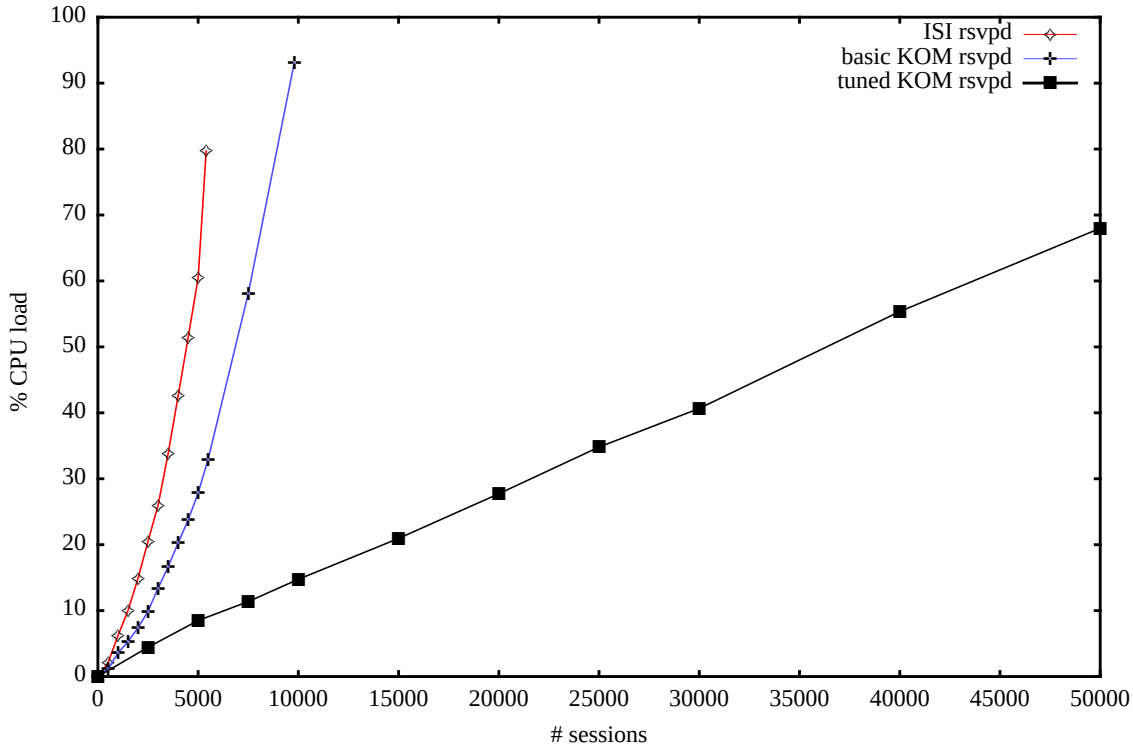


Figure 13: Performance Curve for ISI and KOM rsvpd

5.6.4 Fuzzy Timer Handling

While in theory only fuzzy timer handling can guarantee the property of overall linear complexity by simplifying access to the timer container, the previous experiment shows that, by enabling a fine-grained timer wheel, essentially this linearity is already achieved. In fact, a further modification of implementing fuzzy timers is needed to achieve any visible improvement at all. Because of the effects of switching between timer management and interface service, which is described in the previous section, all timers from the current slot are fired whenever the system enters the timer management. This further reduces the number of switches and calls to `select` and consequently, the overall processing load. A comparison with regular operation, which indicates the additional performance gain, mainly at a high load, is shown in Table 5. At a load of

about 58,000 flows, the system exceeds the maximum amount of main memory that is available and starts swapping to disk. This prohibits any further performant execution under this exceptional high load.

Table 5: Experiment Results: Performance of KOM rsvpd & Fuzzy Timer Optimization

experiment settings			tuned KOM rsvpd		fuzzy KOM rsvpd	
# flows	time interval	avg. lifetime	% CPU	Memory	% CPU	Memory
0	--	--	*0.00*	*4724K*	*0.00*	*4724K*
20000	*25 msec*	*250.00 sec*	*27.73*	*41632K*	*26.12*	*41632K*
40000	*12 msec*	*240.00 sec*	*56.69*	*78556K*	*53.37*	*78556K*
50000	*10 msec*	*250.00 sec*	*70.56*	*97012K*	*63.96*	*97012K*
58000	*8 msec*	*232.00 sec*	--	--	*~70.00*	*>108M*

5.6.5 Parallel Message Processing

This experiment is carried out to investigate the scalability of the multi-threaded message processing on a multi-processor platform. The experiment setup is very simple and shown in Figure 14. The end-systems E_1 and E_2 are the same PCs as in the other experiments and are connected to a router R. Both end-systems act as sender and receiver and create a large number of flows. A SparcServer 1000 with four 60Mhz CPUs running Solaris 2.6 serves as router. Because a separate thread is needed in the RSVP daemon to receive raw IP packets and dispatch them to the worker threads and another thread is used for timer handling, at least four CPUs are needed to carry out a reasonable experiment with two network interfaces.

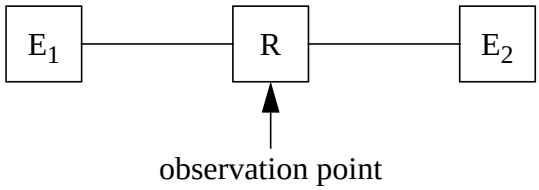


Figure 14: Experiment Setup for Parallel Processing

In order to test the capabilities of this system, tests have been run in single-threading mode and in multi-threaded mode with enabling an increasing numbers of CPUs. The goal of each test is to find the highest number of flows that can be handled reliably. Therefore, the RSVP

daemon has been slightly modified to regularly check the difference between the number of PSB and RSB objects. If this difference increases over a certain threshold, the daemon stops and reports the numbers of successfully established reservations. Because the total number of flows that can be sustained by this router is rather small, the RSVP refresh time is set to 3 seconds in order to increase the effect of established sessions compared to the creation of new ones. As well, to decrease the high influence of system code, which cannot be executed truly multi-threaded, the software is compiled without compiler optimization. The results are listed in Table 6. Each test is executed ten times and both the highest and lowest result are not taken into account for calculating the average result.

Table 6: Experiment Results: Performance of Parallel KOM rsvpd

# CPUs	# flows (individual tests)	avg. # flows
single-threaded	*451, 425, 464, 473, 466, 450, 450, 494, 520, 489*	*467*
1	*345, 389, 386, 380, 373, 373, 393, 350, 357, 366*	*371*
2	*552, 478, 571, 605, 571, 532, 563, 556, 518, 572*	*554*
3	*707, 723, 693, 756, 731, 718, 711, 702, 729, 727*	*719*
4	*592, 621, 711, 662, 652, 655, 648, 666, 655, 663*	*653*

It becomes clear from the resulting performance figures, that the potential for parallelization gains is indeed given, but certainly limited, at least on the tested platform. Furthermore, when comparing the results for single-threaded execution with those of multi-threaded execution on a single CPU, a significant overhead for synchronization mechanisms can be observed. These limitations must be partially accounted to the insufficient support of the operating system to support multi-threaded reception of raw IP packets and other low-level services, but also to inherent limitations of RSVP processing and the improvable implementation design of the parallel code in the KOM protocol engine. To this end, the result of implementing multi-threaded message processing is somewhat unsatisfactory. On the other hand, the design and implementation of multi-threaded message processing should be considered as a proof of concept, rather than the final design of a production-level implementation. Especially, with proper operating system support, the need for a separate dispatcher thread (which might very well form the bottleneck of the current system) and its synchronisation would be eliminated. As can be concluded from the results of Section 5.6.3 and Section 5.6.4, the overall performance of the RSVP

daemon is to a great extent determined by the system-level task of receiving packets from the network.

Another conclusion can be drawn from these tests, which backs up the above considerations. Testing the efficiency gain of a multi-threaded RSVP implementation on a simple and small multi-processor workstation as in these tests, is probably not sufficient to fully reveal parallel processing efficiency. For example, the performance drop when comparing execution on 4 CPUs can be explained as follows. On this platform, up to 3 CPUs can be bound to a process group exclusively. When using 4 CPUs, the RSVP daemon competes with other background processes and consequently, the overall scheduling effort increases for the operating system. This is reflected by the lower performances and indicates, that these tests cannot be regarded as real tests with 4 CPUs.

As discussed in Section 5.3.3, there is a broad field for further work on tuning the design and implementation of multi-threaded RSVP operations. Additionally, it would be very desirable to compare the results obtained during these tests with performance figures from different hardware and operating system platforms.

5.6.6 Lifetime of Flows

The experiments in Section 5.6.3 and Section 5.6.4 indicate that the average lifetime of flows has only limited influence on the computational overhead for a certain number of flows. In order to further investigate this issue, a dedicated set of tests has been done to examine this effect. The results are listed in Table 7.

Table 7: Experiment Results: Influence of Average Flow Lifetime

experiment settings			fuzzy KOM rsvpd	
# flows	time interval	avg. lifetime	% CPU	Memory
10000	*30*	*150.00 sec*	*13.96*	*23168K*
10000	*25*	*125.00 sec*	*14.75*	*23168K*
10000	*20*	*100.00 sec*	*14.99*	*23168K*
10000	*10*	*50.00 sec*	*15.77*	*23168K*
10000	*5*	*25.00 sec*	*16.65*	*23168K*
10000	*3*	*15.00 sec*	*21.48*	*23168K*
10000	*1*	*5.00 sec*	*77.10* *	*23168K*

* number of successful reservations: ~ 9700

These numbers back up the assumption that the average lifetime of flows has only limited influence on the overall computational overhead, as long as it is above the RSVP refresh interval, which has been set to 30 seconds for these tests. If the lifetime of flows becomes shorter, this generates an absolute increase in the number of RSVP messages and results in a much higher processing load. In fact, for these cases, it can be noticed in Table 7 that the increase of CPU load is approximately inverse proportional to the lifetime of flows. It can be concluded that usually a large fraction of CPU load is caused by refresh messages and on the other hand, that there is not much difference in the processing effort for setup messages as compared to refresh messages.

Indirectly, this result demonstrates the large potential for performance gains by extending RSVP with mechanisms to reduce the amount of state refresh messages, like those presented in Section 5.1.2. However, this particular behaviour could also be an artefact of this specific implementation, therefore, further work covering different implementations would be needed to investigate the details. Unfortunately, at this time, no such implementation is available. The ISI rsvpd cannot reliably handle the deletion of sessions, hence, this kind of experiment is currently not possible.

5.6.7 Other Experiments

Some other experiments have been carried out to assess this implementation under a variety of aspects. Because their results are highly bound to the specific scenario, they are somewhat illustrative, but they may not be regarded to be as relevant as the above experiments. Therefore, they are not documented here in the same level of detail.

RSVP & Packet Classification

In combination with the ALTQ package [Cho98], two adjacent routers running KOM rsvpd can both sustain the signalling for 10,000 flows and at the same time, classify and schedule 25,000 packets per second. Note that this result does not make any statement about the aspect whether all flows actually receive their QoS objective. Evaluating the ALTQ package is beyond the scope of this thesis.

End-to-End Setup Latency

Using the setup shown in Figure 15, tests have been carried out to measure the setup latency of RSVP requests. R_0 and R_3 are not handling any background RSVP session. R_1 and R_2 are load-

ed with up to 20,000 flows. The total end-to-end setup latency usually varied between 22 and 26 milliseconds, independent of the load of intermediate routers. Consequently, the setup latency can be estimated to be about 5-6 milliseconds per intermediate hop, which shows that even along a path with a large number of hops, the end-to-end setup latency will very probably be acceptable.

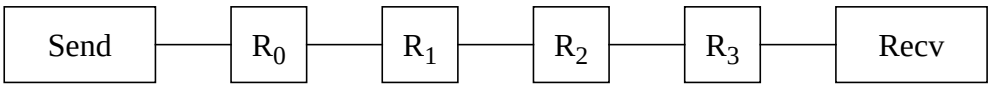


Figure 15: Experiment Setup for End-to-End Latency

5.7 Summary of Results

The assessment of RSVP's technical feasibility started with collecting and analysing the available material. Very soon it became obvious that the publicly available code as well as previous publications are not sufficient to study the aspects that were deemed interesting for this work. Therefore, a new implementation of RSVP has been developed from scratch. The overall design of the protocol engine has been published in [Kar00a]. It employs the notion of objects and relationships between them to efficiently store and access protocol state. It is innovative in its design and for example, allows easy inclusion of multi-threaded message processing. Furthermore, certain design and algorithmic extensions for the implementation of an RSVP engine have been proposed, some of which have been published in [KSS00]. An enormous potential for performance gains has been demonstrated by tuning the implementation appropriately.

In the performance experiments of Section 5.6, RSVP has been evaluated with respect to its basic mode of operation. The main goal of this work is to show the performance potential, even without further changes to the protocol. From the performance figures, it can be deduced that the suitability of RSVP as a general purpose signalling interface and protocol is much better than generally assumed. A standard PC router, at equipment cost of about 600 Euros plus network interfaces, can handle the signalling for more than 50,000 sessions in a realistic scenario.

If RSVP is applied for trunk signalling as presented in Section 4.5, there is already no reason to question RSVP's performance. However, experiments such as those carried out in this work are needed to assess its real-world applicability. Given the results of Section 5.6.6, it can be de-

rived that the proposed modifications of the protocol to reduce the number of refresh messages, will drastically improve its performance and thus, its applicability. The indicative numbers obtained for packet classification and scheduling, as well as end-to-end setup latency, further back up the claim about RSVP's applicability, even for employment in an IntServ-like QoS technology, which might be carried out in access networks.

As an illustrative example, consider a commercial high-speed router equipped with standard PC hardware as control processor. Furthermore, assume that the performance of an RSVP implementation can be tuned to sustain the signalling of 100,000 flows on such hardware (which does not seem unrealistic). If telephone calls have a bandwidth requirement of 64 Kbit/sec, the router could handle the signalling equivalent of 6.4 Gbit/sec. If such telephone calls make up for 32% percent of the overall capacity, such a router could handle the per-flow signalling for up to eight OC-48 links. This calculation does not even include the highly promising proposals presented in Section 5.1.2. Of course, telephone calls might be compressed and use less than 64 Kbit/sec. However, it is also very likely that they can easily be aggregated as presented in [SKWS99], which reduces the number of flows.

Essentially, the user-level RSVP implementation presented in this chapter is not the bottleneck for operation on a standard UNIX platform. Instead, the execution of system services largely determines the overall performance. This can be concluded from the experimental results, including those measuring the capabilities of multi-threaded message processing. Consequently, further work, especially on different hardware and operating system platforms, is needed to better understand the ultimate limits of an RSVP engine.

Besides its complexity of operation, RSVP is often objected to as being overly complex for implementation. The experience from this implementation shows that RSVP indeed exhibits a certain complexity. However, it was possible to realize an almost complete and even multi-threaded implementation of RSVP investing less than 18 person-months of development effort. Given the large applicability and the inherent complexity of the fundamental problem of providing performant end-to-end services, it can be argued that this experience contradicts those objections.

Chapter 6: Calculation and Charging

In this chapter, the investigation of QoS signalling is continued by considering the second important aspect of a future multi-service Internet, which is given by a commercialized environment. This work is carried out as follows. First, a set of requirements is described which are deemed important in a commercial environment. Second, a cost and price calculation framework is described and exemplified for a set of service classes. The ability to support cost-based pricing is a minimal requirement for any charging mechanism. Therefore, a distributed cost-based charging model is introduced, which serves as the basis for defining charging mechanisms. Then, a set of mechanisms is presented and discussed, which have been developed to relate RSVP service invocations to the corresponding charging information. Additionally, a cost-based calculation model for advance service requests has been developed, which is described and related to the RSVP mechanisms. Finally, an auction-based calculation scheme is selected and its potential for realization is briefly discussed, as well. The calculation models have been developed up to a point where economists can pick them up and continue to refine and optimize their characteristics.

6.1 Goals and Expectations for Charging of Communication Services

Some fundamental assumptions about the relationship between market participants have to be reviewed when the Internet is considered as a commercial communication network where users are charged according to their resource consumption. These assumptions are mainly driven by the individual market participant’s point of view.

- Each participant is independent and individually seeks to minimize its costs while maximizing its profit. This assumption fundamentally contradicts the pursuit for a globally optimal price function.
- Participants do not necessarily trust each other, not only with regard to authentication, but also in terms of correct information.
- Participants are used to a high level of legal security.
- Consumers of commodities are used to a high level of service and consumer protection.

- Communication prices are set independently by each network provider, *but* the price for a service likely depends on the costs for sub-services that are needed from other providers.

In the following subsections, these general assumptions are elaborated in more detail from different perspectives.

6.1.1 User Requirements

Predictability of Charges. Users want to be able to predict the costs of using a particular application, which include the expenditures for the communication services induced by this application. Therefore, an exact a-priori specification of communication charges would be desirable. However, if this requirement cannot be fulfilled, a set of weaker demands can be sufficient. First, a user should be able to roughly estimate his charges. Such an estimation does not need to be exact but should give at least a rough feeling to the user – similar to the knowledge that an international phone call of several minutes duration costs more than a Euro and not just a few cents. Second, a worst-case price should be known. Finally, it must be prohibited that a user is charged a higher price than previously announced, without giving his explicit approval.

Transparency and Accuracy of Charging. To find out how much is spent for which application and what are the reasons for this, users need the ability to determine the costs of a particular session, e.g., if an application uses several flows, the costs for each of these should be stated explicitly. Furthermore, for some users it might also be of interest to see where inside of the network the major charges are caused. This may give them information to switch to a different provider in future. Detailed per-session information about charges can also be used to decide whether a certain service and its quality offer a good value for the price. Since not all users are interested in such details, each user must be able to decide how much information should be given.

Convenience. Charging components should not make the usage of communication services much more difficult. The charging mechanisms themselves as well as the final bill based on the information gathered by the charging system must be convenient for its users. Hence, it must be possible for users to define ‘standard charging behaviour’ for their applications so that they are not bothered with details during the start up of an often used application. On the other hand, they should be able to change such a description easily to have control over their expenditures, e.g., changing spending caps. Furthermore, most users want to have as few separate bills as possible, i.e., have contracts and according business procedures with only one provider.

6.1.2 Provider Requirements

Technical Feasibility. The charging approach and its mechanisms must be feasible and operable with low effort. Otherwise, if it becomes too complex, the costs for the charging mechanisms might be higher than their gains. A set of real-life user trials needs to be performed to assure any of such characteristics. The added overhead for communication due to additional information transmitted between senders, network nodes and receivers, and also for processing and storage purposes especially in network nodes, e.g., to keep and manipulate charging information, must be as low as possible [FSVP98]. In addition, the introduction of scalable and low effort security mechanisms is essential for any type of counterfeit-proof charging records and billing data.

Variety of Business Models. The business of providing network service over packet-switched networks must be sustainable and profitable to attract the necessary investments into the infrastructure. It is unlikely to expect all service providers to adopt exactly the same business model and strategies. Therefore, charging mechanisms must be flexible enough to support a large variety of business models and inter-operate between multiple network domains employing different models. As well, a charging system must be flexible enough to handle different pricing strategies, for example during peak and off-peak times.

6.1.3 System Requirements

Flexibility. When information is transmitted from a sender to one or several receivers, the flow of value associated with this information can be (1) in the same direction as that of the data flow, (2) in the opposite direction, or (3) a mixture of both because both sides benefit from the information exchange. For example, in the first case, the sender transmits a product advertisement, in the second case, the receiver retrieves a movie for playback, and in the third case both sides hold a project meeting via a video-conference system. To support these different scenarios, a charging architecture must provide flexible mechanisms to allow the participants in a communication session to specify their willingness to pay for the charges in a variety of manners. Senders must be able to state that they accept to pay for some percentage of the overall communication costs or up to a specified total amount. Similarly, receivers may state what amount of costs they will cover. Additionally, charging mechanisms must allow for flexible distribution of communication charges among members of a multicast group. A number of cost allocation strategies can be found in [HSE97].

Fraud Protection and Legal Security. One of the most important issues demanded by participants is protection against fraud, i.e., that they do not have to pay for costs they have not incurred and that no one can misuse the system. The fear of users is that a provider may cheat or that other users may use their identity or derogate from them in any other way. Providers want to be sure that users indeed pay for the used service. A prerequisite against fraud is technical security, such that users cannot damage, misuse or intrude the provider's communication systems. Finally, legal security creates the demand that in case of a failure, there is enough information to determine responsibility for it.

6.1.4 Network Operation Requirements

Stability of Service. When a particular service with a certain quality has been agreed upon by the user and the provider, it must be ensured that the service is indeed delivered to the user. Hence, an exact definition of 'quality assurance is met' is needed. On the other hand, users must be able to estimate the impact of such quality goals on their applications, hence the definition must not be too complex. For example, if multiple users start a video conference application, they likely request a communication service with a specified bandwidth and delay. If the provider assures to deliver this service, the users expect no quality degradation and a very low probability of service disruption during the conference. In case of quality degradation or service disruption, an appropriate refund mechanism must be applied, which largely depends on the type of application, and hence, should be negotiated during the setup of a communication service.

Reliability of Service. In order to provide the basic infrastructure for an multi-service Internet, service availability must be very reliable. Current telephone networks are usually designed to keep the blocking probability in the order of 10^{-4}. Similar requirements are likely to apply to integrated services networks as well. To assure such a low blocking probability, even during peak hours, significant effort in the area of network and traffic engineering is necessary, which in turn must be accompanied by appropriate business calculation. A slightly different situation exists in case of per-packet QoS guarantees without explicit flow admission control. In that case, the notion of blocking probability might be replaced by reliability of service measured in terms of probability that the promised level of QoS is violated.

6.2 Cost-based Price Calculation

Under a profit-contribution model for communication services, the terms cost and price can be used somewhat interchangeably from the network provider's perspective. Marginal costs are negligibly low and in case of limited capacity, opportunity costs basically equal prices and are implicitly included when such a model is optimized. It can be assumed that actual market prices consist of a fixed transaction component, a resource allocation component and possibly other components. A fixed flow setup charge in combination with resource-based components leads to the usual characteristic that the function of price per resource unit is sub-additive for an increasing amount of resources. Furthermore, actual market prices can be influenced by marketing considerations and deviate from calculated prices. Nevertheless, it is important to internally use precise calculation as a reference model for the daily business process. In fact, the ability to perform cost-based price calculation is essential in order to carry out reliable internal calculation and utilize the results as input to capacity planning and marketing. The term *cost-based price calculation* is used to refer to internal cost and price calculation for the resource-based price component.

A multi-service Internet employs multiple service classes of packet-switched network communication. For each class, QoS is described by a vector of partially different parameters. All service classes compete for the same underlying network resources, such that an internal calculation model should be related to resource usage. However, service requests cannot directly be compared with respect to resource consumption, especially across service classes. Therefore, the existence of multiple service classes presents new challenges to a cost and price calculation model, which are only partially addressed by existing work. In this section, a framework for carrying out such cost-based price calculation is presented and specifically applied to the IntServ service classes. As well, several applications of such a calculation model are presented to illustrate its usefulness.

6.2.1 Single-Service Cost-based Price Calculation

Certain restrictions are usually applied to a calculation model to keep the economic time horizon limited and the complexity of the overall problem tractable. Specifically, the notion of a *business cycle* describes the time period in which a specific investment volume has to be recovered. One must further assume the existence of an *aggregated price-demand estimation* for each time during the business cycle. A simple profit-maximizing calculation model for a net-

work providing a single communication service class can then be expressed as maximizing the the following formula [MV91]:

$$\int_0^{Tb} R(\gamma(t))dt - K(C) \qquad t \in [0,Tb] \qquad \text{subject to } \gamma(t) \le C \tag{1}$$

Variables used:

$\gamma(t)$	aggregated demand at time t
$R(\gamma(t))$	aggregated revenue at time t
Tb	duration of business cycle
C	total available resource capacity
$K(C)$	amortization of capital investment over one business cycle

In such a model, the time parameter is a constant scaling factor, i.e., price and demand are applicable per fixed time unit, which can be chosen arbitrarily small. In the big view, a calculation model can be used in two areas of the cyclic calculation and planning process, shown in Figure 16. First, during capacity planning, network capacity can be increased as long as the increase is covered by expected revenue for this investment. However, changing the capacity of a communication network happens on a rather long time-scale, therefore, as a second application, this calculation model can be used to optimize revenue under a given limited capacity. In the second case, opportunity costs come into play when there is more demand than supply. If a service request has to be refused because another service request occupies resources, then the potential revenue of the refused request can be considered as opportunity costs of the accepted one. Although opportunity costs are not directly expressed in (1), they are implicitly included when optimizing it. In general, during capacity planning as well as network operation, a certain target value can be expected to limit the fraction of resources usually available for high quality service invocations. This is desirable, for example, to keep the blocking probability low or to prohibit starvation of best-effort traffic.

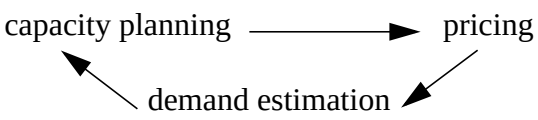


Figure 16: Cyclic Dependency among Calculation Tasks

6.2.2 Multi-Service Cost-based Price Calculation

There are several constraining factors for cost-based price calculation for a packet-switched multi-service network, resulting from multiple service classes using the same resources. In this section, two axiomatic requirements are established and then, a general calculation formula resulting from these requirements, is presented. Finally, some additional aspects are discussed, which strongly devise cost-based price calculation to be modelled according to the axiomatic constraints.

Axiomatic Constraints

Linearity. The price for resource usage must be linear.

$$a \cdot p(x) = p(a \cdot x) \quad \text{for resource usage } x \tag{2}$$

This requirement is due to the possibility of arbitrage (resale at low transaction costs) in packet-switched communication networks. Because the service of transmitting packets from one node to another can be used for many different applications, it might be hardly feasible for regulation authorities to prohibit arbitrage in a conventional (i.e. legislative) way. Furthermore, the objective of prohibiting resale might be dropped. Any non-linear pricing scheme, however, can be exploited by arbitrage [MV91]. Even if external arbitrage is not an issue, linear resource prices seem to be most appropriate for internal calculation, because they properly reflect resource consumption.

Uniformity. The price for each resource's usage must be uniform across service classes.

$$p_{S_1}(x_q) = p_{S_2}(x_q) \quad \text{for each resource } q\text{, usage } x \text{ for each pair of service classes } S_1,S_2 \tag{3}$$

There are two reasons for this axiom. First, requests for different service classes might be substituted by customers, exploiting the knowledge about a service class' definition. This can be done for immediate use or for resale, which in turn resembles a kind of arbitrage. Second, opportunity costs might apply across service classes, if a request for one service class has to be refused, because a request for another service class occupies the resources.

General Price Calculation Formula

A common requirement for pricing communication services is that prices should be known before a service is requested [FD98,FSVP98,KSWS98a], hence the price per resource unit must be stable during a specific time period. Assuming this and taking into account both axioms from above, a general price calculation formula for multi-service networks should be defined as follows:

$$p(x,t) = a_1(t)x_1 + a_2(t)x_2 + ... + a_n(t)\,x_n \quad \text{for resource vector } x = (x_1,x_2,...,x_n) \text{ at time } t \qquad (4)$$

This linear price formula must be used for all service classes. For capacity planning it can be used to determine the optimal network capacity. In case of limited capacity, the optimal traffic mix during peak-load periods can be estimated similarly. This calculation method is termed *linear cost-based price calculation*.

Further Considerations

Auctions. From an economic point of view, every sales transaction can be considered as an auction [MM98]. Winner determination takes place by ordering all bids and choosing the highest one(s). In case of a multi-service network, multiple resources have to be considered, which is called *combinatorial auction*. The underlying theoretical problem of winner determination in combinatorial auctions is proven to be NP-complete [RPH98], but approximate polynomial solutions exist [San98]. The problem becomes even harder, if bids from multiple service classes for multiple resources cannot be ordered at all. However, if cost and price calculation is uniform for all service classes, this additional complexity is resolved.

Demand Interdependence. For packet-switched multi-service networks, it is very likely that demand patterns are interdependent between different service classes, because users might combine or substitute traffic flows of multiple service classes within a single application. Additionally, interdependency between resource parameters can exist. As an example, for a delay guaranteed service, the amount of buffer needed is largely determined by bandwidth and delay characteristics. If the price for a critical resource, e.g. bandwidth, is increased, demand for the service class, and thus demand for other resources, decreases as well. If a uniform price function is used, which represents all resource parameters, per-flow demand estimation for each service class can be replaced by estimating aggregated demand interdependency per resource parameter. This seems to be easier to accomplish, based on past measurements and experiences.

Multicast. A thorough study of allocating costs among members of a multicast group is presented in [HSE97]. Cost allocation is described by splitting each link's costs among a defined subset of group members. The particular definition of the subset determines the overall allocation strategy. Of course, the sum of all cost fractions must equal the total costs for a link. Realizing such an approach becomes much simpler, if costs can be expressed as a linear function of resource parameters, especially if charges are shared among receivers with heterogeneous QoS requirements.

6.2.3 Linear Cost-based Price Calculation for Integrated Services

The IntServ architecture is the only existing set of end-to-end service classes, which is currently defined by the IETF. Therefore, the calculation framework is exemplified for this set of service classes.

The most evident obstacle for developing a calculation model for the IntServ architecture is that certain service classes are currently not precisely defined. Therefore, this section begins by refining the service definitions. The other details of each service definition can be found in [Wro97b], [SPG97] and [GGPR96], respectively. For reasons of brevity and simplicity, the effects of traffic distortion for this model are not explicitly considered, other than what is specified in the error terms during Guaranteed service negotiation. Initially, this model is focused on a single link or a specific path through a network cloud, connecting two IntServ-enabled routers, although this restriction is partially dropped in Section 6.2.5. These are the relevant flow specification and error term parameters [Wro97a] which are used for the rest of this discussion:

b	bucket depth
p	peak rate
r	token rate
M	flow MTU
R	service rate
S	slack term
C	rate-dependent error term
D	rate-independent error term

The approach described here presents a general idea for a calculation method, rather than specific calculation rules to be used for each implementation. However, the IntServ service definition refinements given below can be considered to closely resemble realistic services.

Guaranteed. For the duration of a service invocation, each router is guaranteed to always have service rate R available for a flow conforming to the requested token bucket. Furthermore, all incoming packets exceeding rate R but below rate p are forwarded within the (indirectly) specified delay bound and no conforming packets are dropped due to buffer overflow as long as such bursts are not larger then b.

Guaranteed Rate. A flow is guaranteed to be serviced with average rate r. No specific buffering is guaranteed.

Controlled Load. A flow conforming to a token bucket is forwarded almost without queuing delay or loss, as long as its data rate is not higher than *r*. Bursts are forwarded with as little queuing delay and loss as possible, depending on the actual load situation. This is achieved by allocating a specific excess service rate and buffer for each flow and enabling flows to borrow unused resources from each other. An implementation might, for example, use a *guaranteed rate scheduler* [GLV95] in conjunction with *hierarchical link sharing* [FJ95] to accomplish such forwarding. Assuming constant excess parameters *f* and *g* to be used for each flow (depending for example on the total number of flows and desired failing probability), the following simple formulas can be defined for the amount of service rate *R* and total buffer *B*:

$$R = r + (p - r) \cdot f, \qquad B = b \cdot g \quad \text{with } f, g \in [0,1] \tag{5}$$

Please note that the following considerations do not rely on exactly the above definitions and formulas, but only on having any precise specification of resource usage in the first place.

Virtual Rate Parameters

In reality, only one parameter (service rate, i.e., forwarding capacity) denotes the total available rate resource of an outgoing link. However, there are up to two rate parameters, *r* and *R*, in IntServ service specifications with even different semantics depending on the actual service class. In order to allocate costs to reservation requests, a resource model using three *virtual rate parameters* is defined:

- The *token rate* (q_T) describes the forwarding rate that is always available and expected to be constantly used by a flow.
- The *clearing rate* (q_C) denotes a guaranteed forwarding rate on top of the token rate that is reserved per delay-guaranteed flow, but expected to be used only for bursts of data.
- The *residual rate* (q_R) is a forwarding rate on top of the token rate, which is only statistically available to a flow. This resource can consume the unused capacity of q_C.

These parameters can be used to express the resource consumption of service requests by mapping the rate parameters *r* and *R* from an IntServ flow specification to the virtual rate parameters according to Table 8. This mapping follows directly from the above service refinements and the definition of virtual rate parameters.

IntServ Calculation Model

Considering buffer space as additional resource parameter, the linear function

$$p(x_T,x_C,x_R,x_B) = a_T x_T + a_C x_C + a_R x_R + a_B x_B \tag{6}$$

Table 8: Rate Allocation for IntServ Service Classes

service class	q_T	q_C	q_R
Guaranteed	*r*	*R - r*	-
Controlled Load	*r*	-	*(p-r) · f*
Guaranteed Rate	-	-	*r*

can be used to assign resource consumption respectively prices to a flow requesting token rate x_T, clearing rate x_C, residual rate x_R and buffer space x_B. Prices are applicable per fixed time unit, which can be chosen arbitrarily small. The formula for calculating the amount of buffer *B* for Guaranteed service is given in [SPG97]. Converted back to the original IntServ parameters, the price function for each services class can be expressed as follows:

$$p_G(r,R) = p(r,R\text{-}r,0,B) = a_T \cdot r + a_C \cdot (R\text{-}r) + a_B \cdot B \quad (7)$$

$$p_{CL}(r) = p(r,0,(p\text{-}r)f,bg) = a_T \cdot r + a_R \cdot (p\text{-}r) \cdot f + a_B \cdot b \cdot g \quad (8)$$

$$p_{GR}(r) = p(0,0,r,0) = a_R \cdot r \quad (9)$$

These price functions form the basis for an IntServ calculation model, which is linear and uniform across multiple services classes and therefore, fulfils the axiomatic requirements described above.

For certain scheduling approaches (see [Zha95]), *schedulability* is an additional internal resource parameter. However, the service classes currently under consideration for IntServ, as well as the PHBs, which are standardized for DiffServ, heavily rely on rate-based scheduling semantics. Particularly, for Guaranteed service, each scheduler has to approximate a rate-based scheduling behaviour. Therefore, it is not needed to explicitly consider schedulability as separate resource.

6.2.4 Application of Linear Cost-based Price Calculation to Optimal Pricing

In this section, the calculation model is applied to an optimality approach for profit-oriented price calculation. The authors of [WPS97] present a very general and complete model for optimal pricing of multiple guaranteed service classes under consideration of price-demand functions. It is correctly pointed out there that analytically solving the whole model is mathematically intractable, therefore an approximating procedure is described to carry out planning and calculation. While other research approaches often deal with optimal pricing in a sense of optimal welfare, this pricing scheme is targeted to maximize profit for the provider.

However, as noted in [WPS97], a similar model can be developed to maximize other objectives. By slightly modifying and applying linear calculation for IntServ service classes, the model can be simplified and, at the same time, enhanced in several ways:

- Instead of using very general assumptions about admission control and the properties of service classes, the cost-based calculation model below specifically considers the definition of actual service classes by using resource-based parameters and linear calculation. Thereby, the applicability to multiple real service classes is given.
- In [WPS97], communication services and demand patterns are modelled by the notion of calls, i.e., call probability, call duration, static QoS, etc. While being applicable to ATM service classes, this model does not fit well with the IntServ framework. Instead, the linear cost-based model only uses aggregated demand functions for each time period, i.e., no assumption about specific flows is needed, but only an overall estimation of aggregated demand per network resource, depending on the price-vector. By that, the new model implicitly encompasses the above details and also covers dynamic QoS.
- As mentioned in the article, [WPS97] does not cover interdependency among service classes and furthermore implicitly assumes a discrete set of service classes. In the cost-based model, the fact that each service class offers a vector-space of resource quantities is taken into account, although the estimation of demand interdependency between resources remains as an open issue.

In accordance with Section 6.2.1, the price function (6) is extended by a time parameter. The core formula which shows the total revenue that is to be optimized can then be specified and looks as follows (roughly using the notation of [WPS97]):

$$\int_0^{Tb} \left\{ \sum_{X = T, C, R, B} a_X(t) \cdot \gamma_X(t) \right\} dt - K(C) \tag{10}$$

under constraints

$$\gamma_T(t) \leq C_{TCR} \qquad t \in [0,Tb] \tag{11}$$

$$\gamma_C(t) \leq C_{TCR} - \gamma_T(t) \qquad t \in [0,Tb] \tag{12}$$

$$\gamma_R(t) \leq C_{TCR} - \gamma_T(t) \qquad t \in [0,Tb] \tag{13}$$

$$\gamma_B(t) \leq C_B \qquad t \in [0,Tb] \tag{14}$$

Variables used:

$a_X(t)$	price coefficient for each unit of q_X at time t, corresponding to (6)
$\gamma_X(t)$	aggregated demand for q_X at time t, for price vector (a_T,a_C,a_R,a_B)
Tb	duration of business cycle
C_{TCR}	total available service rate (reservable bandwidth)
C_B	total available buffer space
$K(C)$	amortization of capital investment over one business cycle

Constraints (11), (12) and (13) denote the fact that the amount of service rate reserved as token rate cannot be reused, whereas service rate used as clearing rate can be used simultaneously as residual rate. Constraint (14) states that buffer space cannot be reused. Please note that this is not in contradiction with multiple Controlled Load flows borrowing resources from each other.

Comparing (10) with the corresponding formula in [WPS97] shows that using virtual rate parameters and considering only aggregated demand significantly reduces the mathematical complexity but nevertheless enhances the level of detail by considering real resources instead of a general admission control expression. In general, this approach should be very useful to apply theoretic results in a real environment.

6.2.5 Economic Aspects of Guaranteed Service

Linear cost-based calculation is introduced as an approach to model resource and price calculation for single IntServ routers and attached links. In this section, it is shown how this modelling technique can be exploited to further analyse certain economic aspects of the Guaranteed service class. This also demonstrates how linear calculation can be extended to eventually cover a whole network domain consisting of multiple IntServ routers and respective transmission lines, and to analyse economic end-to-end aspects. A brief overview of the Guaranteed service class is given in Appendix D.

For this investigation, a general charging model is assumed where charges from each router are accumulated and shared between sender(s) and receiver(s). The values of price coefficients a_X used in this section are not assumed to be globally uniform, instead they are local to each router. The service rate R is assumed to always be larger than token rate r.

Token Rate vs. Clearing Rate

In [DVR98], it is pointed out that a receiver might choose to lower its service rate requirements R by increasing the average data rate r when requesting Guaranteed service. Since q_C, the difference of R and r, can be used for providing Guaranteed Rate service and Controlled Load service, a pricing scheme should give the right incentives for users to choose r according to their average data rate. Using linear calculation, this can be achieved by setting an appropriate higher price for q_T than q_C. The economically optimal price relation between q_T, q_C and q_R is part of the optimization problem from Section 6.2.4.

Error Terms

The C and D error terms, which are part of Guaranteed service negotiation, partially determine the service rate that must be requested by a receiver to guarantee a specific delay bound. In general, from an economic point of view, higher incoming C and D values lower the service quality, because a larger R is needed to achieve the same delay. The economic impact should be considered, for example, if an advanced QoS-oriented routing algorithm takes into account charges, it might be very important to have a quantitatively precise expression for this service degradation in order to value and compare different paths. The increase in service rate introduced by additional error terms (C_a,D_a) can be expressed as follows (see page 87 for definitions):

$$\text{let } X = \frac{b - M}{p - r} \tag{15}$$

$$R_a(C_a,D_a) = \frac{pX + M + C + C_a}{X + Q - (D + D_a)} - \frac{pX + M + C}{X + Q - D} = \frac{C_a(X + Q - D) + D_a(pX + M + C)}{(X + Q - D - D_a)(X + Q - D)} \tag{16}$$

Because $R \geq r$, R_a is part of the q_C resource, the cost increase at each router is given by:

$$p(C_a,D_a) = a_C \cdot R_a(C_a,D_a) \tag{17}$$

Not considering the economic impact of error terms could lead to a situation where a service provider exports high C and D values in order to cause a higher reservation for R. Internally, however, the real C and D values can be used and a smaller reservation for R is needed to guarantee the end-to-end delay.

Slack Term

As denoted in the relevant research and standardization documents about Guaranteed service, e.g. [WC97,SPG97,GGP+95], the slack term parameter S in a service request is intended to

flexibly relax the resource requirements at intermediate routers. Among other scenarios, this parameter can be used by first calculating the necessary service rate *R* depending on the desired end-to-end delay *Q*. Then, a higher service rate R^h is requested and the resulting difference of end-to-end delay is set as slack term parameter. The slack term can be 'consumed' by an intermediate router for reducing both delay and rate requirements, depending on the type of scheduler. In case rate requirements are reduced, a bottleneck router installs a smaller service rate R^l according to the formula given in [SPG97]* and adjusts the reservation message, such that upstream routers only install R^l as well. The effects of using the slack term in such a way can best be explained by rewriting the delay formula for Guaranteed service [SPG97] as:

$$Q = \frac{(b-M)\cdot(p-min(R_i))}{min(R_i)\cdot(p-r)} + \frac{M}{min(R_i)} + \sum_{i=1}^{n}\frac{C_i}{R_i} + D_{tot} \qquad (18)$$

for *n* routers, with R_i and C_i denoting the local service rate and error term at each router.

All downstream routers after a bottleneck router install the requested service rate R^h, which generates an economically unfortunate situation for the receiver. Considering (18), the smallest service rate installed along the path (in this case R^l) determines the pure end-to-end queuing delay. Having installed R^h at a number of routers only locally affects the additive delay component resulting from the local rate dependent error term *C*. Thereby, usability of the slack term in such a scenario largely depends on

- the relation of delay introduced by *C* to pure queuing delay, and
- the relation of total upstream to total downstream amount of *C*

as given at the bottleneck router.

Effectively, the receiver pays for a higher service rate at some routers, but does not perceive the total utility from it. The costs can be expressed as follows:

$$\text{cost}_{\text{slack}}(R^h,R) = a_C \cdot (R^h - R) \text{ at each router installing } R^h \qquad (19)$$

On the other hand, prices are lower at routers installing R^l:

$$\text{save}_{\text{slack}}(R^l,R) = a_C \cdot (R - R^l) \text{ at each router installing } R^l \qquad (20)$$

When different service rates are installed at routers due to the basic slack term mechanism, it is very likely that costs exceed savings, because the resulting increase in pure queuing delay can

* Note that the slack term formula on page 13 of [SPG97] should read: $S_{out} + \frac{(b-M)(p-R_{out})}{R_{out}(p-r)} + \frac{M+C_{toti}}{R_{out}} \le S_{in} + \frac{(b-M)(p-R_{in})}{R_{in}(p-r)} + \frac{M+C_{toti}}{R_{in}}$

only be recovered under pathological error term settings. Again, this economic impact should be considered when installing reservations using the slack term parameter.

Given (18), some suggestions can be made to extend the slack term mechanism. Often it could be advantageous to only locally install a lower rate and forward R^h instead of R^l to upstream routers. This would increase the maximum reduction of service rate and generate an even larger range of adaptability. In this case, the global minimum R (R_{min}) has to be transmitted in addition to the currently defined parameters and the slack term formula of [SPG97] has to be changed to:

$$S_{out} + \frac{(b-M)(p-R_{new})}{R_{new}(p-r)} + \frac{M + C_{toti} - C_i}{R_{out}} + \frac{M + C_i}{R_i} \leq S_{in} + \frac{(b-M)(p-R_{min})}{R_{min}(p-r)} + \frac{M + C_{toti}}{R_{in}}$$

with $R_{new} = min(R_{min}, R_i)$ and R_i, C_i denoting the local service rate and error term. (21)

The remaining variables have the same meaning as in [SPG97]. When the reservation is forwarded, R_{min} is set to the value of R_{new}.

This idea can even be extended as follows. A router using the slack term sends a confirmation message containing its local C and D terms as well as its local service rate back to the receiver. Given this information, a receiver can optimize R^h when refreshing its reservation. In case of global price information, further optimization would be possible by setting a certain R^h at ‘cheap’ links while setting R^l at ‘expensive’ ones, e.g. a transatlantic link. In theory, this creates a regular linear optimization problem, but even then, the necessary information exchange seems hardly be feasible with the current design of RSVP and also somewhat violates the Edge Pricing paradigm.

6.2.6 A Formal Model of Distributed Cost-based Charging

This section deals with the issue of extending cost-based price calculation to a distributed environment. In order to explain how charges and payments are calculated in a distributed manner, a formal definition of the necessary information exchange is given. Then, it is shown by solving an appropriate equation that all payments lead to exact revenue of the calculated local cost for each hop. For the purpose of explanation, the model is initially restricted to only one sender. An important issue is how to represent prices and payments. In this model, a price is a *price per resource unit*. In this context, the term *resource unit* is largely dependent on the representation of the communication service class, e.g., if the service class offers the parameters *bandwidth* and

delay, the price depends on these parameters. The parameters might be mapped to uniform internal parameters as shown in Section 6.2.3. Additionally, the total price for a communication session depends on the duration of this session. In reality, a payment can be represented, for example, as a direct exchange of virtual money or a credit or debit to an account. Below, a formal model of a network, prices and payments, as well as their allocation to a sender and multiple receivers, is presented.

Let $i = 0,... ,n,n+1,...n+m$ be a number of nodes in a multicast session, where
$i = 0$ denotes the sender.
$i = 1,...,n$ denote the intermediate nodes, and
$i = n+1,...,n+m$ denote the receivers. (22)

Let $m(j)$ be a *multicast function* that denotes the previous hop for a node j:
$m : \{1,...,n+m\} \rightarrow \{0,...,n\}$ with
$m(j) = 0$ for at least one $j \in \{1, ..., n\}$
$m(i) \neq i$ for all $i \in \{1, ..., n\}$
$m(i) = j \Rightarrow m(j) \neq i$ (23)

To make the charging procedure as transparent as possible for the sender and all receivers, the model is based on a *total charge*, which is eventually known to all end systems. Let C_i denote the total charge that has to be paid (by whoever) to connect hop i to the multicast tree. In order to recover this amount, a node splits it (according to a local policy) into multiple fractions $c_{i,j}$ for each outgoing interface where a service is established. A local price $L_{i,j}$ depending on the providers local pricing scheme is added.

Let $c_{i,j}$ denote a fraction of C_i for $m(j) = i$, with $\sum_{j,\, m(j) = i} c_{\mathrm{m(j),j}} = C_i$ (24)

Let $L_{i,j}$ denote the local price for a request on the outgoing interface to j, $m(j) = i$. The total charges for a hop is then calculated as follows:

$C_j = \quad c_{m(j),j} + L_{m(j),j}$ (25)

Between two adjacent hops, a charge is paid in the upstream direction. This payment is eventually recovered from the receiver end systems. The paid charge consists of a fraction of the charge until the current hop and the local price at the current hop. Let $RP_{i,j}$ denote such a receiver payment from a downstream hop i to an upstream hop j. Let r be the *fraction* the sender is willing to pay, so the receiver has to pay a fraction of 1-r.

$$RP_{i,j} = (c_{m(i),i} + L_{m(i),i}) \times (1 - r) = C_i \cdot (1 - r) \text{ for } m(i) = j \quad (26)$$

Additionally, there are downstream payments that are eventually recovered from the sender. The charge consists of the previously paid downstream payments and the sender fraction of the local price at the current hop. Let $SP_{j,i}$ denote a sender payment from an upstream hop *j* to a downstream hop *i*:

$$SP_{j,i} = \sum_{k,\, m(k) = i} SP_{i,k} + \sum_{k,\, m(k) = i} L_{m(k),k} \times r \quad (27)$$

Finally, let E_j denote the earnings at node *j*. These are defined as the difference between the incoming and outgoing payments and it is shown that they are equal to the sum of local prices:

$$E_j = \sum_{m(k) = j} RP_{k,j} + SP_{i,j} - \sum_{k,\, m(k) = j} SP_{j,k} - RP \text{ for } m(j) = i \quad (28)$$

It follows that:

$$\begin{aligned} E_j = & \sum_{k,\, m(k) = j} (c_{m(k),k} + L_{m(k),k}) \times (1 - r) + \sum_{k,\, m(k) = j} SP_{j,k} + \sum_{k,\, m(k) = j} L_{m(k),k} \times r \\ & - \sum_{k,\, m(k) = j} SP_{j,k} - (C_j \times (1 - r)) \\ = & \sum_{k,\, m(k) = j} c_{m(k),k} \times (1 - r) + \sum_{k,\, m(k) = j} L_{m(k),k} \times (1 - r) + \sum_{k,\, m(k) = j} L_{m(k),k} \times r \\ & - (C_j \times (1 - r)) \\ = & (C_j \times (1 - r)) + \sum_{k,\, m(k) = j} L_{m(k),k} - (C_j \times (1 - r)) \\ = & \sum_{k,\, m(k) = j} L_{m(k),k} \end{aligned} \quad (29)$$

It can be concluded that charging mechanisms, which adhere to this model, fulfil the minimal requirements of allowing for cost-based price calculation.

If receivers request shared reservations that apply to at least one common sender, they are merged on shared links. In that case, each sender's fraction must not directly be applied to the single charge on a shared link, otherwise the distribution of payments does not come out correctly. If for example two senders independently specify to cover half of the charge, the use of shared reservation style would cause them to effectively pay for the total cost of a shared link, whereas a receiver might get away for free. The following definition is used to handle this case.

Let $L_{s,i,j}$ denote the local price for a fixed filter reservation regarding sender *s*. The price $FFP_{s,i,j}$ for a fixed filter reservations can then be expressed as:

$$FFP_{s,i,j} = \quad (c_{i,\mathrm{j}} + L_{\mathrm{s,i,j}}) \times (1 - r) \tag{30}$$

Let $SFP_{i,j} = max_{s \in SF}(FFP_{s,i,j})$ denote the price for a single shared reservation style from hop i to hop j. Let r_s denote the charging fraction for sender s. When SF denotes the set of senders merged by a shared reservation style, the sender payment can be expressed as:

$$SP_{s,j,i} = \sum_{k,\, m(k) = i} SP_{i,\mathrm{k}} + \sum_{k,\, m(k) = i} SFP_{i,\,k} \times r_s \times \frac{FFP_{s,\,i,\,k}}{\sum_{s,\, s \in SF} FFP_{s,\,i,\,k}} \tag{31}$$

The definition of $RP_{i,j}$ has to be modified accordingly. Similar results can be found in [Her96].

6.3 Charging Mechanisms for RSVP

A fundamental aspect for the charging mechanisms presented in this section is the adherence to the Edge Pricing paradigm [SCEH96], which is briefly mentioned in Section 4.4.3. Corresponding to this paradigm, a user is charged only by the first network provider along the data path. This charge includes all expenses that subsequently might have to be paid by the provider when data is forwarded to another provider. While in principle a market participant may have business relations to multiple other participants, every single service instantiation is requested from and charged by exactly one peer participant. Edge Pricing is not necessarily needed to accomplish the requirements presented in Section 6.1, but it is an appealing paradigm that helps meeting demands like transparency, flexibility, convenience and legal security. Edge Pricing reduces the problem of multi-lateral contracts to a sequence of bilateral contracts and therefore hides much of the complexity which is introduced by the existence of multiple service providers and heterogeneous networks in the communication path.

While the charging mechanisms below are described in terms of applying them at each hop on the data path, it should be easy to see that this is not a necessary requirement. It is possible to partially deploy such mechanisms in the Internet, or exert them only at inter-domain boundaries. This is an immediate implication of adhering to the Edge Pricing paradigm. In terms of the QoS signalling architecture that is presented in Section 4.2, these mechanisms are carried out between service enablers.

6.3.1 Overview

As mentioned in Section 2.2.2, it is important to combine the design of networking technology with sound business models. This in turn requires appropriate charging interfaces. It is general-

ly important to consider two technical aspects of a charging architecture for a signalling interface. First, other than authentication, all objects of charging information should be considered optional in order to limit computational complexity for the default case. Second, it is important that charging-related handling of such service requests can be delegated to another entity denoted as *policy server*. Recapitulating the conceptual architecture presented in Section 4.2, this establishes a two-tier architecture, consisting of service enablers and policy servers. This is quite similar to the architecture developed in the IETF RAP working group [YPG00], which describes the distinction between a *Policy Decision Point* (PDP) and a *Policy Enforcement Point* (PEP). The PDP is analogous to the policy server and the PEP can be thought of as being the service enabler. One of the main advantages of such a decoupled architecture is that the filtering process of incoming service requests is transparent for the outside, because service requests can be redirected to (or intercepted by) the service enabler, or a separate policy server. Each of these components can delegate decisions or instruct another component with the results of a local decision. As long as the service interface is not affected by such internal decisions, any combination can be employed, which creates the highest flexibility, for example, to choose whether pricing and charging information is tied with service information or transmitted separately.

If prices for communication services are fixed per amount of resource consumption, it might not be necessary to include any charging information into signalling requests, other than authentication information of the originator. However, the option of charging information included in RSVP messages as presented in [KSWS98a] and [FSVP98] allows for the provision of additional semantics at a service interface. First, volatile pricing can be used to carry out auctions [RFS99] for resource access and second, a call-by-call service can easily be offered to end-users being connected to multiple service providers, especially, if electronic payments can be efficiently included into such signalling messages. If charging information and service requests are decoupled at the service interface, they have to be synchronized as belonging together later in the transmission path. Such synchronization creates additional overhead, which especially might impact the setup latency of per-flow requests. As discussed in Section 2.3.1, it is preferable to optimize the system for this most challenging case, therefore it seems reasonable to allow for the optional inclusion of charging information into service requests. Effectively, the policy-relevant parts of RSVP messages then carry the following information:

```
FLOWSPEC     :=  <service description>
POLICY_DATA  :=  <authentication> [<tariff>] [<payment>]
<tariff>     :=  <metering method> <price>
```

The field <service description> carries the selected service class out of a set as presented in Section 4.3. Authentication can be carried out as proposed in [YYP+00]. The field <tariff> optionally refers to an entry in a tariff structure that is otherwise announced or it directly includes the relevant information about metering and pricing. For metering, several alternatives are possible yet not exhaustive:

- metering by service request (as e.g. in [KSWS98a] and [FSVP98])
- metering by counting (aggregated) packets (e.g. [KS97], [Bri99], or [BBCW94])
- metering by counting special indications (such as ECN marks [RF99])

Similarly, <price> either refers to any externally available pricing table or directly includes price information. The field <payment> can be included in case of volatile per-request pricing [RFS99] or call-by-call services, if the originator of the request must indicate its willingness to pay a certain price.

The definition of charging information that is exchanged asynchronously to service invocations is beyond the scope of this work, therefore the following mechanisms focus on the potential of embedded information, which is transmitted as part of RSVP service requests. The mechanisms cover both unicast and multicast transmission, as well as sharing of transmission costs between senders and receivers. The RSVP specification already incorporates hooks for policy-related actions, namely the exchange of POLICY_DATA objects. Here, the proposed definition of certain protocol elements is described, along with their semantics and the mechanisms how to use them. These can be only rough definitions for a variety of reasons. The definition of a pricing function is a local matter of each service provider and probably depends on the service class that is actually chosen to transmit data. Furthermore, refinements to the protocol elements can be always introduced upon agreement between the operators of two adjacent service enablers.

6.3.2 Protocol Elements

The protocol information for charging is defined according to the general RSVP policy extensions proposed in [Her00a], but the charging mechanisms can as well be realized using a different general framework. According to [Her00a], a POLICY_DATA object contains one or multiple *policy elements*, which contents are not further defined in the referred proposal. Therefore, two policy elements for charging are described below. The goal is to define a small but ex-

pressive set of policy elements that can be used to exchange charging information for a variety of calculation models.

The basic elements of these charging mechanisms are given by a *Downstream Charging Policy Element (DCPE)* and an *Upstream Charging Policy Element (UCPE)*. The DCPE is sent downstream within the POLICY_DATA object of PATH messages. The fields in this structure are used to express a provider's price as well as a sender's payment information. Intermediate service enablers consider this information as input to their local price calculation and update or replace the policy elements by their own data. Upon arrival of a PATH message at the receiver's end system, the total price has been manifested and the receiver decides whether it is willing to pay this price to request a service. If yes, it issues a RESV message containing an appropriate UCPE containing its payment information. The same mechanism is applied at intermediate nodes, such that in general the arrival of a RESV message, which contains an appropriate UCPE, indicates the downstream hop's consent to be charged for a service request. Note however, that these mechanisms are not necessarily invoked synchronously at each hop along an end-to-end path for each service request. It is well feasible that relatively short-lived flow requests are mapped onto longer-lived trunk requests as described in Section 4.5.

With respect to the general form of policy-relevant information as presented in the previous section, the protocol elements refine the fields <price> and <payment>. The details of authentication are omitted here, because it is assumed that the standard security and identity mechanisms of RSVP, as described in [BLT00] and [YYP+00], can be used or appropriately extended. Metering is assumed to be done per service request.

Downstream Charging Policy Element

The *Downstream Charging Policy Element (DCPE)* is defined as follows:

```
DCPE ::=     <current price>,
             <max price>,
             <duration of price validity>,
             [ <sender payment> ]

<sender payment> ::= <sender fraction>,
                     [ <max flowspec> ],
                     [ <limit per receiver> ],
                     [ <limit per branch> ],
                     [ <max number of branches> ],
                     [ <max number of hops> ]
```

The field <current price> contains a representation of the currently valid price for requesting the service. Since this information might be volatile, it is bounded by <max price>. The prices in a communication network are expected to change over time, depending on the calculations of network providers, both in the short term (due to congestion situations) and in the long term. The <duration of price validity> field indicates how long the upstream hop assumes the current and maximum price information to remain stable. It is important to notice that a service enabler can hardly be held liable for this price information. Even if providers could be forced to charge the announced price, a service enabler might be implemented to simulate an admission control failure in such a case. The validity of price announcements will be largely determined by market forces and customers' sensitivity.

A sender can indicate its consent to cover a fraction of the total transmission charge. The <sender fraction> field allows the sender to specify the fraction of charge it accepts to pay. In order to protect the sender from arbitrarily high costs, it is necessary to restrict the maximum charging amount independently of any underlying restriction in distribution of data. A first approach would allow a sender to specify a maximum charging amount. However, there are obstacles to this procedure. Consider the case where a sender is interested in reaching a large user population with its data flow and sets a very high maximum amount. Each provider is independent in setting its prices, so if any provider had knowledge about a receiver that is connected directly to its network, it could set its price high enough to let the total sum be just below the maximum amount, but still be much higher than its normal price, hence, prohibit any other receiver to receive the data free of charge. The solution is to allow for the sender to give a fine-grained specification of its interests. Rather than specifying the maximum charging amount, the sender specifies a maximum per receiver, i.e., per complete path in the multicast tree in <limit per receiver>. Additionally, a sender can set an upper level of charges per single link with <limit per branch> and roughly restrict the geographic distribution of the sponsored flow by setting <max number of branches> and <max number of hops>. Together, the two 'branch'-fields can be used to restrict the sender's total charging amount per hop to the product of both values. Additionally, both fields together limit the overall number of sponsored nodes and end-systems. The total amount of charges can be calculated by multiplying the total number of end-systems with <limit per receiver>. The maximum QoS can be determined by the field <max flowspec>. If the <max number of hops> field is set in a DCPE, it must be decremented before it is forwarded within an outgoing DCPE. The other limitation fields must not be changed. It is

important to notice that service enablers must be discouraged from changing those fields, if it is not harmful for the forging party in the first place. Therefore, it is necessary that forgery can be detected through end-to-end control. Of course, there is a problem with processing these fields during the advertisement phase, while no actual service has been established. Nevertheless, a mechanism to limit the sender's expenditures is deemed useful to increase an end-system's control over the charging process it is subject to.

Upstream Charging Policy Element

The *Upstream Charging Policy Element (UCPE)* is defined as follows:

```
UCPE ::= [ <credit> ]
         [ <sender debit> ]
         [ <payment> ]
```

In the simplest case, the UCPE information can be omitted completely, because emitting a RESV message expresses a next hop's consent to be charged for the transmission service. However, in certain circumstances, it might be useful to transmit additional information. The <credit> field contains the actual charge, which the requesting node is actually willing to pay for this service. In case of fixed prices, this field mainly serves as control field for the requesting node to explicitly confirm the announced price. In case of service requests between intermediate nodes, the field <sender debit> contains the amount of money that the next hop expects to receive from the sender payment. The <payment> field contains additional information about the payment, e.g., the selection of an account, the identification of a prepaid billing card or an electronic payment. The maximum duration of validity for such a request is implicitly defined by the refresh timer value of the RESV message that carries the UCPE.

6.3.3 Application to Cost-based Price Calculation

Given the calculation model from Section 6.2.3, the price representation for cost-based price calculation can be formulated by a single homogeneous price function representing all service classes considered in the calculation model:

$$\begin{aligned} price := \quad & price\ for\ q_T, \\ & price\ for\ q_C, \\ & price\ for\ q_R, \\ & price\ for\ q_B \end{aligned}$$

In order to carry out distributed cost-based price calculation as defined in Section 6.2.6, prices can be accumulated at each hop and because the cost-based price function is linear, upstream

prices can easily be split at multicast branches. Below, it is explained how to carry out strict distributed cost-based price calculation employing these charging mechanisms. When describing the operation at a certain hop, the following indices are used:

- *i* denotes the previous hop
- *j* denotes the current hop
- *k* denotes the next hop

The local price $L_{j,k}$ for connecting the next hop is locally configured. The incoming DCPE carries the information C_j in <current price>. The fraction $c_{j,k}$ of C_j (24) is determined by the local price calculation module. The sum of the local price and the fraction of the incoming price (25) is placed into the <current price> field of the outgoing DCPE. When receiving a UCPE, the receiver payment $RP_{k,j}$ is calculated by multiplying the price with the amount of resources and duration of service invocation (set indirectly through the RSVP refresh period). The sender payment $SP_{j,k}$ can be found in the field <sender debit> in an incoming UCPE. The upstream sender payment $SP_{i,j}$ is calculated according to (27) or (31) respectively when transmitting the UCPE as part of a RESV message. Note that the <sender debit> field thereby automatically accumulates the sender payment when travelling upstream.

6.4 Cost-based Price Calculation for Advance Service Requests

It turns out that taking into account charging and pricing for service requests actually helps defining and implementing sensible services. In this section, an advance service specification is described, which employs pricing to generalize the technical service interface.

A number of approaches to specify service requests in advance have been published so far. Many of these concentrate on the issue of enabling advance service in the first place and signalling appropriate requests between network nodes. The fundamental problem of resource allocation in advance is depicted in Figure 17. Given a certain amount of future requests and no limitation on service duration, is it possible to schedule incoming service requests?

The work presented in [GSW99] indicates that occasionally preempting existing service invocations in favour of advance requests can increase overall resource utilization. In [SP97], an agent-based reservation system is presented, in which immediate and advance reservations are handled differently. Advance reservations always have to specify a finite duration and are never preemptable. Immediate reservations never specify a duration and are always preemptable. The

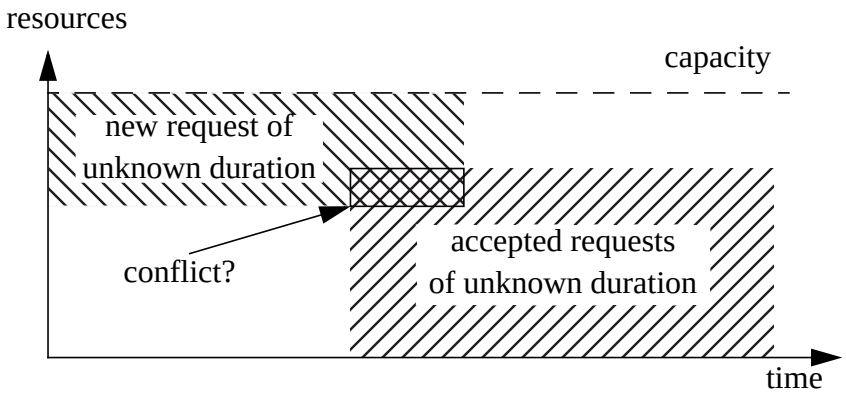


Figure 17: Scheduling of Advance Reservations

system considers certain time horizons, called *lookahead time* and *bookahead time*, to decide about acceptance of immediate and advance reservations. However, this service model introduces unnecessary limitations, which are of questionable virtue. For example, the authors note that selection of the time horizons is crucial for useful operation and certain requests are inherently precluded. On the other hand, users are expected to pay for service requests, so the question remains why certain requests (which are complicated for the system to handle) should be completely prohibited, instead of just setting appropriately high charges. Different handling of advance and immediate requests is introduced as an architectural benefit, however, it only adds complexity to the system. The system is further described, evaluated and implementation details for admission control are given in [SNNP99].

A different approach [FGV95] suggests that advance reservations also specify a service duration, but immediate reservations are not preemptable. Admission control for reservation requests in advance is done by only considering other advance reservations. Advance and immediate reservations are isolated by dynamically partitioning the network resources. Because the partition for advance reservations has to be large enough to admit all requested future reservations, this might lead to a situation, in which a significant amount of resources cannot be assigned to immediate requests, yet being unused.

In [BLB98], an architecture for realizing advance reservations in an IP/RSVP-based network is suggested and discussed. For the RSVP policy framework, the relevant proposed standardization document [Her00b] defines priority levels for service preemption. However, no background on advance reservation and the task of assigning priority levels is given. Further details

on the exact signalling procedures for enabling advance reservations on top of RSVP can be found in [SBK98]. Other approaches which are not discussed here for reasons of brevity include [Rei94,DKPS95]

Basically all previously suggested approaches conceive the fundamental admission control problem associated with service requests in advance. However, the attempts to deal with and completely solve this problem by technical means usually fall short, because of the limited scope of such approaches. One particular problem is given by the strict conceptual separation of immediate and advance reservations and the requirement to specify the duration of an advance reservation. As a consequence, this irrevocably limits the service time.

Realizing these problems, an integrated and generic service definition for immediate and advance invocations is presented below, in which a part of the admission control problem is delegated to a policy module and integrated with price calculation. This service definition does not fundamentally deviate from other suggestions, but the model is significantly less complex, yet more general, in that functionality at the resource layer is restricted to essential aspects, thus being very simple.

6.4.1 Network Service

The goal is to specify a uniform service description that covers both immediate and advance service requests. The service description should impose as few restrictions as possible on potential service requests. On the other hand, each service enabler must be able to determine whether a pending request can be accepted without violating guarantees given to other requests.

When accepting an advance service request, there is a *hold-back time*, the time frame between service request and service invocation. The fundamental problem when accepting advance requests can be formulated as follows: During the hold-back time, how can the allocated resources be used for other requests? If other requests arrive, which specify a service duration, it can be determined whether these are schedulable. A more difficult situation is given with requests which do not specify a fixed service duration. Three basic solutions exist for this problem. The first is that each service request, including immediate ones, also specifies a duration and is only accepted if resource availability can be guaranteed for the whole duration. The second possibility is to preempt service requests when their resources are needed for an advance request. As a third alternative, resources could be partitioned for immediate and advance service, such that no preemption is needed and only advance requests have to specify a duration. It

can be concluded from previous research efforts (see above) that at least one of these solutions has to be adopted by the network. However, there is no need to technically restrict the system to either one.

Partitioning of resources should be avoided if possible, because it prohibits resource sharing. Even in case of dynamic partitioning [FGV95], future advance requests block resources for other immediate requests. Declaring the duration of service invocations might not be possible and acceptable for all users and usage scenarios. On the other hand, the possibility of preemption might also not be acceptable under all circumstances. Therefore, a new network service is specified here, that does not rely on partitioning and integrates both preemption and duration declaration in a general way. Nevertheless, the potential for precisely predicting service guarantees is retained. This goal is achieved by distinguishing between duration of non-preemptable and actual service lifetime.

Service Definition

A service request R is described at request time by the 4-tuple (r,s,e,v) as follows:

- r: time of service request
- s: begin of service
- e: end of non-preemptable service
- v: amount of resource capacity

That is, at time r, a user requests an advance service invocation of capacity v, starting at time s, which is guaranteed not to be preempted until time e. This description does not include the actual service duration, which can be arbitrary. The difference of s and r expresses the hold-back time for an advance request. The key supplement to this service description is the following specification: At time t, each service request is in a state $p(t)$, called *preemption priority*, with

$$p(t) = \begin{cases} 1 & s \le t < e \\ 0 & \text{else} \end{cases} \tag{32}$$

If $p(t) = 1$, then the reservation request is guaranteed not to be preempted. A service invocation is assigned a preemption priority of 1 for the time that is specified in the service request. At the end of this duration the service is not automatically torn down, instead it is just considered preemptable for the sake of scheduling other non-preemptable requests. Employing this additional state description, the flexibility for requesting and managing advance service requests is extended, because even if a duration has to be specified for non-preemptable requests, this does

not necessarily result in a fixed a-priori service time. This service definition is graphically depicted in Figure 18. Some examples are given to demonstrate the flexibility of this service definition, each requesting an arbitrary amount of capacity v:

- immediate and preemptable request at time t_0: $R(t_0,t_0,t_0,v)$
- advance request at time t_0 for time t_1 requesting a minimum service time l: $R(t_0,t_1,t_1+l,v)$
- immediate request at time t_0 requesting a minimum service time k: $R(t_0,t_0,t_0+k,v)$

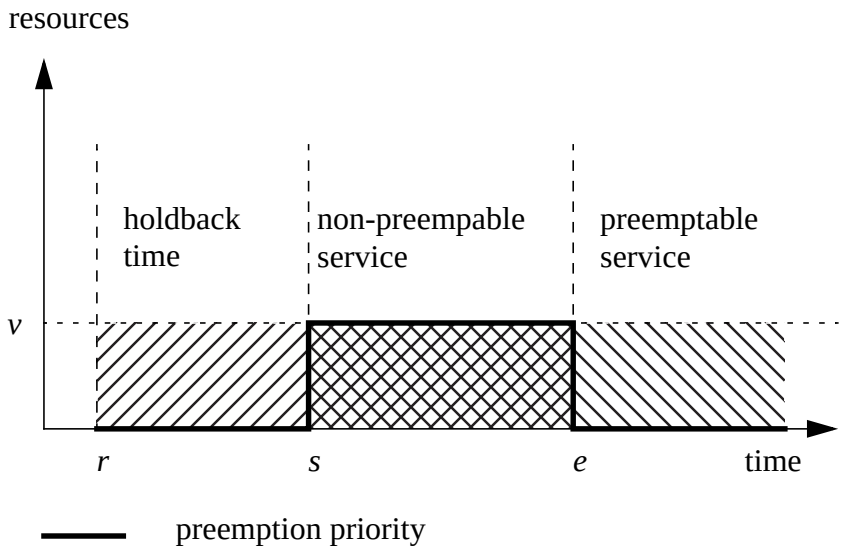


Figure 18: Advance Service Definition

Using this service definition, each possible instantiation of immediate and advance requests combined with the choice of preemption priority can be requested. The service definition is independent of the actual duration of service, it only determines the amount of time when a service invocation is not to be interrupted. It turns out that considering non-preemptable invocation time is sufficient to define a homogeneous service description for immediate and advance requests.

Admission Control

In order to provide service guarantees for blocking probability and preemption, an admission control algorithm is needed. For this algorithm, only non-preemptable service requests have to be considered at each time, because all other requests can be preempted. In this sense, the total load at time t is defined as follows:

$$\text{load}(t) = \sum_{i=1}^{n} v_i \cdot p_i(t) \quad \text{for } p_i, v_i \text{ from all service requests } R_i,\ i = 1,...,n \tag{33}$$

Defining C as total capacity and t_0 as current time, a set of requests R_i, (i = 1,...,n), is schedulable, iff

$$\text{load}(t) \le C \quad \text{for all } t,\ t \ge t_0 \tag{34}$$

Next, it is shown intuitively how to use this definition as an admission control condition and then its usage is formalized. Consider the situation shown in Figure 19. The dotted line denotes the total available capacity of resources and the long-dashed line depicts the current time t_0. The dashed line represents existing and requested non-preemptable requests. At time t_0, two new advance requests arrive, one of which is schedulable while the other one is not. For an immediate request, non-preemption can be guaranteed for a certain amount of time. Preemptable requests are not shown in this figure, because they do not influence the calculation of overall schedulability of non-preemptable requests.

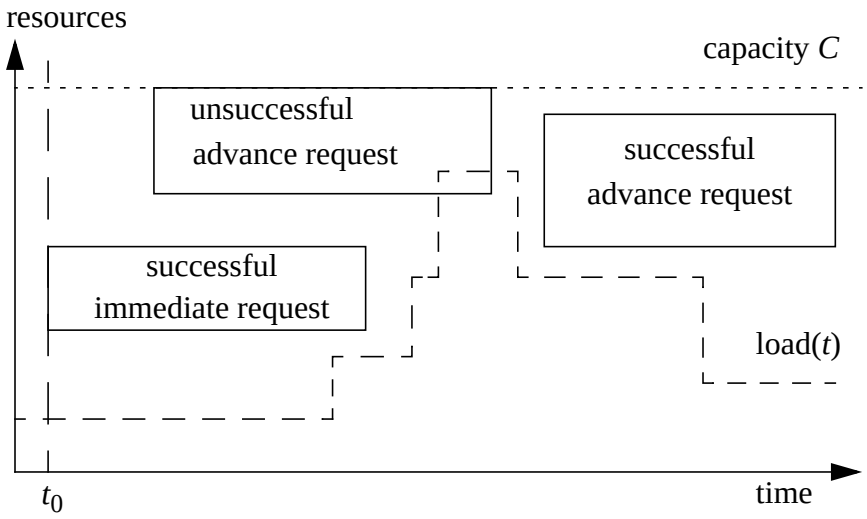


Figure 19: Existing and New Advance Requests

The admission control condition can be specified as follows:

At time t_0, a new service request $R_x = (t_0, s_x, e_x, v_x)$ can be accepted by the system, iff

$$\text{load}(t) + v_x \le C \quad \text{for all } t,\ s_x \le t < e_x \tag{35}$$

For admission control, it is sufficient to consider those times at which load(*t*) changes its value. If these times are denoted with τ_j, a simple algorithmic description can be given as follows:

```
decision = Accept
for each τj, sx ≤ τj < ex
    if (load(τj) + vx) > C
    then decision = notAccept
endfor
```

Note that this admission control condition does not principally differ from those of existing proposal, it just considers a subset of existing requests only. Therefore, proposals to efficiently implement such an admission control algorithm, as for example the work presented in [SNNP99], can be applied here, as well.

Service Invocation

There are several ways of invoking this service with a signalling protocol. One possibility would be to use a handshake mechanism:

user → system: REQUEST(*s*,*v*)

system → user: RESPONSE(e_{max})

user → system: CONFIRM(*e*) or REFRAIN

The user requests a certain amount of resources at time *s* and the system responds by specifying the maximum duration this reservation can be guaranteed to be non-preemptable. Then, the user either confirms requesting the service by choosing an end time or refrains from service invocation.

However, a handshake mechanism like this inhibits the problem that additional overhead is needed to keep the decision an atomic one. State information and timers would be needed to detect hanging invocations. Therefore, the following protocol elements can be used to invoke the reservation service:

user → system: REQUEST(*s*,*e*,*v*)

system → user: ACCEPT or REJECT(e_{max})

When using this service, a user specifies start time *s*, end time *e* and an amount of resources *v*. The system responds by either accepting the request or rejecting it, depending on its current state. In case the service is rejected, the system announces the currently possible maximum duration for non-preemptable service on an *informational* basis, i.e., without guarantees. This information can be used by the end-system to adapt its requirements and issue a new request.

Additional information might be added in case of service rejection, for example, an alternative start time. This service invocation model is idempotent and atomic and therefore, significantly reduces the complexity of protocol implementation. It also nicely integrates with RSVP's one-pass mechanism for reserving resources [SB95].

6.4.2 Policy Module

As discussed in the previous sections, there are fundamental conflicts associated with the admission control problem for advance requests. It seems clear that a general solution to this problem cannot be found, especially not purely in the resource layer. Therefore, this issue is approached by delegating the decision about acceptance of advance and preemption of existing service invocations to a policy module. In general, advance service requests introduce discrimination between usage requests and therefore, a policy module is needed to control, coordinate and compensate for resource consumption in the first place.

General Aspects

Several constraints can be identified when a policy scheme for advance resource allocation is being developed. The issue of protocol implementation is discussed at the end of Section 6.4.1. Using a policy module requires a network node to actually make two decisions atomically (admission control & policy control) which increases complexity, as discussed in Section 5.3.1. This overhead is bound, because the service specification employs a very simple invocation model.

It seems to be an open question whether advance requests should be subject to additional charges or receive a discount. Advance requests increase complexity in the network, however, a network provider can extract planning information for the future, which can be economically useful. To decide whether an advance service request should be given a rebate or charged an additional fee largely depends on the ability to adapt a network's capacity to demand, i.e., the planning horizon. If a request is received, which allocates resources after a certain point in time and if the sum of all requests are significant enough to adapt capacity, it is potentially suitable to grant a discount for this request. However, such a discount is currently beyond the scope of this model, because many other externalities would have to be considered as well, for example trust in the user, time of payment, general market developments, etc.

Advance requests and specification of preemptable and non-preemptable service time create additional means of discrimination between usage requests, therefore, compensation can be de-

manded from users requesting such features. Therefore, an immediate and preemptable reservation is considered as ‘normal’ and increased fees are due for further service characteristics. Although preemption is an integral part of the service model, in real operation it can be considered as an exceptional condition that does not occur regularly, because of careful capacity planning. Under this assumption, an alternative suggestion for pricing would be the airline model of overbooking aircraft seats. This could be applied by charging the same price for preemptable and non-preemptable requests and in case of preemption, a compensation would be paid by the network provider.

Next, requirements to a pricing model for the basic scheme are formulated, considering the service definition from Section 6.4.1. The effort of holding an advance request increases with the amount of time it is booked ahead, because other requests are potentially blocked. Consequently, the charge for a request should positively correlate to its hold-back time, i.e. (s-r). Similarly, a positive correlation should apply for the duration of non-preemptable service (e-s) and its price. Such a pricing model additionally serves as barrier against highly problematic requests, without completely prohibiting them in the resource layer. For example, a request for an infinite duration of non-preemptable service is not excluded in the service definition, but given a positive correlation, it results in an infinitely high price. As another example, a request in advance specifying no non-preemptable service duration provides no benefit to the user, but nevertheless requires management effort by the system. Hence, a higher price than for an immediate request should apply to discourage users from such requests.

The pricing model that is proposed in the following section is mainly intended to provide information for internal calculation of a network provider. In particular, prices only denote those parts of the total price which are resource-dependent (see Section 6.2). An additional fixed fee would cover the fixed costs per flow setup, for example for state maintenance in the system.

Pricing Model

In order to derive prices for service requests, an a-posteriori service description is needed, which includes an additional parameter d, expressing the actual duration of the service invocation. The service description for a request R is then given by the 5-tuple (r,s,e,v,d). The price consists of three components reflecting actual resource usage by reservation and scheduling effort for advance respectively non-preemptable reservations. Following the justification in Section 6.2.2, all price components are assumed to be linear in the amount of resources. Considering the requirements listed in the previous section, the price function looks as follows:

$$p(r, s, e, v, d) = a_1 \cdot v \cdot d + a_2 \cdot v \cdot (s - r) + a_3 \cdot v \cdot (e - s) \quad (36)$$

The first addition term expresses plain resource consumption during the actual service time and would in reality probably be refined according to (6) on page 88. The second addition term accounts for the hold-back time of an advance request, and the third addition term includes non-preemptable service time into price calculation. Note that all addition terms in this formula depend on the amount of resources v that is being reserved. This is due to the fact that for each price component the corresponding amount of effort is correlated to the amount of resources. The coefficients a_1, a_2 and a_3 are subject to economic calculation of a network provider, which is beyond the scope of this work. The following aspects explain how the above requirements are fulfilled by this price formula:

- The price positively correlates to the hold-back time and for an infinite request, the price becomes infinite through the second addition term.
- The price positively correlates to the non-preemptable service time and for an infinite request, the price becomes infinite through the third addition term.
- A ‘useless’ advance request without any non-preemptable service is still subject to additional charges through the second addition term.

The basic claim of this work is that the generic service model in combination with this pricing approach provides a higher flexibility than previous work, yet reasonable control of immediate and advance reservation requests. The approach essentially precludes infinite service requests by making them infinitely expensive. Thereby, the fundamental scheduling problem for advance service requests is partially delegated to self-regulation of the user. If desirable, it could be combined with a partitioning approach as in [FGV95] in a sense that a (dynamically sized) partition is priced differently.

6.4.3 Service Extension

In this section, the general applicability of this approach to advance service requests is demonstrated by extending the service definition and also covering this extended service model by a modified price calculation. The service extension is done by allowing modification of the non-preemptable duration of an existing reservation request. In that sense, a modification can be classified (see Figure 20) by the fact whether the new non-preemptable service duration is completely covered by the previous selection (case 1) or not (case 2). If yes, no special action has

to be performed at the resource layer, whereas otherwise admission control has to be executed on the modified request. However, both cases require activity in the policy module.

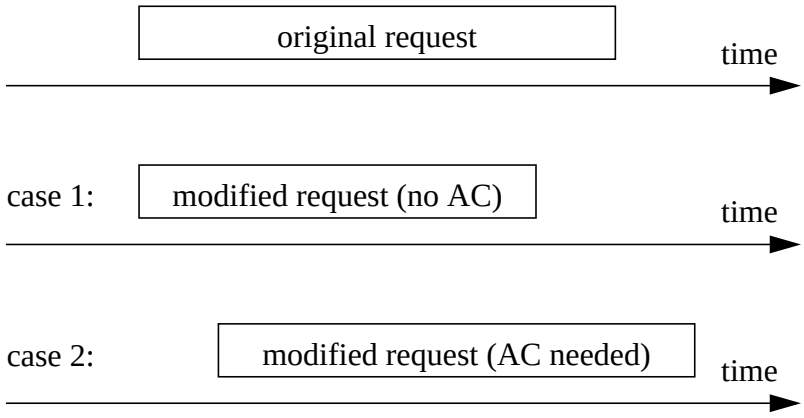


Figure 20: Modification of Advance Reservation Requests

Modified Service Request without Admission Control

As a specific example, consider the option that users are allowed to reduce the amount of non-preemptable service time by lowering the end time parameter *e*. This can formally be reflected by an additional parameter *e'*:

e': modified end of a non-preemptable service request, with $e' < e$ (37)

In order to cover this extended service by a policy module, the a-posteriori service description has to be extended, as well. Besides including *e'* into the service description, the time of this modification request is important, because the earlier the non-preemptable service time is reduced the more benefit (from better scheduling potential) the system has.

m: modification time of a service request (38)

Given the above considerations, the discount for such a modification should depend on both *e'* and *m*. It should not affect the price components for resource consumption and hold-back time from (36), but only the surcharge for non-preemption, i.e., the third price component from (36). A discount formula has to adhere to some other requirements as well:

- if $m = r$, the discount should cover the whole surcharge
- the discount should never exceed the surcharge
- if $m = e$, the discount should be zero
- the discount should never fall below zero

Using (36) as a basis, the discount can be expressed as follows:

$$\text{discount}(r, e, e', m) = a_3 \cdot v \cdot \left(b_1 \cdot \left(1 - \frac{m - r}{e - r} \right) + b_2 \cdot \left(\frac{e - e'}{e - r} \right) \right) \text{ with } b_1 + b_2 = 1 \qquad (39)$$

The last factor (*discount factor*) of this formula determines the discount in relation to the original surcharge and consists of two components. The expression multiplied by b_1 denotes the influence of *when* the request is modified, while the expression multiplied by b_2 describes by *how much* the non-preemptable time is reduced. The coefficients b_1 and b_2 allow weighting both aspects. The discount factor varies between 0 and 1. This discount formula satisfies all requirements listed above. A similar formula can be derived for deferring the start time without modifying the end time of non-preemptable service.

Modified Service Request with Admission Control

If admission control is needed for a modified service request, it has to be treated differently by the system. For admission control, the existing request has to be taken into account, such that it is not counted twice. Since admission control might fail, it seems most appropriate to consider this as a new service request. With respect to policy control, this is suitable as well, because the existing request can be deleted and charged, applying the discount calculation of the previous section. In case of acceptance, a new price for the modified request can then be calculated from scratch.

6.4.4 Application of Charging Mechanisms for Advance Service Requests

Both service and charging-related parameters of an advance service invocation have to be expressed in a signalling protocol. There are two options to include the technical parameters in RSVP, either by creating separate service classes for which the extended service model is applied or by allowing the extended model for all service classes. For the first alternative, dedicated TSPEC and FLOWSPEC objects should be employed to carry the parameters, whereas the second alternative can be realized by defining a new ADVANCE object for the preemption and timing values.

The price parameters for advance service requests can be exchanged by DCPE and UCPE objects as given in Section 6.3.2. The resource component a_1 from (36) can be expressed as presented in Section 6.3.3. The hold-back and non-preemption price components a_2 and a_3 are needed as additional price parameters. Assuming that the complete price formula (36) is implic-

itly included in the tariff description or explicitly referred to, the originator of a service request has all information available to build the appropriate UCPE for its RESV message.

6.5 Auction-based Price Calculation

In this section, the goal is to demonstrate the applicability of the basic RSVP charging mechanisms in the presence of a completely different price calculation scheme, which is given by carrying out auctions. In [RFS99], the authors describe a system employing *Delta Auctions* as a specialized type of Second-Price auctions [Vic61] to sell flow-based network service to clients in a multi-provider environment. Delta Auctions are characterized by the fact that winner determination does not take place at a fixed time, but rather continuously. Whenever a new bid arrives, this is compared to existing ones and if it is high enough, the request is admitted. If necessary, other requests have to be preempted. Since interrupting requests without providing the opportunity to increase the bid is not very acceptable to customers, a delay phase is suggested, in which the originator of a request that is about to be interrupted can increase its bid. The actually charged price (clearing price) for service invocations is given by the market clearing price, i.e., the highest rejected bid. As mentioned in Section 6.2.2, such an auction scheme is highly complicated, if varying amounts of multiple resources are requested atomically. Therefore, only the bandwidth resource is considered in [RFS99].

Employing RSVP as signalling protocol and the policy elements in Section 6.3.2 for transmitting price information, Delta Auctions can be carried out as follows. The field <current price> of the DCPE carries the current clearing price as an information for end-systems. In <max price> the highest current bid can be stored, if desired. The end-systems request services and store their bid into <payment>. Because of the periodic exchange of RSVP refresh messages, this process is carried out continuously. By comparing their own bid with the current clearing price, end systems can estimate the potential for being preempted and can react by appropriately increasing their bid. Therefore, a dedicated delay period is not necessary before preempting requests.

Such an auction can even be employed for multiple service classes including different QoS parameters at the same time by internally applying the transformation into uniform resource parameters as described in Section 6.2.3 to incoming requests and bids.

6.6 Summary of Results

In this chapter, the results from investigating the commercial aspects of a future multi-service Internet have been presented. A set of requirements has been identified, which serve as the basic guidelines to design appropriate technology. Cost-based price calculation is not likely to be the eventual method to determine actual sales prices, but an important prerequisite to carry out reasonable short and long time calculation and planning. Therefore, a framework for cost-based calculation has been developed and exemplified to create a calculation model for the IntServ service model. This work has been published in [KSWS99a] and [KSWS99b]. A formal model to perform distributed cost calculation is given. Then, a set of charging mechanisms has been designed and verified to enable distributed cost-based price calculation. An earlier version of this work has been published in [KSWS98a]. These mechanisms have been experimentally implemented as a proof of concept [Bet99]. As well, the charging mechanisms are verified to allow for charging of advance service requests, based on a respective calculation model. This model has been published in [KBWS99]. Finally, a completely different calculation approach, namely auction-based price calculation, is selected, and it is explained how the protocol elements for charging also allow the employment of this model with RSVP.

Chapter 7: Conclusion and Outlook

7.1 Summary

The focus of this thesis has been to investigate the issue of QoS signalling and charging for a future, commercialized multi-service Internet. Particular goals have been set to examine the suitability of RSVP to serve as a general QoS signalling protocol and to analyse the potential of cost-based price calculation in multi-service networks. Both the focus and these goals are formulated in Chapter 1 and motivated in Chapter 2.

In Chapter 3, existing work and fundamental knowledge about QoS and economics in packet-switched communication networks is introduced and reviewed. The most prevalent conclusion from existing work is a very high level of uncertainty about the optimal technology and pricing strategy to deliver end-to-end QoS. This uncertainty is very likely to produce a heterogeneous scenario, in which different providers employ different approaches. Nevertheless, inter-operation is a major requirement to deliver end-to-end services. Therefore, it is necessary to develop sound and flexible signalling interfaces, which do not prohibit heterogeneity, and at the same time provide for efficient realization of highly demanding services.

Such an architecture is presented in Chapter 4. It begins with a minimal general taxonomy of signalling architectures. Then, an architecture is proposed, which employs RSVP as its main signalling interface. Realizing certain shortcomings of the current specification of RSVP, appropriate extensions have been designed, particularly to negotiate trunk service invocations between subnets while reducing the amount of state information for service advertisements. The resulting design of RSVP enables to express requests for coarse QoS assurances for traffic aggregates and at the same time, to specify per-flow fine-grained QoS invocations. Such a homogeneous design is deemed superior to a situation with multiple protocols, because it eliminates functional redundancy. The applicability to a wide variety of scenarios is shown by a usage case evaluation.

The number of opinions about RSVP appears to be in no way related to the amount of actual knowledge and experimental results about RSVP’s performance. Only one publicly available

implementation exists so far and, unfortunately, it turns out not to be a well-suited candidate for further investigations. Therefore, a completely new design for an RSVP engine is described in Chapter 5. The innovative basic design is accompanied by a number of implemented improvements for the traffic and policy control interfaces and for timer handling. Furthermore, it allows for an easy introduction of multi-threaded message processing. An implementation has been created, which is publicly available and can hopefully serve as a basis for further examinations and developments by other research groups. Extensive performance tests have been carried out and are reported in this thesis. It turns out that the signalling for more than 50,000 flows can be handled on standard PC hardware without problems. The experimental results for both fuzzy timer handling and multi-threaded message processing have revealed limited but existing performance gains, even on a standard workstation.

In Chapter 6, the commercial aspects of a multi-service Internet are investigated. Cost-based price calculation is deemed an important prerequisite for capacity planning and sales price calculation, therefore a framework and model for cost-based price calculation has been developed. Afterwards, the design of charging mechanisms for RSVP is explained. These charging mechanisms are examined to allow for distributed cost-based price calculation. As another contribution, a model for integrating advance service requests into a network service by appropriately pricing them is developed. The charging mechanisms for RSVP have been verified to enable this as well as an auction-based model to sell QoS-enabled network transmission.

7.2 Conclusion

The major conclusion that can be drawn from the work reported in this thesis is that performance of RSVP is much better than appears to be generally assumed. Furthermore, although the experimental results in this thesis have only shown limited additional performance gains, the suggested improvements for an implementation design promise a significant potential to increase performance of employing RSVP in high-end routers. Through the extensions presented here, RSVP gains additional flexibility for a large variety of application scenarios. When combined with recent results to design efficient classification algorithms and implement them using cheap standard hardware, e.g. [DBCP97,WVTP97,KS98], even the realization of the IntServ QoS model does not seem to be completely out of scope for the middle-term future.

It is certainly feasible to combine RSVP service invocations with the respective charging information. Furthermore, it is possible to carry out cost-based price calculation for multiple serv-

ice classes and even distributed cost-based calculation. The advantages of charging and pricing for service requests can be further exploited by increasing the service interface for network services to cover advance requests, as well. A simple set of charging mechanisms can be used to enable plain cost-based price calculation, as well as supporting the generalized service model. Even a completely different pricing approach, namely auction-based calculation, can be carried out with these mechanisms.

7.3 Outlook

Many issues remain open for further work. There are ongoing efforts to develop QoS technologies, which incur less effort than flow isolation and flow-based scheduling. The economic aspects of providing multi-service network communication are not yet fully understood. Especially, it is not clear what the precise trade-off is between coarse-grained QoS technology in combination with overprovisioning on the one hand and the higher effort to realize fine-grained QoS technology on the other. Charging mechanisms and calculation models as those presented in this thesis have to be verified in large-scale realistic scenarios. Some more fundamental open issues are to design both mechanisms and algorithms for QoS path selection and routing.

Also, it is a very interesting task to define a reasonable subset of RSVP, which might be deployed sooner than the complete protocol specification. On the other hand, many further mechanisms have been specified for RSVP and it would be highly insightful to combine all of them in one protocol implementation to eventually study the behaviour in a large-scale. Such a testbed should include the full gamut of RSVP signalling, COPS policy operations and a combination of flow-based and DiffServ-based packet forwarding technology.

At Darmstadt University of Technology, further work in this area will be carried out by means of local, national and international research projects. The goal is to at least partially build a testbed as outlined above and experimentally study real operation of the Internet’s next generation protocol suite and gain insight to assess the respective competing proposals. This work will eventually lead to constructive proposals on how to consolidate the current alternatives for QoS provision. Additionally, it is intended to actively participate in further activities on the development of commercial service interfaces for Internet operation.

References

[ATM96a] The ATM Forum. Private Network-Node Interface (PNNI) Signalling Specification Version 1.0, March 1996.

[ATM96b] The ATM Forum. User-Network Interface (UNI) Signalling Specification Version 4.0, July 1996.

[AWK+99] George Apostolopoulos, Doug Williams, Sanjay Kamat, Roch Guerin, Ariel Orda, and Tony Przygienda. RFC 2676 - QoS Routing Mechanisms and OSPF Extensions. Experimental RFC, September 1999.

[BBC+98] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. RFC 2475 - An Architecture for Differentiated Services. Experimental RFC, December 1998.

[BBCW94] Roger Bohn, Hans-Werner Braun, Kimberly C. Claffy, and Stephen Wolff. Mitigating the Coming Internet Crunch: Multiple Service Levels via Precedence. *Journal of High Speed Networks*, 3(4):335–349, April 1994.

[BBD+99] Erol Basturk, A. Birman, Gary Delp, Roch Guerin, R. Haas, Sanjay Kamat, Dilip Kandlur, Ping Pan, Dimitrios Pendarakis, Vinod Peris, Raju Rajan, D. Saha, and Doug Williams. Design and Implementation of a QoS Capable Switch-Router. *Computer Networks*, 11(1-2):19–32, January 1999.

[BCS94] Robert Braden, David Clark, and Scott Shenker. RFC 1633 - Integrated Services in the Internet Architecture: an Overview. Informational RFC, June 1994.

[Bet99] Joachim Betz. Entwurf und Implementierung eines Abrechnungsmechanismus für RSVP-Datenströme. Diplomarbeit. Institut für Informatik, University of Mannheim, July 1999. in German.

[BFM+96] Anindo Banerjea, Domenico Ferrari, Bruce A. Mah, Mark Moran, Dinesh C. Verma, and Hui Zhang. The Tenet Real-time Protocol Suite: Design, Implementation, and Experiences. *IEEE/ACM Transactions on Networking*, 4(1):1–10, February 1996.

[BG97] Torsten Braun and Stefano Giorcelli. Quality of Service Support for IP Flows over ATM. In *Proceedings of Kommunikation in Verteilten Systemen 1997, Braunschweig, Germany*, pages 109–123. Springer, February 1997.

[BGS+00] Lou Berger, Der-Hwa Gan, George Swallow, Ping Pan, Franco Tommasi, and Simone Molendini. RSVP Refresh Overhead Reduction Extensions. Internet Draft draft-ietf-rsvp-refresh-reduct-04.txt, April 2000. Work in Progress.

[BJS99] Lee Breslau, Sugih Jamin, and Scott Shenker. Measurement-Based Admission Control: What is the Research Agenda? In *Proceedings of the 7th IEEE/IFIP International Workshop on Quality of Service (IWQoS'99), London, UK*, pages 3–5. IEEE/IFIP, June 1999.

[BLB98] Steven Berson, Robert Lindell, and Robert Braden. An Architecture for Advance Reservations in the Internet. Technical report, USC Information Sciences Institute, July 1998. available at http://www.isi.edu/%7eberson/advance.ps.

[BLT00] Fred Baker, Bob Lindell, and Mohit Talwar. RFC 2747 - RSVP Cryptographic Authentication. Standards Track RFC, January 2000.

[Bou98] Jean-Yves Le Boudec. Application of Network Calculus To Guaranteed Service Networks. *IEEE/ACM Transactions on Information Theory*, 44(3):1087–1096, May 1998.

[Bou00] Jean-Yves Le Boudec. A Proven Delay Bound for a Network with Aggregate Scheduling. Technical Report DSC2000/002, EPFL-DSC, Lausanne, Switzerland, January 2000.

[Boy97] Jim Boyle. RSVP Extensions for CIDR Aggregated Data Flows. Internet Draft draft-ietf-rsvp-cidr-ext-01.txt, December 1997. Work in Progress.

[Bra97] Scott Bradner. Internet Protocol Quality of Service Problem Statement. Internet Draft draft-bradner-qos-problem-00.txt, September 1997. Work in Progress.

[Bra98] Robert Braden. RSVP/IntServ MIB issues, June 23rd 1998. Contribution to rsvp mailing list. Available from ftp://ftp.isi.edu/rsvp/rsvp-1998.mail.

[Bri99] Bob Briscoe. The Direction of Value Flow in Connectionless Networks. In *Networked Group Communication, First International COST264 Workshop, NGC'99, Pisa, Italy*. Springer LNCS 1736, November 1999. Invited Paper.

[BYF+00] Yoram Bernet, Raj Yavatkar, Peter Ford, Fred Baker, Lixia Zhang, Michael Speer, Robert Braden, Bruce Davie, John Wroclawski, and Eyal Felstaine. A Framework

For Integrated Services Operation Over Diffserv Networks. Internet Draft draft-ietf-issll-diffserv-rsvp-04.txt, March 2000. Work in Progress.

[BZ97] Robert Braden and Lixia Zhang. RFC 2209 - Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules. Informational RFC, September 1997.

[BZB+97] Robert Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin. RFC 2205 - Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Standards Track RFC, September 1997.

[CBB+98] Eric S. Crawley, Lou Berger, Steven Berson, Fred Baker, Marty Borden, and John J. Krawczyk. RFC 2382 - A Framework for Integrated Services and RSVP over ATM. Informational RFC, August 1998.

[CCH94] Andrew T. Campbell, Geoff Coulson, and David Hutchison. A Quality of Service Architecture. *ACM Computer Communications Review*, 24(2):6–27, April 1994.

[CCR+95] Geoff Coulson, Andrew T. Campbell, Philippe Robin, Gordon Blair, Michael Papathomas, and David Hutchison. The Design of a QoS Controlled ATM Based Communications System in Chorus. *IEEE Journal on Selected Areas in Communications*, 13(4):686–699, May 1995.

[Cha00] Anna Charny. Delay Bounds in a Network with Aggregate Scheduling, February 2000. Available from ftp://ftpeng.cisco.com/ftp/acharny/aggregate-delay_v3.ps.

[Cho98] Kenjiro Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. In *Proceedings of USENIX 1998 Annual Technical Conference, New Orleans, LA, USA*, June 1998.

[CLSS98] Andrew T. Campbell, Aurel A. Lazar, Henning Schulzrinne, and Rolf Stadler. Building Open Programmable Multimedia Networks. *Computer Communications Journal*, 21(8):758–770, June 1998.

[CP99] Shaogang Chen and Kihong Park. An Architecture for Noncooperative QoS Provision in Many-Switch Systems. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, pages 864–872. IEEE Computer Society Press, March 1999.

[Cru91] Rene L. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE/ACM Transactions on Information Theory*, 37(1):114–131, January 1991.

[Cru95] Rene L. Cruz. Quality of Service Guarantees in Virtual Circuit Switched Networks. *IEEE Journal on Selected Areas in Communications*, 13(6), August 1995.

[DB95] Luca Delgrossi and Lou Berger. RFC 1819 - Internet Stream Protocol Version 2 (ST2) Protocol Specification - Version ST2+. Experimental RFC, August 1995.

[DBCP97] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small Forwarding Tables for Fast Routing Lookups. *ACM Computer Communication Review*, 27(4), October 1997. Proceedings of SIGCOMM'97 Conference.

[DH95] Steve Deering and Robert Hinden. RFC 1883 - Internet Protocol, Version 6 (IPv6) Specification. Internet RFC, December 1995.

[DHVW93] Luca Delgrossi, Ralf Guido Herrtwich, Carsten Vogt, and Lars C. Wolf. Reservation Protocols for Internetworks: A Comparison of ST-II and RSVP. In *4rd International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster (United Kingdom)*, October 1993.

[DKPS95] Mikael Degermark, Torsten Köhler, Stephen Pink, and Olov Schelén. Advance Reservations for Predictive Services. In *Network and Operating System Support for Digital Audio and Video, 5th International Workshop, NOSSDAV'95, Durham, New Hampshire, USA*. Springer LNCS 1018, April 1995.

[DVR98] Konstantinos Dovrolis, Maruthy Prasad Vedam, and Parameswaram Ramanathan. The Selection of the Token Bucket Parameters in the IETF Guaranteed Service Class. Technical report, University of Wisconsin-Madison, Madison, WI 53706-1691, USA, July 1998. Available from http://www.cae.wisc.edu/%7edovrolis/tokbuck.ps.

[FD98] Domenico Ferrari and Luca Delgrossi. Charging for QoS. In *Proceedings of 6th IEEE/IFIP International Workshop on Quality of Service, Napa, CA, USA*, pages vii–xiii. IEEE/IFIP, May 1998. Invited Paper.

[FGV95] Domenico Ferrari, Amit Gupta, and Giorgio Ventre. Distributed Advance Reservation of Real-Time Connections. In *Network and Operating System Support for Digital Audio and Video, 5h International Workshop, NOSSDAV'95, Durham, New Hampshire, USA*. Springer LNCS 1018, April 1995.

[FJ95] Sally Floyd and Van Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–368, August 1995.

[FNM+99] Gabor Feher, Krisztian Nemeth, Markosz Maliosz, Istvan Cselenyi, Joakim Bergkvist, David Ahlard, and Tomas Engborg. Boomerang - A Simple Protocol for Resource Reservation in IP Networks. In *Proceedings of IEEE Workshop on QoS Support for Real-Time Internet Applications, Vancouver, Canada*. IEEE Computer Society Press, June 1999.

[FSVP98] George Fankhauser, Burkhard Stiller, Christoph Vögtli, and Bernhard Plattner. Reservation-based Charging in an Integrated Services Network. In *Proceedings of 4th INFORMS Telecommunications Conference, Boca Raton, Florida, USA*, March 1998.

[GF98] Gene Gaines and Marco Festa. RSVP-QoS Implementation Survey, July 1998. Available at http://www.iit.nrc.ca/IETF/RSVP_survey/.

[GGPR96] Leonid Georgiadis, Roch Guerin, Vinod Peris, and Raju Rajan. Efficient Support of Delay and Rate Guarantees in an Internet. *ACM Computer Communication Review*, 26(4):106–116, October 1996. Proceedings of SIGCOMM'96 Conference.

[GGP+95] Leonid Georgiadis, Roch Guerin, Vinod Peris, Raju Rajan, Dilip Kandlur, and Scott Shenker. A Case for Extension of the Guaranteed Delay Service Specifications, November 22nd 1995. Contribution to int-serv mailing list. Available from ftp://ftp.isi.edu/int-serv/int-serv.mail.

[GK97] Richard J. Gibbens and Frank P. Kelly. Measurement-based connection admission control. In *Proceedings of the 15th International Teletraffic Congress - ITC 15, Washington, DC, USA*, pages 879–888. Elsevier Science B.V., 1997.

[GK99a] Richard J. Gibbens and Frank P. Kelly. Distributed Connection Acceptance Control for a Connectionless Network. In *Proceedings of 16th International Teletraffic Congress*, June 1999.

[GK99b] Richard J. Gibbens and Frank P. Kelly. Resource Pricing and the Evolution of Congestion Control. *Automatica*, 35, 1999.

[GLV95] Pawan Goyal, Simon S. Lam, and Harrick Vin. Determining End-to-End Delay Bounds in Heterogeneous Networks. In *Network and Operating System Support for Digital Audio and Video, 5th International Workshop, NOSSDAV'95, Durham, New Hampshire, USA*. Springer LNCS 1018, April 1995.

[GSW99] Albert G. Greenberg, R. Srikant, and Ward Whitt. Resource Sharing for Book-Ahead and Instantaneous-Request Calls. *IEEE/ACM Transactions on Networking*, 1(7):10–22, February 1999.

[HBC+00] Shai Herzog, Jim Boyle, Ron Cohen, David Durham, Raju Rajan, and Arun Sastry. RFC 2749 - COPS usage for RSVP. Standards Track RFC, January 2000.

[HBWW99] Juha Heinanen, Frad Baker, Walter Weiss, and John Wroclawski. RFC 2597 - Assured Forwarding PHB Group. Standards Track RFC, June 1999.

[Her92] Ralf-Guido Herrtwich. The HeiProjects: Support for Distributed Multimedia Applications. Technical Report 43.9206, IBM ENC Heidelberg, March 1992.

[Her96] Shai Herzog. *Accounting and Access Control for Multicast Distributions: Models and Mechanisms*. PhD thesis, University of Southern California, CA, USA, August 1996.

[Her00a] Shai Herzog. RFC 2750 - RSVP Extensions for Policy Control. Standards Track RFC, January 2000.

[Her00b] Shai Herzog. RFC 2751 - Signaled Preemption Priority Policy Element. Standards Track RFC, January 2000.

[HSE97] Shai Herzog, Scott Shenker, and Deborah Estrin. Sharing the "Cost" of Multicast Trees: An Axiomatic Analysis. *IEEE/ACM Transactions on Networking*, 5(6):847–860, December 1997.

[Hus00] Geoff Huston. Next Steps for the IP QoS Architecture. IAB Internet Draft draft-iab-qos-00.txt, March 2000. Work in Progress.

[ISI99] USC Information Sciences Institute. RSVP Software, 1999. http://www.isi.edu/div7/rsvp/release.html.

[ISO98] International Organization for Standardisation (ISO). Programming languages - C++. International Standard, 1998. ISO/IEC 14882:1998.

[ISS00] Integrated Services over Specific Link Layers (issll), 2000. IETF Working Group. http://www.ietf.org/html.charters/issll-charter.html.

[JNP99] Van Jacobson, Kathleen Nichols, and Kedarnath Poduri. RFC 2598 - An Expedited Forwarding PHB. Standards Track RFC, June 1999. RFC 2598.

[Kar00a] Martin Karsten. Design and Implementation of RSVP based on Object-Relationships. In *Proceedings of Networking 2000, Paris, France*, pages 325–336. Springer LNCS 1815, May 2000.

[Kar00b] Martin Karsten. KOM RSVP Engine, 2000. http://www.kom.e-technik.tu-darmstadt.de/rsvp/.

[KBWS99] Martin Karsten, Nicole Berier, Lars Wolf, and Ralf Steinmetz. A Policy-Based Service Specification for Resource Reservation in Advance. In *Proceedings of the International Conference on Computer Communications (ICCC'99), Tokyo, Japan*, pages 82–88, September 1999.

[Ker93] Aaron Kershenbaum. *Telecommunications Network Design Algorithms*. McGraw-Hill, 1993.

[Kle75] Leonard Kleinrock. *Queueing Systems, Volume I: Theory*. John Wiley & Sons, Inc., 1975.

[KMT98] Frank Kelly, Aman Maulloo, and David Tan. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[KS97] Martin Karsten and Ralf Steinmetz. An Approach to Pricing of Connectionless Network Services. In *Management of Multimedia Networks and Services - MMNS'97, Montreal, Canada*, pages 219–230. Chapman & Hall, July 1997.

[KS98] Srinivasan Keshav and Rosen Sharma. Issues and Trends in Router Design. *IEEE Communications Magazine*, 36(5):144–151, May 1998.

[KSBS00] Martin Karsten, Jens Schmitt, Nicole Berier, and Ralf Steinmetz. On the Feasibility of RSVP as General Signalling Interface. In *Proceedings of QofIS 2000, Berlin, Germany*. Springer LNCS, September 2000. Accepted for publication.

[KSS00] Martin Karsten, Jens Schmitt, and Ralf Steinmetz. Generalizing RSVP's Traffic and Policy Control Interface. In *Proceedings of the 7th International Conference on Parallel and Distributed Systems (Workshops), Iwate, Japan*. IEEE, July 2000. Accepted for publication.

[KSSW00] Martin Karsten, Jens Schmitt, Burkhard Stiller, and Lars Wolf. Charging for packet-switched network communication - motivation and overview -. *Computer Communications*, 23(3):290–302, February 2000.

[KSWS98a] Martin Karsten, Jens Schmitt, Lars Wolf, and Ralf Steinmetz. An Embedded Charging Approach for RSVP. In *Proceedings of the Sixth International Workshop on Quality of Service (IWQoS'98), Napa, CA, USA*, pages 91–100. IEEE/IFIP, May 1998.

[KSWS98b] Martin Karsten, Jens Schmitt, Lars Wolf, and Ralf Steinmetz. Abrechnungsverfahren für paketvermittelte diensteintegrierende Kommunikationsnetze. *Praxis in der Informationsverarbeitung und Kommunikation (PIK)*, 21(4):211–218, December 1998. in German.

[KSWS99a] Martin Karsten, Jens Schmitt, Lars Wolf, and Ralf Steinmetz. Cost and Price Calculation for Internet Integrated Services. In *Proceedings of Kommunikation in Verteilten Systemen 1999 (KiVS'99), Darmstadt, Germany*, pages 46–57. Springer, March 1999.

[KSWS99b] Martin Karsten, Jens Schmitt, Lars Wolf, and Ralf Steinmetz. Provider-Oriented Linear Price Calculation for Integrated Services. In *Proceedings of the Seventh IEEE/IFIP International Workshop on Quality of Service (IWQoS'99), London, UK*, pages 174–183. IEEE/IFIP, June 1999.

[KVA98] Yannis A. Korillis, Theodora A. Varvarigou, and Sudhir R. Ahuja. Incentive-Compatible Pricing Strategies in Noncooperative Networks. In *Proceedings of the 17th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'98)*. IEEE Computer Society Press, March 1998.

[LBZ99] Bob Lindell, Robert Braden, and Lixia Zhang. Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules. Internet Draft draft-ietf-rsvp-procrules-00.txt, February 1999. Work in Progress.

[Lei98] Brett A. Leida. Cost Model of Internet Service Providers: Implications for Internet Telephony and Yield Management. Master's thesis, Massachusettes Institute of Technology, Cambridge, MA, USA, February 1998.

[LR98] Tony Li and Yakov Rekhter. RFC 2430 - A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE). Informational RFC, October 1998.

[MEH00] Laurent Mathy, Christopher Edwards, and David Hutchison. The Internet: a Global Telecommunications Solution? *IEEE Network Magazine*, 2000. To appear.

[MESZ94] Danny J. Mitzel, Deborah Estrin, Scott Shenker, and Lixia Zhang. An Architectural Comparison of ST-II and RSVP. In *Proceedings of the 13th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'94)*. IEEE Computer Society Press, 1994.

[Met99] Holger Metschulat. Entwurf und Implementierung einer generischen Laufzeitumgebung für RSVP. Diplomarbeit. Fachbereich Elektrotechnik und

Informationstechnik, Darmstadt University of Technology, April 1999. in German.

[MHSS99] Laurent Mathy, David Hutchison, Stefan Schmid, and Steven Simpson. REDO RSVP: Efficient Signalling for Multimedia in the Internet. In *Interactive Distributed Multimedia Systems and Telecommunication Services*. Springer LNCS 1718, October 1999. Proceedings of IDMS'99.

[MM97] Jeffrey K. MacKie-Mason. A Smart Market for Resource Reservation in a Multiple Quality of Service Information Network. Technical report, University of Michigan, September 1997. Available from http://www-personal.umich.edu/%7ejmm/papers/reserve3.pdf.

[MM98] Jeffrey K. MacKie-Mason. Multi-Agent Systems, Mechanism Design, and Institutions. In *First International Conference on Information and Computation Economies (ICE-98)*, 1998. Invited Talk.

[MMV95] Jeffrey K. MacKie-Mason and Hal R. Varian. Pricing the Internet. In Brian Kahin and James Keller, editors, *Public Access to the Internet*, pages 269–314. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1995.

[MMV97] Jeffrey K. MacKie-Mason and Hal R. Varian. Economic FAQs About the Internet. In Joseph Bailey and Lee McKnight, editors, *Internet Economics*. MIT Press, Cambridge, 1997.

[MV91] Bridger M. Mitchell and Ingo Vogelsang. *Telecommunications Pricing, Theory and Practice*. Cambridge University Press, 1991.

[NC00] Kathleen Nichols and Brian Carpenter. Definition of Differentiated Services Behavior Aggregates and Rules for their Specification. Internet Draft draft-ietf-diffserv-ba-def-01.txt, February 2000. Work in Progress.

[NCS99] Anindya Neogi, Tzi-cker Chiueh, and Paul Stirpe. Performance Analysis of an RSVP-Capable Router. *IEEE Network Magazine*, 13(5):56–63, September 1999.

[NJZ99] Kathleen Nichols, Van Jacobson, and Lixia Zhang. RFC 2638 - A Two-bit Differentiated Services Architecture for the Internet. Informational RFC, July 1999.

[Odl99] Andrew Odlyzko. Paris Metro Pricing: The minimalist differentiated services solution. In *Proceedings of the 7th IEEE/IFIP International Workshop on Quality of Service (IWQoS'99), London, UK*, pages 159–161. IEEE/IFIP, June 1999.

[OPN00] OPNET Modeler, OPNET Technologies, Inc., 2000. http://www.opnet.com/products/modeler/home.html.

[PG93] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[PG94] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. *IEEE/ACM Transactions on Networking*, 2(2):137–150, April 1994.

[PHS99] Ping Pan, Ellen Hahne, and Henning Schulzrinne. BGRP: A Tree-Based Aggregation Protocol for Inter-Domain Reservations. Technical Report CUCS-029-99, Columbia University, New York, NY, USA, December 1999.

[Pos81] Jon Postel. RFC 791 - Internet Protocol. Standards Track RFC, September 1981.

[PS97] Ping Pan and Henning Schulzrinne. Staged Refresh Timers for RSVP. In *Proceedings of Global Internet'97, Phoenix, Arizona, USA*, November 1997. also IBM Research Technical Report TC20966.

[PS99] Ping Pan and Henning Schulzrinne. YESSIR: A Simple Reservation Mechanism for the Internet. *ACM Computer Communication Review*, 29(2):89–101, April 1999.

[PSC98] Kihong Park, Meera Sitharam, and Shaogang Chen. Quality of Service Provision in Noncooperative Networks: Heterogeneous Preferences, Multi-Dimensional QoS Vectors, and Burstiness. In *Proceedings of First International Conference on Information and Computation Economies (ICE-98)*, pages 111–127, 1998.

[Rei94] Wilko Reinhardt. Advance Reservation of Network Resources for Multimedia Applications. In *Proceedings of 2nd International Workshop on Advanced Teleservices and High Speed Communications Architectures (IWACA'94), Heidelberg, Germany*. Springer LNCS 868, October 1994.

[RF99] Kadangode K. Ramakrishnan and Sally Floyd. RFC 2481 - A Proposal to add Explicit Congestion Notification (ECN) to IP. Experimental RFC, January 1999.

[RFS99] Peter Reichl, George Fankhauser, and Burkhard Stiller. Auction Models for Multi-Provider Internet Connections. In *Proceedings of 10. GI/ITG Fachtagung MMB'99, Trier, Germany*. VDE-Verlag, September 1999.

[RHV99] Kadangode K. Ramakrishnan, Gisli Hjalmtysson, and Jacobus E. Van der Merwe. The Role of Signaling in Quality of Service Enabled Networks. *IEEE Communications Magazine*, 37(6):124–133, June 1999.

[RL95] Yakov Rekhter and Tony Li. RFC 1771 - A Border Gateway Protocol 4 (BGP-4). Standards Track RFC, March 1995.

[RPH98] Michael H. Rothkopf, Aleksandar Pekec, and Ronald M. Harstad. Computationally Manageable Combinational Auctions. *Management Science*, 44(8), August 1998.

[RVC99] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol Label Switching Architecture. Internet Draft draft-ietf-mpls-arch-06.txt, August 1999. Work in Progress.

[SA98a] Jens Schmitt and Javier Antich. Extended Traffic Control Interface for RSVP. Technical Report TR-KOM-1998-04, Darmstadt University of Technology, July 1998.

[SA98b] Jens Schmitt and Javier Antich. Issues in Overlaying RSVP and IP Multicast on ATM Networks. Technical Report TR-KOM-1998-03, Darmstadt University of Technology, July 1998.

[San98] Tuomas Sandholm. Winner Determination in Combinatorial Auctions. In *First International Conference on Information and Computation Economies (ICE-98)*, 1998. Invited Talk.

[SB95] Scott Shenker and Lee Breslau. Two Issues in Reservation Establishment. ACM Computer Communication Review, 25(4), October 1995. Proceedings of SIGCOMM'95 Conference.

[SBK98] Alexander Schill, Frank Breitner, and Sabine Kühn. Design and Evaluation of an Advance Reservation Protocol on top of RSVP. In *IFIP 4th International Conference on Broadband Communications, Stuttgart, Germany*, pages 23–40. Chapman & Hall, April 1998.

[SCEH96] Scott Shenker, David Clark, Deborah Estrin, and Shai Herzog. Pricing in Computer Networks: Reshaping the Research Agenda. *ACM Computer Communication Review*, 26(2):19–43, April 1996.

[SCFJ96] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RFC 1889 - RTP: A Transport Protocol for Real-Time Applications. Standards Track RFC, January 1996.

[SFY95] Jakka Sairamesh, Donald F. Ferguson, and Yechiam Yemini. An Approach to Pricing, Optimal Allocation and Quality of Service Provisioning in High-Speed Packet Networks. In *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95)*, pages 1111–1119. IEEE Computer Society Press, June 1995.

[She95] Scott Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal on Selected Areas in Communications*, 13(7):1176–1188, September 1995.

[SKS00a] Jens Schmitt, Martin Karsten, and Ralf Steinmetz. Design and Implementation of a Flexible, QoS-Aware IP/ATM Adaptation Module. In *Proceedings of ATM'2000, Heidelberg, Germany*. IEEE, June 2000. Accepted for publication.

[SKS00b] Jens Schmitt, Martin Karsten, and Ralf Steinmetz. Efficient Translation of Network Performance Parameters for Transport of IP Packets over Cell-Switched Subnetwork. In *Proceedings of ECUMN'2000, Colmar, France*. IEEE, June 2000. Accepted for publication.

[SKWS99] Jens Schmitt, Martin Karsten, Lars Wolf, and Ralf Steinmetz. Aggregation of Guaranteed Service Flows. In *Proceedings of the Seventh International Workshop on Quality of Service (IWQoS'99), London, UK*, pages 147–155. IEEE/IFIP, June 1999.

[SMMT97] Luca Salgarelli, Martino De Marco, Guido Meroni, and Vittorio Trecordi. Efficient Transport of IP Flows Across ATM Networks. In *Proceedings of IEEE ATM'97 Workshop, Lisboa, Portugal*, pages 43–52. IEEE, May 1997.

[SNNP99] Olov Schelén, Andreas Nilsson, Joakim Norrgard, and Stephen Pink. Performance of QoS Agents for Provisioning Network Resources. In *Proceedings of the 7th IEEE/IFIP International Workshop on Quality of Service (IWQoS'99), London, UK*, pages 17–26. IEEE/IFIP, June 1999.

[SP97] Olov Schelén and Stephen Pink. An Agent-based Architecture for Advance Reservations. In *Proceedings of 22nd IEEE Conference on Computer Networks (LCN'97), Minneapolis, USA*. IEEE, November 1997.

[SPG97] Scott Shenker, Craig Partridge, and Roch Guerin. RFC 2212 - Specification of Guaranteed Service. Standards Track RFC, September 1997.

[SV98] Dimitrios Stiliadis and Anujan Varma. Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.

[SWKS99] Jens Schmitt, Lars Wolf, Martin Karsten, and Ralf Steinmetz. VC Management for Heterogeneous QoS Multicast Transmissions. In *Proceedings of the 7th International Conference on Telecommunications Systems, Analysis and Modelling, Nashville, Tennessee*, pages 105–125, March 1999.

[SZ99] Ian Stoica and Hui Zhang. Providing Guaranteed Services Without Per-Flow Management. Technical Report CMU-CS-99-133, Carnegie-Mellon University, Pittsburgh, USA, May 1999.

[SZN97] Ion Stoica, Hui Zhang, and T. S. Eugene Ng. Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. *ACM Computer Communication Review*, 27(4), October 1997. Proceedings of SIGCOMM'97 Conference.

[TBVZ00] Andreas Terzis, Robert Braden, Steve Vincent, and Lixia Zhang. RFC 2745 - RSVP Diagnostic Messages. Standards Track RFC, January 2000.

[TKWZ00] Andreas Terzis, John Krawczyk, John Wroclawski, and Lixia Zhang. RFC 2746 - RSVP Operation Over IP Tunnels. Standards Track RFC, January 2000.

[Tur86] Jonathan Turner. New Directions in Communications. *IEEE Communications Magazine*, 24(10), October 1986.

[Var96] Hal R. Varian. *Intermediate Microeconomics - A Modern Approach*. W.W. Norton and Company, New York, USA, 1996.

[VHN93] Carsten Vogt, Ralf-Guido Herrtwich, and Ramesh Nagarajan. The Heidelberg Resource Administration Technique - Design Philosophy and Goals. In *Proceedings of Kommunikation in Verteilten Systemen, Munich, Germany*. Springer, March 1993.

[Vic61] William Vickrey. Counterspeculation, Auctions and Competetive Sealed Tenders. *The Journal of Finance*, 16:8–37, 1961.

[VL87] George Varghese and Anthony Lauck. Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility. *Operating Systems Review Special Issue: Proceedings of the Eleventh Symposium on Operating Systems Principles, Austin, TX, USA*, 21(5):25–38, November 1987.

[VLK99] Gustavo De Veciana, Tae-Jin Lee, and Takis Konstantopoulos. Stability and Performance Analysis of Networks Supporting Services with Rate Control - Could the Internet be Unstable? In *Proceedings of the 18th Annual Joint*

Conference of the IEEE Computer and Communications Societies (INFOCOM'99). IEEE Computer Society Press, March 1999.

[WC97] Paul White and Jon Crowcroft. Integrated Services in the Internet: State of the Art. *Proceedings of IEEE*, 85(12):1934–1946, December 1997.

[WGC+95] Ian Wakeman, Atanu Ghosh, Jon Crowcroft, Van Jacobson, and Sally Floyd. Implementing Real Time Packet Forwarding Policies using Streams. In *USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems, New Orleans, Louisiana, USA*, pages 71–82, January 1995.

[WPS97] Qiong Wang, Jon M. Peha, and Marvin A. Sirbu. Optimal Pricing for Integrated-Services Networks with Guaranteed Quality of Service. In Joseph Bailey and Lee McKnight, editors, *Internet Economics*. MIT Press, 1997.

[Wro97a] John Wroclawski. RFC 2210 - The Use of RSVP with IETF Integrated Services. Informational RFC, September 1997.

[Wro97b] John Wroclawski. RFC 2211 - Specification of the Controlled-Load Network Element Service. Standards Track RFC, September 1997.

[WVTP97] Marcel Waldvogel, George Varghese, Jonathan Turner, and Bernhard Plattner. Scalable High Speed IP Routing Lookups. ACM Computer Communication Review, 27(4):25–36, October 1997. Proceedings of SIGCOMM'96 Conference.

[YHB+00] Raj Yavatkar, Don Hoffman, Yoram Bernet, Fred Baker, and Michael Speer. SBM (Subnet Bandwidth Manager): A Protocol for RSVP-based Admission Control over IEEE 802-style Networks. Internet Draft draft-ietf-issll-is802-sbm-10.txt, January 2000. Work in Progress.

[YPG00] Raj Yavatkar, Dimitrios Pendarakis, and Roch Guerin. RFC 2753 - A Framework for Policy-based Admission Control. Informational RFC, January 2000.

[YYP+00] Satyendra Yadav, Raj Yavatkar, Ramesh Pabbati, Peter Ford, Tim Moore, and Shai Herzog. RFC 2752 - Identity Representation for RSVP. Standards Track RFC, January 2000.

[ZDE+93] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network Magazine*, 7(5):8–18, September 1993.

[Zha95] Hui Zhang. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10), October 1995.

Appendices

Appendix A - RSVP Overview

RSVP has been designed as the Internet's resource reservation protocol to support the Integrated Services architecture [BCS94]. The following sections give an overview about its basic operations, excluding the extensions and generalized usage scenarios presented in the main body of the thesis.

Design Goals, Principles and Properties

By using RSVP, hosts are enabled to request a specific QoS from the network. RSVP propagates the QoS request to all the routers along the path and additionally maintains state information within routers and hosts to provide the requested service. It can therefore be regarded as a *state establishment and maintenance protocol* [ZDE+93]. The protocol is placed on top of IP, thus not requiring any routing mechanisms in RSVP itself, but using unicast and multicast routing mechanisms provided by the network layer. RSVP does not transfer application data but operates as a control protocol only. From the beginning, the protocol was designed with *group communication* in mind, and so many design objectives are due to situations arising in multicast data transfers. The designers of RSVP had several goals in mind [ZDE+93]:

1. heterogeneous receivers,
2. dynamic multicast group membership,
3. aggregation of resource reservations,
4. channel changing feature,
5. adaptation to network dynamics,
6. tunable and controllable protocol overhead,
7. independence of other architectural components.

RSVP Operation

In RSVP a data stream is modeled as a simplex distribution tree rooted at the source and extending to all receivers. A source application, of which there can be many, begins participating in a

group by sending a PATH message containing a flow specification to the destination address, be it unicast or multicast. The PATH message serves two purposes:

- to distribute the flow specification to the receivers,
- to establish path state in intermediate RSVP agents to be used in propagating reservation requests along the exact reverse routes.

RSVP does not restrict a source from transmitting data even when no receiver has installed a reservation to it, however service guarantees are not enforced. Also, there may be some best-effort receivers, while other receivers may use reserved resources for QoS-enabled transmission. In case of multicast communication, each receiver must first join the associated multicast group in order to begin receiving PATH messages, yet this is a function of the multicast routing protocol and therefore outside the scope of RSVP.

Each receiver may use information from PATH messages and any local knowledge (computing resources available, application requirements, cost constraints) to determine its QoS requirements. It is then responsible for initiating its own reservation. For that, it generates a RESV message which travels towards the sender along the reverse path of the RESV message. This can be done because the intermediate RSVP-capable routers, which reserve network resources along the subnet leading toward the receiver, can use the established PATH state to propagate the reservation request towards the respective sender(s). Reservation message propagation ends as soon as the reservation encounters an existing distribution tree with sufficient resources allocated to meet the requested QoS, i.e., until the reservation request can be merged into an existing reservation. This receiver-initiated reservation style shall enable RSVP to accommodate heterogeneous receiver requirements and by the merging process it is designed to be scalable for large multicast groups.

The RESV message contains information about the desired QoS and also a filter specification. As well, it determines which out of the following three filters styles is applied:

- the fixed filter style, which can address a fixed number of senders and contains a specific flow specification for each sender,
- the shared explicit style, which can address a fixed number of senders and contains a flow specification for all of these senders,
- the wildcard filter style, which address all senders for a flow and contains a flow specification that is the same for all of these senders.

On their way to the sender, reservation requests have to pass local admission control tests in the routers lying on their path. If the reservation is too demanding for one of these intermediate systems, it is rejected and the receiver that issued the reservation request, obtains an indication of the reservation failure. This is essentially a one-pass or unilateral method of negotiation of the service characteristics, however it is enhanced in RSVP by a mechanism called advertising.

The overall approach to QoS negotiation in RSVP is called *One-Pass With Advertising* (OPWA) [SB95]. Sources of data flows periodically send so-called advertisement messages which are actually contained in the PATH messages of the sender as *ADSPEC* objects. These are used to advertise (beforehand) the end-to-end service that would result from any given service request. On their way to the (potential) receivers, the advertisement messages accumulate information about quantities such as latency, bandwidth and hop count in each router on the path for several categories of service, thereby giving the receivers an idea of what kind of service level they can expect. Thereby, the receiver's task of forming reasonable reservation requests is simplified by the OPWA mechanism.

Since RSVP transmits the PATH and RESV messages periodically, it maintains soft state in the intermediate nodes. While PATH refreshes serve the automatic adaptation to changes in the unicast routing or multicast distribution tree and install path state in any new branches of the tree, RESV refreshes maintain established reservations and incorporate changed receiver reservations, thereby accommodating for dynamic QoS changes. It should be noted that RSVP is no call setup protocol, because reservation requests are issued in parallel to the data transfer, and can hence be made at any time during the data transfer phase.

This refresh-based mechanism allows orphaned reservations and state to be automatically timed out and recovered. However, the proper choice of the refresh interval is still an open issue. This choice affects, of course, the protocol overhead and on the other hand the responsiveness of the protocol with respect to network dynamics and changing receiver requirements.

Design Goals Revised

In this section, each of the design goals mentioned at the beginning of this appendix is briefly discussed with respect to its fulfillment.

Heterogeneous Receivers. Heterogeneous receivers are supported by the receiver-oriented reservation model and the merging of reservations at the outgoing interface towards the next hop.

Dynamic Multicast Group Membership. Dynamic membership in a multicast group is supported by decoupling resource reservation from connectivity in combination with the soft-state approach.

Aggregation of Resource Reservations. Aggregation of resource reservations is enabled both for multiple next hops behind an outgoing interface and for multiple reservations, which are sent towards common receiver(s) according to the reservation style.

Channel Changing Feature. Channel changing features have not been carried over into the current specification of RSVP [BZB+97].

Adaptation to Network Dynamics. By employing the OPWA model in combination with the periodic exchange of refresh messages, RSVP is able to adapt to changes in the network, even if a router is not explicitly informed about such changes.

Tunable and Controllable Protocol Overhead. By changing the period of refresh messages, the robustness in the face of network dynamics can be traded off against the signalling overhead induced by the protocol.

Independence of Other Architecture Components. Because RSVP can inter-operate with arbitry routing and traffic control modules, it is independent of other components in an overall router architecture.

Appendix B - Relations for RSVP Engine

As mentioned in Section 5.2, the design of state objects within the RSVP engine is based on relations describing protocol and respective state information. The full list of relations is presented below.

SessionKey	
Destination Address	*IP address*
Protocol Id	*0 ... (2^8-1)*
Destination Port	*0 ... (2^{16}-1)*

SenderKey	
Source Address	*IP address*
Source Port	*0 ... (2^{16}-1)*

InterfaceKey	
Local Address	*IP address*
Logical Interface Handle	*0 ... (2^{32}-1)*

HopKey	
Interface Address	*IP address*
Logical Interface Handle	*0 ... (2^{32}-1)*

Session	
Session	*SessionKey*
Filter Style	*{WF,SE,FF}*

Path State	
Session Address	*SessionKey*
Sender	*SenderKey*
Previous Hop	*HopKey*
Incoming Interface	*InterfaceKey*
Traffic Specification	*TSpec*
Outgoing Interfaces	*List of InterfaceKeys*

Reservation State	
Session Address	*SessionKey*
Next Hop	*HopKey*
Outgoing Interface	*InterfaceKey*
Applicable Senders	*FilterSpec*
Reservation	*FlowSpec*

Outgoing Interface State	
Session Address	*SessionKey*
Outgoing Interface	*InterfaceKey*
Applicable Senders	*FilterSpec*
Merged Reservation	*FlowSpec*

Previous Hop State	
Session Address	*SessionKey*
Previous Hop	*HopKey*
Forwarded Reservation	*List of FlowDescriptors*

Appendix C - Class Diagram of Traffic Control Interface

In the following diagram, only the classes and their relationships are shown in Coad/Yourdon notation in order to clearly distinguish abstract classes. The list of attributes and methods is omitted for reasons of brevity.

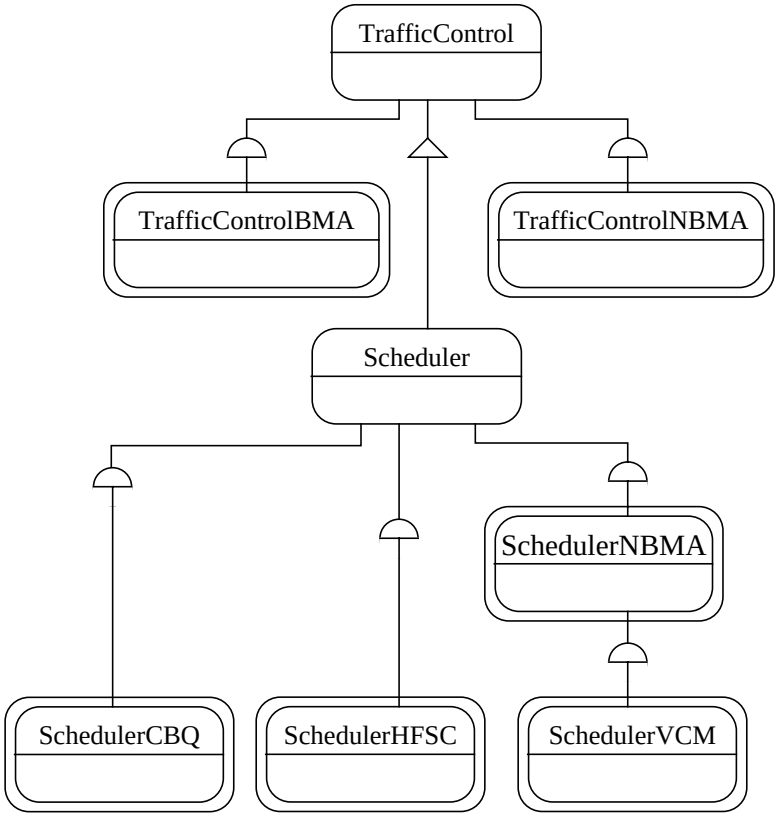


Figure 21: Class Diagram of Traffic Control Modules

Appendix D - IntServ Guaranteed Service Class

Guaranteed Service (GS) as specified in [SPG97] provides an assured level of bandwidth, a firm end-to-end delay bound and no queuing loss for data flows that conform to a given traffic specification (TSpec). The TSpec, which is essentially a double token bucket, i.e. two token buckets in series, is characterized by the following parameters:

- the token bucket rate *r* (in bytes/s),
- the token bucket depth *b* (in bytes),
- the peak rate *p* (in bytes/s),
- the maximum packet size *M* (in bytes), and
- the minimum policed unit *m* (in bytes).

Due to its mathematically provable bounds on end-to-end queuing delay it can be considered to be of high importance for time-critical applications. The mathematics of GS are originally based on the work of Cruz [Cru95] (refined by others, see e.g. [Bou98]) on arrival and service curves. In case of the IntServ specifications the arrival curve corresponding to the *TSpec(r,b,p,M)* is

$$a(t) = min(M + pt, b + rt) \tag{40}$$

whereas the service curve for GS is

$$c(t) = R(t - V)^{+} \text{ where } V = \frac{C}{R} + D \text{ and } R \text{ is the service rate} \tag{41}$$

assuming that the stability condition $R \geq r$ holds. Here, the *C* and *D* terms represent the rate-dependent respectively rate-independent deviations of a packet-based scheduler from the perfect fluid model as introduced by ([PG93], [PG94]).

While the TSpec is a double token bucket it is sometimes more intuitive to regard the mathematical derivations for a simple token bucket *tb=(r,b)* (which is equivalent to assuming an infinite peak rate). In this simplified case, the end-to-end delay bound is given by

$$d_{max} = \frac{b}{R} + \frac{C}{R} + D \tag{42}$$

While for the more complex TSpec as arrival curve it applies that

$$\begin{aligned} p \geq R \geq r \qquad & d_{max} = \frac{(b - M)(p - R)}{R(p - r)} + \frac{M + C}{R} + D \\ R \geq p \geq r \qquad & d_{max} = \frac{M + C}{R} + D \end{aligned} \tag{43}$$

From the perspective of the receiver desiring a maximum queuing delay d_{max}, the rate R (in bytes/s) that has to be reserved at the routers on the path from the sender follows directly from (42) and (43):

for the simple token bucket *tb(r,b)*

$$R = \frac{b + C}{d_{max} - D} \tag{44}$$

for the complete *TSpec(r,b,p,M)*

$$R = \begin{cases} \dfrac{p\dfrac{b - M}{p - r} + M + C}{d_{max} + \dfrac{b - M}{p - r} - D} & p \geq R \geq r \\ \dfrac{M + C}{d_{max} - D} & R \geq p \geq r \end{cases} \tag{45}$$

While the buffer to guarantee a lossless service for the single token bucket is simply b, the buffer formula for the TSpec's double token bucket is more complicated:

$$B = \begin{cases} M + \dfrac{(p - R)(b - M)}{p - r} + C + RD & p \geq R \geq r, \dfrac{C}{R} + D \leq \dfrac{b - M}{p - r} \\ b + r\left(\dfrac{C}{R} + D\right) & \dfrac{C}{R} + D > \dfrac{b - M}{p - r} \\ M + p\left(\dfrac{C}{R} + D\right) & R \geq p \geq r \end{cases} \tag{46}$$

To illustrate the meaning of the C and D terms, one can refer to their values in case of a PGPS (Packetized General Processor Sharing) scheduler [PG93], because they also apply to many other scheduling algorithms [Zha95]

$$C = M; \qquad D = \frac{M'}{c} \tag{47}$$

where M is the maximum packet size of the flow, M' is the MTU and c is the speed of the link. In real routers, there are potentially many other contributions to these error terms, e.g., link layer overhead for segmentation and reassembly in the case of ATM or token rotation times for FDDI or Token Ring.

Personal Data Sheet (Lebenslauf)

Martin Karsten

geboren am 10. Juli 1971 in Kempen-Hüls, jetzt Krefeld

1977 - 1981	Katholische Grundschule Schmalbroich, Kempen
1981 - 1990	Liebfrauengymnasium Mülhausen, Grefrath bei Krefeld
	Abschluß: Abitur 06/1990
10/1990 - 11/1996	Studium Wirtschaftsinformatik, Universität Mannheim
	Abschluß: Diplom-Wirtschaftsinformatiker, 11/1996
02/1991 - 04/1992	Zivildienst, Diakonisches Werk Mannheim
09/1994 - 08/1995	Austauschstudium University of Waterloo, Kanada, Diplomarbeit
seit 12/1996	Wissenschaftlicher Mitarbeiter im Fachbereich Elektrotechnik und Informationstechnik der Technischen Universität Darmstadt