

Quality of Service Agents in the Internet

Olov Schelén

Division of Computer Communication
Department of Computer Science and Electrical Engineering
Luleå University of Technology
SE-971 87 Luleå
Sweden

August 1998

Supervisor
Professor Stephen Pink

Abstract

The Internet is a multi-purpose communication network where users and applications with different service demands share network resources. To meet these demands, mechanisms for quality of service differentiation in the network are proposed. These mechanisms offer service level agreements to the clients by performing traffic scheduling and network resource reservations. This thesis is focused on scalability issues associated with resource reservations and on supporting resource reservations in advance.

We present an admission control architecture where clients can make resource reservations through agents. For each domain in the network there is an agent responsible for admission control. The architecture provides scalable resource reservations for virtual leased lines. One objective of the architecture is to minimize reservation state and admission control overhead in agents. Therefore, reservations from different sources to the same destination domain are aggregated in agents as their paths merge toward the destination. Another objective is to minimize state and packet processing overhead in routers. To accomplish this, routers are pre-configured with a small number of service classes. Agents instruct edge routers to mark packets as they enter the network according to the service agreements. Core routers read the marks when scheduling packets. Thus, there is no information about individual virtual leased lines stored in routers along the path.

In our work on advance reservations, we show that resources can be shared between advance and immediate reservations without being pre-partitioned. Admission control for immediate reservations can use information about resources to be allocated for advance reservations in the near future and, if necessary, makes resources available by rejecting immediate requests. Preemption of immediate reservations is made when the admission control cannot make resources available through rejection of immediate requests. The risk of preemption can be varied by changing the admission control parameters. The results show the cost in terms of resource utilization, rejection probability and preemption probability. We also evaluate the processing and memory cost for advance admission control in a link-state routing domain. The results show that an agent can manage per-link resource reservations and perform path-sensitive admission control using a general-purpose computer.

Contents

Abstract	iii
Publications	ix
Preface	xi
Acknowledgments	xiii
0 Dissertation Overview	1
0.1 Introduction	3
0.1.1 Why quality of service?	3
0.1.2 Why advance reservations?	4
0.1.3 The scalability issue	5
0.2 Organization of the dissertation	6
0.3 Personal contribution	7
0.4 Related work	8
1 Aggregating Resource Reservations	11
1.1 Introduction	13
1.2 Reservation Model	15
1.3 Aggregation	16
1.4 Policing and classification	18
1.5 Interoperation with RSVP	19
1.6 Conclusion	19

2	An Agent-based Architecture for Advance Reservations	21
2.1	Introduction	23
2.2	Overview of the architecture	25
2.2.1	Mixing advance and immediate admission control	27
2.3	Advance reservations over several domains	27
2.3.1	Unicast	27
2.3.2	Multicast	28
2.4	Packet classification	29
2.5	Security and policing	31
2.6	Setting up forwarding state in routers	32
2.7	Routing performed by agents	32
2.8	Comparison with the RSVP policy architecture	33
2.9	Conclusions	34
3	Resource Sharing in Advance Reservation Agents	37
3.1	Introduction	39
3.2	The agent architecture	40
3.3	Admission control for immediate and advance reservations	46
3.3.1	Parameter-based or measurement-based?	46
3.3.2	Service model	47
3.3.3	Bookahead and lookahead	48
3.3.4	Admission control criteria	49
3.3.5	Reservation state	51
3.4	Simulations	51
3.4.1	Simulation parameters	51
3.4.2	Advance reservation requests	52
3.4.3	Total utilization	53
3.4.4	Rejection and preemption probability	55
3.4.5	Lookahead time	56
3.4.6	Evaluation	58
3.5	Conclusion	59
4	Advance Reservations for Predictive Service in the Inter-	
net		61
4.1	Introduction	63

4.2	Framework	64
4.3	Duration Intervals	65
4.4	Admission Control Decision for Advance Reservations	66
4.5	State Requirements	69
4.6	Simulations	70
4.7	Setup protocols for advance reservations	77
4.8	Related Work	79
4.9	Further work	80
4.10	Conclusions	80
5	Performance of Admission Control Agents	83
5.1	Introduction	85
5.2	Prototype Setup	86
5.3	Scope	86
5.4	Advance admission control	87
5.4.1	Continuous or slotted time	87
5.4.2	A binary search tree over time	88
5.4.3	A segment tree over time	89
5.5	Immediate admission control	91
5.6	Evaluation of single-hop admission control speed	91
5.6.1	Dependence on the number of rejections	92
5.6.2	Dependence on the number of reservations	93
5.6.3	Dependence on insertion order	93
5.6.4	Dependence on the reservable time frame	93
5.6.5	Dependence on the length of requested durations	93
5.6.6	Memory requirement	94
5.7	Routing calculations	95
5.8	Evaluation of multi-hop admission control speed	96
5.8.1	On-demand route computation	96
5.8.2	Pre-computed paths	98
5.8.3	Memory requirement	98
5.9	The cost of request-reply transactions	99
5.10	Conclusion	100
	Bibliography	101

Publications

This thesis is based on papers that have been published in conferences or journals. Below, all publications concerning this work are listed and organized in five parts. The first paper listed in each part is included in this thesis.

Part 1

- Olov Schelén, Stephen Pink: Resource Reservation Agents in the Internet. In *Proceedings of 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98)*, Cambridge, United Kingdom, July 1998, pp. 153–156.
- Olov Schelén, Stephen Pink: Aggregating Resource Reservations over Multiple Routing Domains. In proceedings of IFIP Sixth International Workshop on Quality of Service (IWQoS'98), Napa Valley, California, May 1998, pp. 29–32.

Part 2

- Olov Schelén, Stephen Pink: An Agent-based Architecture for Advance Reservations. In *Proceedings of IEEE 22nd Annual Conference on Computer Networks (LCN'97)*, Minneapolis, Nov 1997, pp. 451–459

Part 3

- Olov Schelén, Stephen Pink: Resource sharing in advance reservation agents. To appear in *Journal of High Speed Networks* special issue on Multimedia Networks.
- Olov Schelén, Stephen Pink: Sharing Resources through Advance Reservation Agents. In *Proceedings of IFIP Fifth International Workshop on Quality of Service (IWQoS'97)*, New York, May 1997, pp. 265–276.

Part 4

- Mikael Degermark, Torsten Köhler, Stephen Pink, and Olov Schelén. Advance Reservations for Predictive Service in the Internet. In *ACM/Springer Journal of Multimedia Systems*, 5(3), May 1997.

- A shorter version of this paper is: Mikael Degermark, Torsten Köhler, Stephen Pink, and Olov Schelén: Advance Reservations for Predictive Service. In *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)*, Durham, New Hampshire, April 1995, pp. 3–14.

Part 5

- Andreas Nilsson, Joakim Norrgård and Olov Schelén: Performance of Admission Control Agents. *Not yet published.*

Preface

This work has been carried out during four years at Luleå University of Technology. The first incentive came from my supervisor Stephen Pink, who first visited Luleå in the spring of 1994. I had then for some years been working as a lecturer in functional programming, structured programming, real-time systems, and computer communication. My previous research area involved process algebra, proof checkers and declarative programming. Steve inspired Mikael Degermark and me to begin graduate studies in computer communication.

In the fall of 1994, we started work on advance reservations. In the spring of 1995, this resulted in one of the papers listed in **part 4**. In the spring of 1996, I spent six months at University of Massachusetts at Amherst. Then, the first ideas on an agent-based architecture for advance reservations were born. An objective was to keep reservation state out of the routers. I investigated agent-based admission control methods and their impact on network utilization. This resulted in one of the papers listed in **part 3**.

During the spring of 1997, I wrote up my ideas on an agent architecture for admission control and this resulted in **part 2**. At the same time there was a growing concern in the research community about the scalability problems of RSVP. Work on differentiated services started in the IETF, which was very much in line with our ideas. This encouraged us a lot. In the fall of 1997, I worked more specifically on aggregating reservation state in agents. This resulted in the papers listed in **part 1**.

In the spring and summer of 1998, we developed a prototype agent. We experimented and measured the performance of admission control in our prototype. This resulted in **part 5**.

Acknowledgments

Most of my work has been carried out at the Department of Computer Science and Electrical Engineering at Luleå University of Technology. I would like to thank all my colleagues there for providing a stimulating work environment.

My supervisor, Professor Stephen Pink, has been the primary source of inspiration, knowledge and pressure. He has also provided contacts with people in the international research and development community. Thank you Steve for everything. Our head of the department, Professor Svante Carlsson, has also supported me. Thank you Svante for encouragement and discussions.

I am grateful to my co-authors. Especially Mikael Degermark, who has very good ideas and lots of energy. You have been the best example of how to transform oneself for succeeding with a Ph.D. Thank you Mikael. I would also like to mention Joakim Norrgård. We worked together a lot during the summer before I was scheduled to finish the thesis. Thank you Joakim.

Professors Jim Kurose and Don Towsley hosted my five months at University of Massachusetts at Amherst. Thank you for the support with everything while I was there. I also want to thank the people at Amherst for providing a nice work environment.

In 1985-1988, I pursued a Licentiate of Technology degree on interactive proof checkers for process algebra. During that period of time I was brought into the foundations of computer science. Especially I want to thank Björn von Sydow, Lennart Andersson and Anders Grennberg for the guidance and Nils-Olof Forsgren for co-authoring the thesis.

Research in Computer Communication started in Luleå when the Centre for Distance-spanning Technology (CDT) was formed. The founders of CDT have my gratitude. The funders of my work have been the CDT, Ericsson, Telia Research, TFR, and Hewlett Packard.

I want to thank my relatives, especially my parents Ivy and Jan who have always supported me. Finally, I give great thanks to my wife Karin and our children Oscar and Henny. You have supported me and inspired me endlessly. Because of you, I have had a fantastic time.

To Karin, Oscar and Henny

Part 0

Dissertation Overview

0.1 Introduction

The Internet has become the ubiquitous network for global data communication and may also become the same for the whole of telecommunication. This means providing service for users with different demands for network service quality and different willingness to pay for higher quality. Also, there is a wide range of applications that require various network performance. To meet these requirements, the Internet needs support for quality of service (QoS) differentiation. This thesis describes an architecture and some accompanying mechanisms, based on agents, for supporting QoS differentiation in a large network such as the Internet.

To set up services, clients and providers establish *service level agreements* that give clients different access to resources when there is contention for using them. Providers meet the service demands by scheduling their resources for various purposes. Resources are finite and the objective is to schedule these resources so clients have the best overall network utility. There are cases when resources are not, or cannot be, scheduled to meet further service requests while still meeting already established service level agreements. Therefore, providers will occasionally have to reject some service requests. For this reason, service negotiation is also known as *admission control*. In this thesis, we define an admission control architecture where clients can request their network resources immediately as they are needed or ahead of time through *advance reservations*. Our architecture supports resource reservations for *virtual leased lines*. The architecture scales to large networks and large number of reservations by aggregating reservation state for several virtual leased lines.

The purpose of this introduction is to raise and motivate the issues that are covered in this thesis. In section 0.1.1, I motivate why there is a need for quality of service and resource reservations at all. In section 0.1.2, I motivate why there is a need for supporting resource reservation in advance. In section 0.1.3, I describe why scalability issues are crucial for resource reservations in the Internet.

0.1.1 Why quality of service?

Traditionally, the Internet has offered a single level of service known as *best-effort* service. In a pure best-effort model, the resulting quality of service depends on the current traffic load over the links involved. The only way to provide different service levels is by having various bandwidths of the access links. In the rest of the network, there are no mechanisms for providing different service levels. This model was sufficient for a homogeneous user community and a small set of applications that could adapt and do relatively well under various conditions. The network could be provisioned to deliver an acceptable quality of service to everybody, most of the time.

Now, the Internet is used by many applications that have very different requirements on network service quality. Interactive real-time applications (e.g., Internet telephony, multimedia conferencing etc) require low round-trip delay and a minimum throughput to work well. These applications are run by users with very different demands (e.g., business users, home entertainment users, etc). It is hard to determine

a single level of service to fit all users and applications in terms of price and performance. Multi-level service differentiation can be an aid to meet these new demands.

To provide an incentive for clients to choose among different services, there must be some sort of charging scheme, to provide the options of paying less for limited service or paying more for extended service. A goal of service differentiation is to provide the services needed by clients at a cost related to the resources made available to them. This will encourage efficient network usage by providing more resources to those who need them and are prepared to pay for them, and by providing cheaper network access to those who need little resources or use them at off-peak hours.

0.1.2 Why advance reservations?

Service agreements involving a minimum expected level of service must involve some kind of admission control to make sure that there are sufficient resources in the network to meet the agreements. Admission control gives the provider an opportunity to reject service to new clients so that committed service can be given to clients that were already granted admission. Clients, once admitted, have some expectation on the service. In the traditional telephone network, there is only one level of service. Admission control is performed as we initiate a call by dialing a number. Once admitted, clients know that they have sufficient quality for voice communication and that they may hold on to the call as long as they like.

This service model may not be appropriate for the Internet. First, there is a new set of real-time applications involving many users. Carrying out sessions (e.g., conferences, lectures, games, etc) with these applications often require a certain degree of planning so that the participants are attending at the same time. Booking network resources should be part of this planning. It is unfortunate if a planned event has to be cancelled with late notice because it turns out there are insufficient network resources at the time.

Second, there is an increasing demand for long-term planning of trunk network capacity. This is often provided through virtual leased lines and virtual private networks. These services today involve tunneling and manual configuration of routers along the paths. There is a demand for having automated setup of such services. Trunk bandwidth contracts generally involve planning of network resources over time (e.g., time-limited contracts, various bandwidths at different times of the day etc). It is reasonable that admission requests can include timing information to allow this.

In this thesis we meet the demands for planning network resources over time by exploring support for advance reservations. In addition to the traditional so-called *immediate* admission control (where no timing information is included) we provide *advance* admission control. Advance admission control allows clients to plan their activities provided they give a duration (i.e., a starting point and a finishing point) of the service level agreement. This also allows providers to schedule their resources over time and predict when the total capacity will be insufficient.

0.1.3 The scalability issue

Service differentiation provides some obvious advantages, but there are drawbacks too. The primary objections concern cost in terms of increased network complexity, more processing overhead, more storage of reservation state etc. It is currently a research issue whether these costs can be justified [12]. The opponents argue that efforts are better spent by increasing the overall network throughput and by focusing on better network management [47]. In this thesis we suggest an architecture for scalable resource reservations and evaluate the performance and costs in terms of network utilization, processing overhead, state etc.

The background for scalability concerns is found in the architecture of the network. The Internet is a datagram network, where each packet is processed individually by the nodes in the network. For best-effort service there is no connection setup phase and the network does not rely on any connection state to perform its task (although there may be some state cached to increase the processing speed). This model scales well for supporting large numbers of short-lived sessions. It is challenging to introduce support for service differentiation in this model without breaking the present Internet's capacity to scale to many users and different kinds of usages.

There have been many proposals based on introducing per-flow reservations in the network. For the Internet, the most well known approaches are the IETF standards for Integrated Services in routers [79, 78, 36] and the end-to-end reservation setup protocol RSVP [11]. These proposals result in per-flow state in the network nodes along the path of the flow. An advantage with such an approach is that fine-grain service agreements can be supported by having per-flow packet scheduling in routers. However, it has been argued that these models are too complex and do not scale well enough to be used in the backbone of the Internet [52].

This has resulted in a new approach being taken in the IETF, known as *differentiated services*. The guideline is to provide scalable QoS support by having no per-flow state in routers. The basic idea is that IP packet headers include a small label (known as the diffserv field) that identify the service they should be given. Core routers are configured with a few service classes and the packet labels are used to map the packets into the right service class. The architecture relies on packet markers and policing functions at the edges of the network to ensure that the services are used according to the service level agreements made with the clients. Differentiated services may provide a range of services, both relatively defined and absolutely defined. To provide absolutely defined (minimum expectation service), resources must be administrated externally from the routers to make sure that the service commitments can be met. This can be arranged both manually through network managers or automatically through admission control units.

In this thesis we suggest a QoS architecture that can provide *virtual leased lines* over differentiated services by providing admission control in agents, i.e., the reservation state is stored and aggregated in agents. We show that path-sensitive service level agreements can be supported with favorable scalability properties. The architecture does not rely on per-commitment state in the routers.

0.2 Organization of the dissertation

This thesis is made up of five parts. All parts, except part 5, have been published elsewhere as separate papers. They are reproduced here in their original form with the following exceptions. The layout of the text is made as uniform as possible (which involves cosmetic changes only). The numbering of sections, equations, figures etc, has been changed to obtain a common numbering scheme throughout the thesis. A common bibliography is used for the whole thesis instead of a separate reference list for each paper.

Part 1:

In this paper, we describe how reservations for virtual leased lines can be supported. Specifically, we focus on aggregating reservation state and show a scalable way of providing path-sensitive end-to-end resource reservations. The reservation state is aggregated as reservations along different paths merge towards a given destination domain. Virtual leased lines are uni-directional, connecting two network domains. Several lines can be combined to build virtual private networks. The purpose of this paper is to define the admission control architecture and describe how we meet the scalability issues for resource reservations.

Part 2:

In this paper, we present general ideas on an agent architecture. Agents have a copy of the same topological database as the routers have. This can be used for many purposes: First, agents can perform routing calculations and set up forwarding tables (forwarding state) in routers. Second, they can do QoS routing that is coupled with admission control, if there is an alternative route that could be used to grant an admission request. Agents distribute packet markers to edge routers to label packets that are to be routed alternatively and setup forwarding state in the routers along the alternative path to recognize this label. Furthermore, we discuss multicast reservations in agents and the cost of packet classification in routers. The purpose of the paper is to give a broader view on what can be supported by an agent-based admission control architecture. The reader may notice that the terminology used in the paper is slightly different from what has been established in the IETF differentiated services group. This is because the paper was written before the IETF work started.

Part 3:

In this paper, we focus on immediate and advance admission control. The paper starts with an overview of our agent architecture (section 3.2). This overview is extracted from part 1 of this thesis. After that, we describe the admission control model and criteria. We show that network resources can be shared between immediate and advance reservations without being pre-partitioned. Admission control for immediate

reservations use information about resources to be allocated for advance reservations in the near future. We explore the effects of providing advance reservations with this model. The results show the cost in terms of resource utilization, rejection probability and preemption probability. This paper contains the core results concerning advance reservations in our architecture.

Part 4:

In this paper, we explore advance reservations in a different framework. We extend a measurement-based admission control scheme, known as predicted service, with support for advance reservations. Measurement-based admission control is performed in the routers along the path of a flow (i.e., an application data stream). In order to prevent preemption, we require that all immediate and advance reservation requests include a duration. The protocol used in this paper for setting up reservations is RSVP. As RSVP is soft-state, we devise methods to reduce the signaling overhead during the booking time. Although we don't in general recommend RSVP as the signaling protocol for advance reservations, we show how RSVP could be adapted for this duty in limited contexts.

Part 5:

In this part, we present an implementation of our architecture (as described in parts 1–3). We provide performance results for admission control in a link-state routing domain. As an example, we do this over the network covering Luleå University of Technology. The implementation runs on a Pentium 300MHz using the NetBSD operating system.

We present performance results on the speed of admission decisions for end-to-end reservations within a routing domain. This includes comparing two different data structures for advance reservations, evaluating route decision speed, and evaluating protocol processing overhead for agents receiving admission requests over the network. The purpose of this paper is to provide measures showing that admission control agents have the capacity to handle large routing domains.

0.3 Personal contribution

In parts 1 - 3, I am mainly responsible for ideas and results. I am also responsible for writing the papers. However, I have had input from co-authors concerning presentation and writing style. The programming and simulations in part 3 are entirely my work. In part 4, the general ideas are a result of joint work. I am primarily responsible for writing sections 4.2, 4.3, 4.4 (except 4.4.2), and 4.7. In part 5, I am mainly responsible for ideas around the experiment setup and for the the writing. I am partly responsible for programming and running experiments.

The scientific contribution of our work is that we have been one among only a few others who have pioneered the field of advance reservations. We have also identified the need for a new admission control architecture that separates packet forwarding performed in routers from admission control and service level agreements performed in agents. We have showed how path-sensitive admission control can be supported and aggregated in a datagram network without needing per-flow reservation setup along the whole path.

0.4 Related work

Resource reservation architectures

There have been a number of resource reservation architectures developed over the last few years [13]. For the Internet, one of the most well-known is the *Integrated Services* (intserv) architecture [41] standardized by the Internet Engineering Task Force (IETF). In this architecture, admission control and packet scheduling is provided in routers for individual flows (i.e., application data streams). The resource reservation setup protocol *RSVP* [66] is used for setting up reservations end-to-end by carrying admission requests to routers along the path of a flow. RSVP supports receiver-initiated soft-state reservations (i.e., reservations are periodically refreshed by receivers). Reservations that are no longer refreshed will automatically time out and disappear. This makes RSVP suitable for supporting resource reservations in dynamic multicast groups. However, the per flow semantics has been identified as a scalability problem [52] and therefore the authors of RSVP and others have recommended that this protocol should currently not be enabled in backbone routers. The scalability problem is most obvious for unicast reservations. This is because the architecture adds per-flow state and classification overhead in routers that originally supported stateless best-effort unicast routing. As we describe in section 1.5, the extra overhead can be justified for multicast reservations. To operate in the global Internet, a mechanism for charging and accounting is required. A *policy architecture* supporting this is under development.

Recently, there has been a number of protocols proposed that are more light-weight than RSVP [1, 24, 59]. These protocols support sender-initiated unicast reservations. There are also proposals for avoiding RSVP overhead by setting up reservation tunnels between edge points of a domain [75]. This kind of aggregation does, however, not easily extend over administrative borders.

The IETF is currently defining a new framework for scalable service differentiation known as *Differentiated Services* (diffserv) [22] [14]. Instead of per-flow commitments, routers support a small number of service classes. Service differentiation is provided by introducing a label in packet headers that is used by core routers to determine which service class packets belong to. The labels are set at the edges of the network, either by applications or by edge routers. Edge routers also perform policing to check that clients follow their part of a service level agreement. The differentiated services

work covers only router functionality and does not include support for establishing service level agreements or for configuring police points at the edges of the network. This can be done manually or through a separate mechanism. Our admission control architecture is an example of a mechanism providing high quality service level agreements and high utilization by collecting topology information, recording per-link reservations, and performing path-sensitive admission control. In part 1, we present a solution for providing virtual leased lines over domains of diffserv capable routers. In [57], another architecture for diffserv is proposed where admission control is performed by bandwidth brokers. The concept of a bandwidth broker is similar to our reservation agent. However, the description is general and it is not specified whether brokers collect topology information or record per-link reservations.

The ATM Forum have developed standards for QoS negotiation and signaling in *ATM* (Asynchronous Transfer Mode) networks [2]. A major difference between ATM and IP is that ATM is connection oriented. In ATM, QoS signaling is initiated from the end-points at connection setup. Our architecture is intended for connection-less networks avoiding per-connection state in network nodes.

Supporting advance reservations

The above mentioned architectures do not include reservations in advance. One of the most well-known architectures supporting advance reservations is the *TENET* protocol suite, developed at UC Berkeley. The TENET suite was one of the first resource reservation architectures proposed and implemented for providing guaranteed quality of service [27, 5]. Support for advance reservations was added by partitioning the network resources so that one partition is available for reservations in advance and the other is available for immediate reservations [26]. In our architecture, as described in part 3, we avoid pre-partitioning resources to allow better utilization of resources.

Researchers at the IBM European Networking Center have also done work on advance reservations. In [77], an overview of the field is given and some issues concerning advance reservation systems are presented. Other work by that group described extensions for ST-2 [20] to support advance reservations [64]. These approaches do address advance reservations but do not cover the architectural issues of scalability and state explosions in routers.

Recently, a new proposal for supporting advance reservations in RSVP came out of the Information Sciences Institute at University of Southern California [7]. Their architecture is based on the most distinct features of our architecture, i.e, they rely on topology aware advance reservation servers that perform path-sensitive admission control within their routing domains. A motive (as in our architecture) is to avoid having any reservation state in routers until the traffic starts. The major difference compared to our architecture is that they use RSVP to claim advance reservations and to setup per-flow reservation state in routers along the path when a session starts. End-systems send reservation messages including a duration, thereby claiming that

resources were reserved in advance. Routers receiving these reservation messages check the claims by sending queries to their advance reservation servers.

In [76], support for advance reservations based on the theory of effective bandwidths is explored. The idea is to allow higher utilization by providing advance admission control using statistical guarantees. When resources turn out to be over-utilized, the most recently booked advance reservation is preempted. In our architecture, we do not preempt advance reservations (only immediate open-ended reservations may be preempted).

Part 1

Aggregating Resource Reservations

Resource Reservation Agents in the Internet

Olov Schelén and Stephen Pink

Computer Science and Electrical Engineering

Luleå University of Technology, SE – 971 87 Luleå, Sweden

{olov, steve}@cdt.luth.se

Abstract

We present an architecture where clients can make resource reservations through agents. For each domain in the network, there is an agent responsible for admission control. The architecture provides scalable resource reservations for uni-directional virtual leased lines. Reservations from different sources to the same destination are aggregated as their paths merge toward the destination. When there are sufficient resources in a destination domain, prefix aggregation is performed. Thus, all reservations for a given destination domain are aggregated as their paths merge. Agents are responsible for setting up police points at edges for checking commitments.

1.1 Introduction

We are designing a resource reservation architecture where clients make admission requests through *agents* [67, 68, 69]. For each routing domain in the network there is an agent responsible for admission control. Each agent knows the topology and static link resources in its domain (figure 1.1). The agent is an end-system that is configured for passively participating in a link-state routing protocol (e.g., OSPF) where each participating router has an identical topological database over the domain. Thus, an agent obtains a copy of the topological database with little signaling overhead, i.e., link-state advertisements are sent from the nearest router to the agent. Agents retrieve link properties, such as static bandwidths, by querying routers seen in the topological database. For this, we use a network management protocol (e.g., SNMP). Queries are done at startup and when topology changes are detected by the routing protocol. Routers need no extensions to allow agents to build a resource map through the management protocol. Agents use the resource map to perform admission control in their domain.

Admission requests contain the bandwidth to be reserved, a source and a destination address. For each request, agents can find where traffic will enter a domain

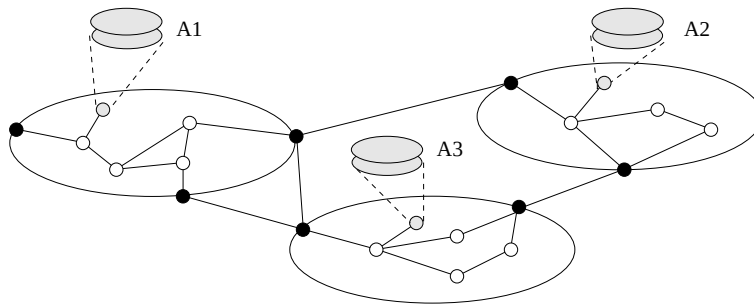


Figure 1.1: Reservation agents and their domains

and therefore agents can setup police points in edge routers checking that incoming traffic conforms. Packets using reserved resources are *marked* by applications or edge routers and checked by police points when entering a domain. Similar ideas, known as *differentiated services*, have been proposed in [14, 57].

In the differentiated services model, packets are marked to obtain either low drop probability or high scheduling priority. In this paper, we abstract from these details and use the term *priority packets* to denote marked packets, and the term *priority traffic* for all priority packets collectively. A key point of differentiated service and our agent architecture is to avoid non-scalable signaling state and expensive per packet processing in routers. Therefore resource reservations are handled by agents (end systems), and packet classification in routers involves only a small number of fixed service classes.

Independently of which packet scheduling model is used, there must be an admission control architecture to make sure that the amount of priority traffic is low enough to receive good quality of service. One issue is whether the admission control architecture should reserve resources over the links that are actually going to be used, or just limit the overall rights to send priority traffic. A scheme that doesn't consider exactly where traffic will go cannot give reliable commitments without very low utilization. Thus, it is likely that such a scheme would need to gamble and accept occasional service loss.

In this work we focus on agent-based admission control where the objective is to administer resources exactly, i.e., admission control in an agent maintains information about reserved resources on each link in its domain. Our objective is to find out if such a scheme can scale. The architecture offers a service comparable to unidirectional *virtual leased lines*. In this model, requests may be immediate and open-ended or made in advance by including starting time and duration for the reservations. In [68] we show the implications of supporting a mix of immediate and advance reservations.

1.2 Reservation Model

An admission request can be directed to any agent, e.g., an agent that manages a user's account. Each agent sets up reservations between any two points in the network by invoking other agents. Thus there is *no difference* between reservations for *sending* or *receiving* data, reservations for *nomadic computing*, or reservations paid by a *third-party*.

Each reservation request contains a bandwidth to be reserved, a source and a destination address. The source address determines the point where the reservation starts (i.e, the point where traffic using the reservation will enter) and the destination address determines the point where it ends. An agent receiving a request first considers whether the starting point is in its domain. If the starting point is *not* in its domain, it finds an agent closer to the starting point and repeats the request with that agent. Technically speaking, it is possible to directly identify the agent that is responsible for the source domain, but in practice clients may prefer using a particular agent. Also, agents may prefer to deal with only a few adjacent agents. This means that a request is repeated in several steps. In figure 1.2, it is shown how a request is issued by node Dx for reserving resources from Ax to Dx.

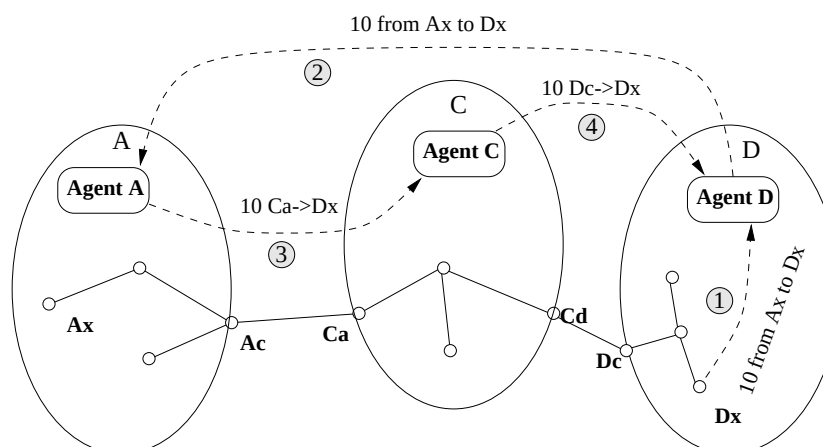


Figure 1.2: A receiver Dx requesting a reservation to receive data from Ax

If an agent finds that the starting point given in the request is in its domain, admission control is performed on the links from the starting point towards the destination, defined by the routing protocol. This is possible since all routers and the agent in a domain have the same topological database. Thus, an agent can perform admission control on the links from the starting point to an edge router in its domain.

An agent cannot perform admission control beyond its domain. Therefore, if the destination is outside of the agent's domain it must request a reservation with the

neighboring agent, giving the edge point where traffic will cross the borders as the source address of the reservation (figure 1.2). Without distinguishing between reservations to different destinations it would be impossible for agents to grant reservations to other domains. This is because there is no way to make sure that there are sufficient reservations along any path in other domains further downstream (unless we accept very low utilization for priority traffic).

1.3 Aggregation

If reservations were always done “on demand” there would be signaling between all neighboring agents along the path for each end-to-end admission request. Therefore, our architecture supports several ways of aggregating reservation state in agents.

To grant requests spanning many routing domains, there must be a commitment from all agents along the path involved in the request. When an agent is making a reservation with a neighboring agent, the source address included in the request determines the point (router interface) where the traffic will cross the borders, i.e., enter the domain of the neighboring agent. The neighboring agent does not care where traffic came from originally. Thus, priority packets can use reserved resources as long as the destination address matches the reservation. This allows agents to aggregate traffic with different sources into one single reservation made with a neighboring agent.

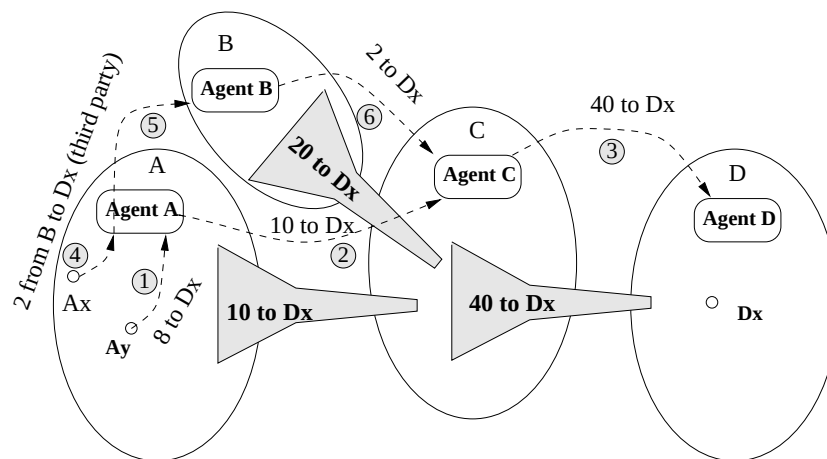


Figure 1.3: Funnels and aggregate reservations for one destination

An end-to-end resource reservation can be seen as a number of consecutive *funnels* (figure 1.3). Priority packets poured into any of these consecutive funnels use reserved resources all the way to the destination. Agents may aggregate several requests with different source addresses into the same funnel, as long as they all specify

the same destination. Agent C (in fig. 1.3) may set up a funnel starting at its edge by pre-reserving resources with neighboring agent D to a distant destination and then aggregate requests for resources between domains C and D into the established funnel. This bulk reservation implies that downstream agents only keep state for the aggregate. The agent that decides to aggregate into a bulk reservation keeps information about its individual commitments constituting the aggregate. Aggregation by merging reservations towards each destination can be done with full control over the resources. The funnels form a *sink-tree* with the root towards the destination. At each merging point, the reserved bandwidth on the outgoing link will be equal to, or larger than, the sum of the reserved bandwidth on the incoming links.

Requests that specify different destinations can not easily be aggregated into a funnel in the general case. If we allow funnels to split out in different directions, upstream agents must keep information about which destinations are involved and how resources are divided between them. This means that the aggregate cannot be seen as one unit. However, a reservation can cover different destinations if resources for the whole aggregate are reserved to any of the involved destinations. In addition, there should be a scalable way of representing the set of destination addresses. Both of these requirements are often met for well provisioned destination domains in the Internet.

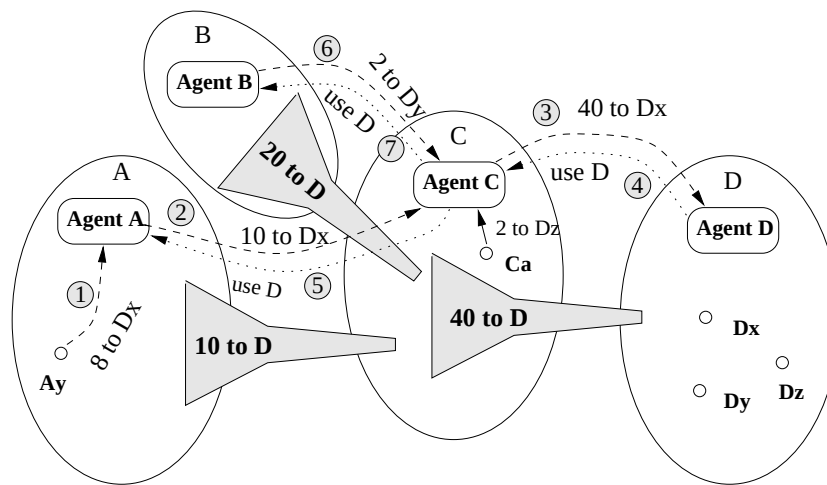


Figure 1.4: Funnels and prefix aggregation

Therefore, we allow aggregation to be increased by *generalizing* a reservation to be valid for all destinations within the same destination domain. It is only the agent responsible for the destination domain that can judge if resources are sufficient for allowing generalization. Consider the example in figure 1.4. Agent D can judge if the resources in D are sufficient to let incoming funnels be valid for the whole domain. Thus, agent D decides when a reservation request for a particular destination node

within D can be replaced with a reservation for any node in domain D (or a subnet within domain D).

In IP networks an *address prefix* is often used to represent a domain (individual nodes within the domain are distinguished by the remaining suffix). An address prefix is represented by an IP address and an associated mask telling how many bits that are part of the prefix. An agent generalizes a request by replying with an *address prefix* to be used as destination address for the reservation (figure 1.4). The aggregation that results from generalizing a reservation is therefore called *prefix aggregation*.

Clients or agents that have been granted a prefix aggregated reservation can use it for any destination matching the prefix of the reservation. Agents providing prefix aggregated reservations must provide the committed bandwidth of priority traffic to any destination in the domain denoted by the prefix. A conservative agent would therefore allow prefix aggregation up to a value equal to the bandwidth of the narrowest link. In practice we believe that this can be quite useful for well provisioned local area networks.

To sum up, aggregation is happening in the agents as routers are not dealing with admission control at all. The motivation for aggregation may be less when reservation state is stored in agents with plenty of memory and disc. However, a per-flow reservation model has inherent scaling problems, both in terms of state and processing cost in agents. Therefore, it is important to show that agents can provide exact resource reservations with good scaling properties. Since agents have immediate access to the routing database, they can also handle changing routes and topologies with an outcome as favorable as possible.

1.4 Policing and classification

The source address in an admission request is used by agents to set up an ingress police point, i.e., in the router closest to the source. Egress policing can be setup by using the destination address, i.e., the routing protocol can find the point where traffic is leaving the domain. By policing at the edges, a provider can check that the rate of priority traffic stays within commitments. At each edge point, there may be several commitments to different destinations. Policing of individual commitments would therefore involve classifying priority packets by their destination addresses before measuring used bandwidth. Although the number of commitments are kept low through aggregation, policing is demanding as it involves per-packet processing compared to reservation setup which involves only per-commitment signaling. At the edge routers, close to the original sources of data, per-commitment policing may be affordable as there are generally few commitments and a low aggregate data rate. If all agents police correctly at the original sources, and we have an architecture that reserves resources along exact routes end-to-end, there is little need for policing in the backbone.

However, a model relying on policing at original sources only, would be volatile as it is based on global trust. Therefore, we suggest adding a model for scalable

policing at edges of backbone domains. The method is to check a dynamic subset of the commitments. The agent can go through its commitments associated with the actual edge point and set up close policing by matching different destinations one-by-one, or randomly, in the router. The key idea is that the agent has knowledge about any policing that could be done, but saves per-packet policing overhead in the routers by not checking all commitments at once. When violating traffic is found, it is reported to the agent (the agent sets up policing and a trap with the router by using a management protocol). When an agent gets a report from a police point about a violated commitment, the agent should report it back to the upstream agent. The upstream agent has knowledge about the constituent parts that were aggregated into the violated commitment. By performing the same method at each agent, the police point for catching the violating traffic will gradually move back towards the sender (as far as there are cooperating agents), where it should have been regulated in the first place. For managing policing from agents, one architectural requirement is to standardize an interface to allow trusted agents to set up police points in their routers.

1.5 Interoperation with RSVP

In RSVP [80], reservations are made for individual data flows (i.e., application data streams). In that model, per-flow reservations are maintained in the routers. This results in scaling problems due to reservation state and packet classification overhead in routers. This model inherently relies on per-flow semantics and therefore it is hard to perform aggregation across different administrative domains. This has led some of the RSVP designers and others to advise keeping RSVP out of the backbone.

Having per-flow reservation state in the routers, as in RSVP, is not advisable for unicast reservations. Even if aggregation is supported over some parts of the path, there would generally be an explosion in state and packet processing overhead compared to ordinary best-effort routing. For multicast, the overhead for RSVP can probably be motivated. Already with best-effort multicast, the network trades off bandwidth savings for per-flow state in the routing table. The extra state for RSVP would probably not add too much overhead. One should, however, make sure that multicast and RSVP are used only when resulting multicast trees are sufficiently dense (branched) to justify the control state. Therefore, the agent architecture can be used for unicast reservations, while RSVP is used for multicast reservations.

1.6 Conclusion

The agent architecture can provide resource reservations in a global network for offering virtual leased lines from any point to any point (i.e., independently of where reservation requests originate). The key ideas of our architecture are *aggregation* as paths merge towards destinations, *prefix aggregation* over destination domains, *advance reservations*, and minimal *flexible policing*.

The architecture separates functionality between packet forwarding in routers and QoS negotiation in agents. For enforcement, agents must set up police points in their edge routers.

Part 2

An Agent-based Architecture for Advance Reservations

An Agent-based Architecture for Advance Reservations

Olov Schelén and Stephen Pink

Computer Science and Electrical Engineering

Luleå University of Technology SE – 971 87 Luleå, Sweden

Email: {olov, steve}@cdt.luth.se

Tel: +46 920 91000 Fax: +46 920 72821

Abstract

We propose an architecture where clients can make advance reservations through agents. For each routing domain in the network there will be an agent responsible for admission control on behalf of the routers in the domain. Requests involving several routing domains are forwarded for admission control with agents along the path for the requested service. Agents maintain hard reservation state using a reliable protocol for agent intercommunication.

Agents start allocating resources for advance reservations in the routers by setting up forwarding state shortly before resources are needed for packet forwarding. Resources are made available for advance reservations by means of rejecting further immediate requests and ultimately by preempting some immediate reservations. We have shown that the risk of preemption can be kept very low.

Thus, agents can set up packet classifiers and schedulers in their routers, allowing routers to get on with their main task, packet forwarding.

Keywords: *Advance Reservations, Reservation Agents, Quality of Service*

2.1 Introduction

Real-time applications such as audio and video conferencing may need resource reservations in the network to perform well. However, many current resource reservation proposals allow only immediate reservations, i.e., reservations at the time the session begins, while real-time events often are scheduled and advertized in advance. Support for advance reservations allows both the network and its users to more effectively

⁰This work was supported by a grant from the Center for Distance Spanning Technology (CDT), Luleå, Sweden

schedule resources when the time of the future event needing reservations is known in advance.

Current protocols designed for resource reservations such as RSVP [80] and the ATM signaling protocol Q.2931 [63], and its Internet analog ST-2 [20] are designed for immediate reservations. In [19], we explain how to extend the soft-state approach in RSVP for support of advance reservations. However, using these protocols for advance reservations would be difficult since senders and receivers would have to be in place and capable of making reservations and large amounts of state would have to be stored in the network for potentially long periods of time.

In this paper we present an architecture where advance reservations are provided by reservation agents. For each routing domain in the network, there will be a reservation agent responsible for admission control on behalf of the routers in the domain (figure 2.1). Users can make reservation requests by contacting any reservation agent. Reservation requests are forwarded to the agent responsible for the routing domain concerned by the request.

Requests involving several routing domains are forwarded for admission control to the agents along the path. Agents can avoid forwarding each request along the path by making long-term bulk reservations with other agents so that admission requests over several routing domains can be granted locally. Agents establish reservation state through reliable handshaking.

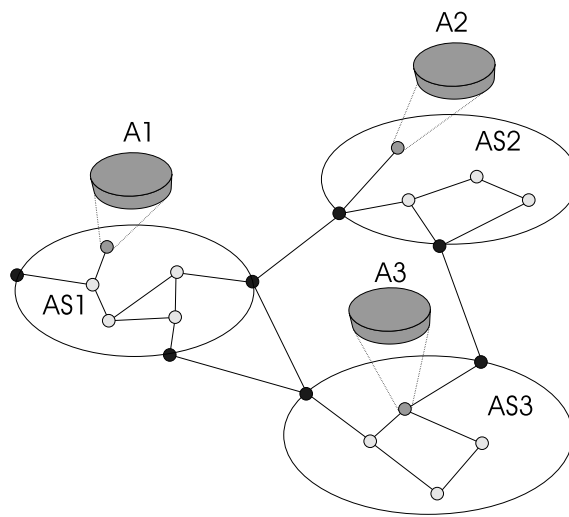


Figure 2.1: Routing domains (AS1 - AS3) and reservation agents (A1 - A3)

When the time to make resources available approaches, reservation agents reserve resources in routers by setting up forwarding state along the path. A routers primary task is packet forwarding, i.e., packet classification and scheduling. Agents

are responsible for setting up appropriate forwarding state in the routers when it is needed.

An advantage with an agent-based approach is that advance reservations can be provided with little support from routers and without having to maintain reservation state in routers through periods when reservations have no effect on packet forwarding. Furthermore, the agent-driven approach does not require reserving endpoints to be present until the session for which resources have been reserved is starting. Reservations can also be made for remote locations, thereby supporting nomadic computing, i.e., reservations for places where the host will be moving in the future. Since reservations are maintained by agents, reservation requests can be made by a third party, who may make reservations for several endpoints. Thus, neither senders nor receivers need to be in place when resource reservations are made.

2.2 Overview of the architecture

The Internet has relied on a more-or-less stateless architecture for providing best-effort service. Soft-state solutions such as RSVP have been proposed to provide immediate reservations in an otherwise stateless network. However, advance reservations do not fit into this model because of the amount of overhead that can be generated over long periods of time.

The basic principle of soft-state is that state is refreshed periodically by endpoints either by sending explicit state setup messages or implicitly by sending data packets. This approach works for state that is deployed only during periods when it is actually used for packet forwarding. Many policies such as immediate resource reservations have this property. The signaling overhead for maintaining state is marginal, especially if policies don't change very frequently. The refresh frequency can be low as route changes will be detected locally and trigger immediate forwarding of messages for setting up state over new paths. For *immediate* reservations this scheme is efficient and flexible.

However, since we expect the book-ahead time for advance reservations to be longer than the actual duration of the session, the relative signaling overhead for maintaining advance reservation state may become unacceptable. We propose that *agents*, deployed in end systems, maintain state about reservations that will be needed for *future* packet forwarding and avoid deploying state in routers until there are packets in the network whose forwarding depends on the reservation.

A reservation agent is a host that is responsible for reserving resources on a part of the Internet. Typically there is one reservation agent responsible for each autonomous system (AS) in the Internet (an AS is an administrative domain containing a number of routers running a routing protocol internally). Such reservation agents advertise themselves periodically on a well known multicast address, giving their identity and which routing domain (AS) they are responsible for. By listening to this well known multicast address, each top level agent will know the existence of all other top level agents. Inside an AS there may be backup agents as well as a hierarchy of agents,

e.g., different agents for different areas of the AS. Most agents are internal, i.e., not visible outside the AS, but if there is an agent failure, one of the backup agents is advertized instead.

Each reservation agent must know the topology of its routing domain to perform admission control on behalf of that domain, and also to detect topology changes that require re-computation of forwarding state setup in routers, or renegotiation with adjacent reservation agents. This topology information can be obtained through a link state routing protocol that provides each node, including the agent, with a topology database. It is in the nature of hierarchical routing that an agent cannot have knowledge about the internals of remote autonomous systems and therefore can only provide admission control for its own routing domain. Thus, we need at least one agent for each routing domain.

For agent-based resource reservations, static link performance measures such as link capacity must be known. In the present version of OSPF [55], there are link metrics for different type of services to allow for type-of-service (tos) routing, but there is no information on link capacity. In QOSPF [81], both static link resources and dynamic available resources are advertized. In our scheme, only static link performance is necessary. This information *could* be included in link state advertisements, which would require an addition to the OSPF specification. Instead we suggest using SNMP for finding out specific properties of links. This solution does not require any changes to the link-state routing protocol, and allows agents to monitor any information made available through SNMP.

To avoid undue signaling overhead and scalability problems, agents do not measure traffic dynamics. Admission control in agents is parameter-based, i.e., considering static link resources and traffic descriptions given in admission requests. Agents may setup resources along shortest paths or perform QoS routing based on the parameters in the admission requests. QoS routes are installed in the forwarding state of the routers.

The architectural goal is to relieve the routers from admission control and other QoS related computing as far as possible. Routers must have an interface so that agents can set up packet forwarding directly. This can be accomplished through SNMP and a special management information base.

For reservation agents to find out which autonomous systems are on the path between source and destination, agents need access to routing information that resides in the border routers of their autonomous system. This can be accomplished either by getting routing information from a border router through SNMP, or by having border routers inject external routing information into their AS. The former may be suitable for stub ASes where no external routing information is injected, while the latter may be suitable for ASes with more than one border router. External routing information together with agent advertisements can be used for forwarding admission requests to other agents.

2.2.1 Mixing advance and immediate admission control

Agents also support immediate reservations. Network resources are available for both advance and immediate reservations. Advance requests specify a duration of the requested service while immediate requests do not specify durations. In case of conflicts due to unknown durations, immediate requests may be preempted.

In a related paper [68], we present simulation results on sharing network resources between advance and immediate admission control when the admission control algorithms are loosely coupled. Advance admission control is performed without considering current immediate reservations. Immediate admission control can be performed without considering advance reservations, but the accuracy will be increased by considering resources to be allocated for advance reservations in the near future. Resources are made available for advance reservations by blocking immediate resource requests, or ultimately by means of preempting service for flows that made immediate reservations. In [68], we present results on resource utilization, rejection probability and preemption probability when resources are shared according to this model.

The results show that admission control agents can handle advance and immediate parameter-based admission control, or alternatively coexist with a separate mechanism for immediate admission control, e.g., the IETF int-serv model where immediate admission control (guaranteed service or controlled load service) is performed in routers.

2.3 Advance reservations over several domains

Agents can interoperate to support end-to-end advance reservation requests in the global network. A host requesting a reservation may consult any reservation agent, e.g., the local one or the favorite agent, just as one may use different travel agents for booking airline tickets. Some agents may already have records for managing accounting etc.

2.3.1 Unicast

A typical unicast advance request includes a duration, a traffic specification, a source, and a destination. There is no notion of sender or receiver associated with the node making the request. The source and destination descriptions may be the addresses of a sending host and a receiving host respectively. However, in many cases we don't know the addresses of endpoints at the time of making the reservation, but we may know the local area network where addresses temporarily acquired during a session. This may be the case for mobile or stationary computers that acquire addresses through a dynamic host configuration protocol (DHCP), or for computers that will be setup for a specific event, e.g., for an IETF conference. An address mask associated with each endpoint address will solve this problem. Address masks also provide a mechanism

for making aggregate reservations, i.e., bulk reservations between different domains of the network.

2.3.2 Multicast

A multicast advance reservation consists of a duration, a multicast address and a list of senders and receivers respectively. Each sender and receiver is specified by an address and an address mask. A multicast reservation tree is identified by its multicast destination address and can be extended by making further reservations for the same multicast address. Overlapping requests, i.e., requests having links in common, is merged by the reservation agent forming one shared multicast tree.

Rendezvous agents

To support advance reservations with little overhead we have a *rendezvous* agent associated with each multicast address. The rendezvous agent has group membership information, i.e., the complete list of senders and receivers. When making advance multicast reservations, the rendezvous agent provides an updated list of participants so that multicast reservations can be made, starting from the agents responsible for receiver domains. Building the tree starting at receivers is more efficient as sessions often involve fewer senders than receivers, and adding a receiver may result in reservations over a few additional links for joining an already established multicast reservation tree.

Some current multicast routing protocols, e.g., sparse mode Protocol Independent Multicast [18] and Core Based Trees [3], use rendezvous points. They are designed for setting up multicast routing trees immediately, i.e., when receivers are in place and communicating. In PIM a rendezvous point only keeps track of senders of the group and each receiver (local router) is responsible for joining with the rendezvous point to obtain a branch of the multicast tree. PIM also allows the multicast tree to be rebuilt on demand from individual receivers (local routers) to use shortest paths. To support new receivers joining, the rendezvous point is always a member of the multicast tree (thus receiving data from all senders).

Our notion of rendezvous agents is different. First to support reservations for not yet present receivers, a rendezvous agent must record all members of the group, i.e., both senders and receivers. Second, the rendezvous agent is only used as a database for finding group membership. There is no obligation that sender data pass the rendezvous agent. Group membership information is used to build multicast trees that will finally be used so that resource reservations can be made on the right branches. It does not make sense to change the distribution tree on demand when there are advance reservations, unless there are sudden changes in topology making it impossible to meet committed service along the chosen paths. The decision about whether to build a shortest-path tree or a core-based tree can be taken by the agents. We leave this issue open for now.

Hierarchical agents

To support scaling, we use a hierarchical approach, i.e., agents register border points of their routing domains rather than specific endpoints. The internal multicast tree and the local members of a routing domain are concealed by the reservation agent. Adding another member to a session where members are sharing resources can be done internally. Inside an AS there is a hierarchy of routing domains and there will be a corresponding hierarchy of reservation agents that aggregate reservations and register with the next highest level. At the top level there are several ASes and corresponding peer agents. These agents should only register involved border routers of their autonomous systems with the rendezvous agent. The task of global rendezvous agents is to maintain the list of member ASes of their multicast groups. Hierarchical approaches have been suggested before, in [51] a hierarchical approach for organizing PIM rendezvous points is suggested.

The scalability problem could be approached by getting rid of rendezvous agents completely. In dense-mode schemes for building immediate multicast trees a flood and prune approach is often used [17]. However, for advance reservations there is no data available until the session starts and therefore the relative cost of flooding control information between agents and keeping prune state during the booking time would be high.

A related issue is how to find the rendezvous agent for a given multicast address. With a hierarchical approach, all agents know about all other agents at the same level (by multicasting), so the problem boils down to partitioning the address space between agents. We propose an approach based on dynamic prefix partitioning of multicast addresses. The range of multicast addresses can be advertised as an address prefix associated with each agent. Agents then need a distributed algorithm to partition a multicast address space between themselves. The solution requires that a range of consecutive multicast addresses is allocated for the agent architecture.

2.4 Packet classification

Deciding which packets that are to use against a particular reservation is known as *packet classification*. This is performed by the routers for each packet and therefore poses a scaling problem. There are basically two ways of doing packet classification.

First, a *filter specification* could be given by clients having reserved resources. Packet classification is performed on the headers of the packets. In RSVP filtering is performed primarily on addresses and destination and source ports. Second, a reservation key (a certificate) could be included in each packet that is to use against a reservation. Packet classification is done on this single field of the header. Depending on how many different service classes we support, the field could be one or many bits long.

The problem with filter specifications is efficiency. In the worst case of filtering on sender addresses, the routers need a filter for each sender to a multicast group.

Multicast routing does not scale well even without packet classification, i.e., for best effort service. Current proposals for multicast routing scales at best as $O(G)$, where G is the number of multicast groups that are served by a router [3]. Thus, the size of the multicast routing table is proportional to the number of multicast addresses that are currently served by the router.

Our goal is to add multicast resource reservations while preserving the $O(G)$ scaling factor also for packet classification. This rules out making classification on specific source addresses. Instead we suggest that the rendezvous agent issue a key for the multicast address to other trusted agents, which in their turn may install the key in routers. The key may be added to packets by clients having reservations for sending to that group, or for better security, by the most local router under authority of a reservation agent. In the latter case only the local router must do classification and policing on packet headers to decide which packets that should be annotated with reservation keys. In IPv6, keys could be included in the hop-by-hop options header. Since agents can compute one aggregate reservation for each hop of the multicast tree, routers can make classification using only one key for each reservation.

This scheme provides scalable support for aggregate multicast reservations, i.e., where a set of senders share reserved resources. The amount of state needed for packet classification in routers is in $O(G)$, which is comparable with best-effort multicast routing. The RSVP model supports a stronger but unscalable model where it is possible to reserve resources in routers for individual senders (known as fixed filters) or for a set of senders (known as shared explicit filter). One motive for having this support is for receivers with insufficient network resources to be able to allocate resources only for the senders that it is interested in. However, in a typical multicast session the set of interesting senders changes dynamically, e.g., with the current speaker of a meeting, so a receiver with insufficient network resources would have to make a lot of re-reservations during the session. Another motive is for misbehaving senders not to be able to steal bandwidth from reservations, i.e., some policing is done by packet classification.

In our model, applications must make sure to use multicast addresses so that independently of the number of sources there is a limit on the bandwidth that is used by the aggregate multicast flow. Audio and video servers, usually have a known bandwidth limit, e.g., they provide flows with different bandwidths that can be selected by receivers depending on the capacity of their access link. It is important to know which maximum bandwidth that is used by a flow before connecting. In multiparty audio sessions bandwidth is limited by senders taking turns speaking. Video applications can be designed in a similar way, e.g., so that only the most recent speakers and the upcoming speakers on the speakers list send motion video. We believe that it is a bad idea to introduce nonscalable network solutions in order to support applications that waste bandwidth.

The advantage of providing shared reservations only is that packet classification can be done using only one classification entry for each flow (multicast address). This together with the possibility for making bulk reservations between subdomains of

the network by using address prefixes to specify leaves of the reservation tree should provide a scalable mechanism for quality of service.

2.4.1 Classification granularity

Agents may chose internally how many different service levels to support. The low end extreme is to have only one admission controlled service level shared between all flows. Then, packets only need to carry one bit to tell whether it is to use the primary service. Routers need only one queue for the primary service and another queue for best effort service. Several service levels can be supported by having a fixed number of queues in routers. The high end extreme would be to allow a queue for each flow, e.g., flow-wise weighted fair queuing. The scaling for this may be acceptable for multicast, as multicast routing already keeps per-flow state. For unicast, however, we suggest using a small number of admission controlled service levels to avoid per-flow state in the routers. If QoS routing is to be performed we may add service classes to distinguish between outgoing interfaces.

2.5 Security and policing

In our architecture, packet classification is performed by including a reservation key in each packet that is using against a reservation. Consequently, malicious senders may take currently used keys and steal bandwidth from resource reservations. However, the agents know which senders are entitled to use reserved resources and occasional policing can be done to catch violating senders. Receivers of a multicast group may even hint to their provider when policing of a group should be done. The multicast reservation scheme should not include policing for unwanted senders at every hop as default. Instead, agents should have a way to setup routers close to the edges with specific policing tasks. When there is a malicious sender identified, an agent should be able to tell the most local trusted router of the malicious sender to exclude the flow. Thus, per source filtering and policing is introduced at specific points close to the edges of the network when necessary. Per-source policing does not scale well and therefore local routers cannot be expected to have sufficient capacity for policing continuously for every local sender. In the general case, policing should be soft in the sense that only a few flows are checked at each time. The policing policy is driven by the reservation agents. An important issue is to develop policing and protection mechanisms that are fast and scalable enough to find and exclude violating flows accurately. The requirements depend on how common deliberate stealing of reserved resources will be.

2.6 Setting up forwarding state in routers

Resource set-up messages are distributed by agents to routers just before the resources are allocated. We use a soft state setup protocol where agents contact and refresh each involved router in their domain periodically. Such a protocol can be optimized by using multicast in cases where there are many routers that are to be updated with the same message. Since soft state is refreshed periodically, we do not need a reliable multicast protocol. Instead the refresh frequency can start high as new state is deployed and then gradually decrease as there is no change in state.

If the overhead for soft state turns out to be too high, a hard state protocol could be used instead. Agents could open a TCP connection (unicast) to each of their routers and setup state reliably. With this approach the routers must be able to clear up reservation state in case a reservation agent should go down and forget about the state.

To allow reservation agents to query and set router state, the routers must expose an interface. For an initial implementation the most straight forward approach is to use SNMP, i.e., having agents setting router state by sending unicast SNMP messages to each of them. The main problem is to make necessary router state available through the management information base (MIB). The authentication problem will be solved by SNMP mechanisms.

2.7 Routing performed by agents

Agents can choose to follow best effort routes and establish forwarding state for reserved resources along the current path. However, sometimes routes change (or flap) although the old route is still available and has sufficient resources for the committed service.

We suggest that agents be responsible for selecting routes and that these routes can be included in the forwarding state set in routers by agents. This also makes quality of service (QoS) routing possible. For advance reservations there is time for calculating QoS routes. When allocating resources in routers, the agents provide the address/interface of the next hop as well. Thus, routing information is included in the packet classification state, so that flows using resources reserved through agents can take a different route than the best effort packets to the same destination.

It is the job of the agent to re-route at link or router failure. If there is a link failure, routers next to the failed link will detect the problem first, e.g., through the OSPF HELLO messages. This will trigger a link state advertisement to other nodes in the AS, including the reservation agent. As the reservation agent finds that a link is broken, it will process the reservations on that link by recomputing routes, checking if there is capacity left for advance reservations through alternative routes. This will work if the capacity is available and flows haven't yet started. If capacity on alternative routes is insufficient, the agent can just wait and retry, or choose to inform users who have made advance reservations about possible problems. If there

are reservations for currently active flows on the broken link and there is capacity in an alternative path, the agent will immediately allocate resources on the alternative path. The old reservation state in the routers can be torn down by the agent or simply left to time out by not being refreshed.

Thus, when there are failures, routes will change only if there is an alternative path that can provide resources, otherwise the traffic is served best effort. When routes change the agents take multi hop decisions over the involved reservations, so the scheme does not suffer from having an arbitrary outcome in terms of who gets what on which hops. When links between different administrative domains fail, several reservation agents need to react.

As rerouting doesn't happen instantaneously, the flow state may direct packets for a short time to an interfaces that is unavailable. When this happens the packets should be sent through best effort routing instead. That includes removing any reservation keys in the packet so that best-effort service is used all the way to the destination. This works fine for unicast, but for multicast there may not be any best effort multicast routing state available and then the packets will be dropped.

If all routers along the path support the agent architecture, all routes are loop-free. However, routing loops may occur when there are routers on the path that only support best-effort routing. To avoid this, the flow state routing must check so that packets arrive on the correct incoming interface. Should a packet arrive on an incoming interface different from the one negotiated during QoS route setup, it may be a packet that coming into the QoS path again after a loop.

2.8 Comparison with the RSVP policy architecture

The RSVP policy architecture of [40] is an extension to the RSVP architecture [80]. Reservation requests goes through both resource based admission control at routers and through policy based admission control managed by local policy modules or by policy servers. Policies are bilateral agreements between neighbor providers and policy admission control is enforced at border nodes only. Reservation requests, carried via the normal RSVP soft state mechanisms, contain policy data objects that are passed to local policy modules at border nodes. Policy data objects may be added, removed, or rewritten by policy modules along the path. Routers have the choice to handle policies in their local policy module or to outsource policy decisions to a policy server. There is a protocol for communication between routers and policy servers using hard or soft state.

An important difference between our approach and the RSVP policy approach is the role of routers. In our architecture, agents handshake to make reservations and to compute the forwarding state for routers, while the routers primary task is to do packet forwarding, i.e., best effort service and service disciplines installed by the agents. Admission control and QoS routing decisions are done by agents. Our goal has been to relieve the routers from computation concerned with QoS contracts and put it in agents instead. Thus admission control and service commitments are

all managed by agents. In RSVP, routers play a central role in admission control and routing. Routing decisions are not part of RSVP but instead made by the best effort routing mechanisms in the routers. Consequently the RSVP mechanism can easily reestablish reservations along changing routes. However, when routes change (or flap) the outcome in terms of who gets reservations depend on the order reservations arrive on the new path. An annoying problem with RSVP is that users do not know if a reservation is granted end-to-end. Non-RSVP routers may be on the path and may disturb end-to-end quality. We believe that RSVP is suitable for providing loose service commitments but it cannot support guaranteed service very well. Our architecture may adapt more slowly to changing topology but it provides a better vehicle for guaranteed quality of service and certainly for advance reservations (which it was designed for from the very beginning). In RSVP senders must be present to make reservations, which might not always be possible far in advance.

Another problem with RSVP is poor support for aggregate reservations. End-points make reservations on a per-flow basis and although reservation messages are merged up the multicast tree, we still have reservations per flow. In addition, packet filtering is designed so there may be classification state for each sender. This is clearly not scalable. In our agent architecture aggregation of reservations is included from the beginning. Classification is based on keys (certificates) and the leaves of the multicast reservation tree may be routing domains (address prefixes). There is work in RSVP on introducing address prefix aggregation (aka CIDR prefixes), but alone this is not enough to handle sparsely located senders to the same multicast group. Packet filtering/classification may still have to be done on a per sender basis.

2.9 Conclusions

Advance and immediate reservations can be supported by agents responsible for admission control on parts of the Internet. Our architecture minimizes disturbing the primary task of the routers: packet forwarding. Soft state in the routers is deployed only during periods when it is needed for packet forwarding.

The agent-driven approach described in this paper does not require the communicating endpoints to be present until the session for which resources have been reserved is starting. Reservations can also be made for remote locations, thereby supporting nomadic computing, i.e., reservations for places where the host will be moving in the future. Since reservations are maintained by agents, reservation requests can be made for a group of participants by a third party.

Reservation agents have access to routing databases for finding out the topology of the routing domain that it performs admission control for. Reservation agents use SNMP to find out the static link resources available in the network and for setting up forwarding state in the routers. Agents perform parameter-based admission control, using traffic descriptions provided by clients making reservation requests. Thus, agents work without signaling any traffic dynamics.

The agent architecture is suitable for parameter-based QoS routing of advance reservations. There is intrinsically more time available for admission control and QoS routing when requests are made in advance.

Part 3

Resource Sharing in Advance Reservation Agents

Resource Sharing in Advance Reservation Agents

Olov Schelén and Stephen Pink

Computer Science and Electrical Engineering

Luleå University of Technology, SE – 971 87 Luleå, Sweden

Tel: +46 920 91000, Fax: +46 920 72801, Email: {olov, steve}@cdt.luth.se

Abstract

We present an architecture where clients can make end-to-end resource reservations through agents. For each domain in the network, there is an agent responsible for immediate and advance admission control. Reservations from different sources to the same destination domain are aggregated as their paths merge toward the destination.

We show that network resources can be shared between immediate and advance reservations without being pre-partitioned. Admission control for immediate reservations use information about resources to be allocated for advance reservations in the near future. An important parameter in the admission control algorithm is the so called lookahead time, i.e., the point at which we actually start making resources available for approaching advance reservations by rejecting immediate requests. In our model, preemption of immediate reservations is made in cases where the admission control cannot make resources available through rejection of immediate requests. The risk of preemption can be varied by changing the lookahead time. We explore, with simulations, the effects of providing advance reservations with this model. The results show the cost in terms of resource utilization, rejection probability and preemption probability.

Keywords: *QoS, advance reservations, resource sharing, agents, admission control, simulation, virtual leased lines*

3.1 Introduction

Real-time applications such as audio and video conferencing may need resource reservations in the network to perform well. However, many of the current resource reservation proposals allow only immediate reservations, i.e., reservations at the time the session begins, while real-time events often are scheduled and advertised in advance.

Clients making immediate reservations for scheduled events will have to judge the appropriate time for making reservation requests to be accepted in due time. When resources are scarce, clients may start making reservations earlier than needed to make sure that they obtain resources in due time. This leads to sub-optimal scheduling of network resources. By offering reservations in advance, clients obtain better support for planned activities and providers obtain information for scheduling resources over time.

Support for advance reservations is also important between different providers. For scalability reasons, it is not desirable that each end-to-end admission request results in call-specific state for each link along the path. Therefore, providers will negotiate bulk resources among each other for carrying aggregated admission-controlled traffic over several domains. In this context, advance reservations are suitable both for providing time-limited contracts and for providing bandwidth that varies at given time intervals, e.g., over hours of the day and days of the week.

We have designed a reservation architecture where immediate and advance admission control is performed by *agents* [67, 69]. The provided service corresponds to *virtual leased lines* between domains. An advantage with an agent-based approach is that reservations can be provided without having to maintain reservation state in routers. Reservations can be made for both senders and receivers. In addition, reservations can be made for remote locations, thereby supporting third party reservations and nomadic computing, i.e., reservations for places where the host will be moving in the future. Our architecture can aggregate admission requests along the path towards a destination. This results in admission control for the aggregate being performed for individual links. In section 3.2, we give the background on our agent architecture.

The focus of this paper is on resource management for individual links. In section 3.3, we describe admission control for immediate and advance reservations [68]. The objective is to share resources between immediate and advance reservations. In section 3.3.1, we describe some architectural considerations about locating admission control in agents. In section 3.3.2, our service model is described. In this model resources for advance reservations will be made available by *rejecting* immediate requests for a short period before resources are to be allocated, or ultimately by *preempting* service for flows that were granted immediate access. Preemption means that a flow of packets using the reservation loses its service quality, i.e., the flow will still be given best-effort service (preemption is defined in section 3.3.2). In section 3.3.3, we define the notions *bookahead* and *lookahead*. In section 3.3.4, we define the admission control criteria. In section 3.4 we show, with simulations, the effects of advance reservations in terms of resource utilization, rejection probability and preemption probability.

3.2 The agent architecture

In [69] and [67], we present an architecture where clients can make resource reservations through *agents*. For each domain in the network there is an agent responsible

for *admission control* (figure 3.1). Each agent knows the topology and static link resources in its domain. The agent is an end-system that is configured for passively participating in a link state routing protocol (e.g., OSPF) where each participating router has an identical topological database over the domain. Thus, an agent obtains the topological database with little signaling overhead. Link state advertisements are sent from the nearest router to the agent. Agents retrieve link properties, such as static bandwidths, by querying routers seen in the topology map. For this, we use a network management protocol (e.g., SNMP). Queries are done at startup and when topology changes are detected by the routing protocol. Routers need no extensions to allow agents to build a resource map through the management protocol.

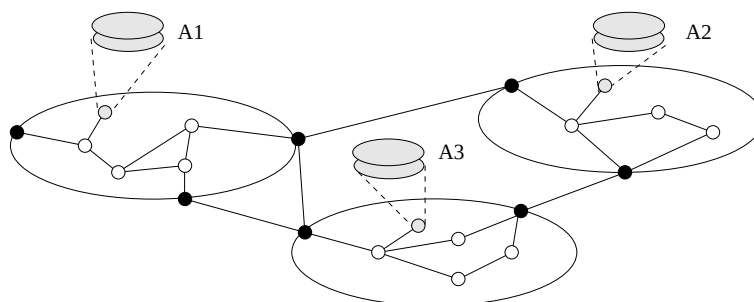


Figure 3.1: Reservation agents and their domains

Packets using reserved resources are *marked* by applications or edge routers. Core routers handle marked packets by classifying them into a small number of predefined service types. Similar ideas have been proposed in the *differentiated services* area [14, 57]. In the differentiated services model, packets are marked to obtain either lower drop probability or higher strict priority. Here, we abstract from these details and use the term *priority packets* to denote marked packets, and the term *priority traffic* for all priority packets collectively. A key point of differentiated services and our agent architecture is to avoid signaling state and expensive per-packet processing in routers. To allow resources to be managed in separate agents, we rely on router mechanisms for service differentiation on the packet level.

Independently of which packet scheduling model is used for differentiating service in routers, there must be an admission control architecture to make sure that the amount of priority traffic is low enough to receive good quality of service. One issue is whether the admission control architecture should reserve resources over the links that are actually going to be used, or just limit the overall rights to send priority traffic. A scheme that does not consider exactly where traffic will go cannot give reliable commitments without very low utilization. Thus, it is likely that such a scheme would need to gamble and accept occasional service loss.

In this work we focus on agent-based admission control where the objective is to administer resources exactly, i.e., admission control in an agent maintains information about reserved resources on each link in its domain. Agents perform immediate or advance admission control without involving the routers. The architecture offers a

service comparable to unidirectional *virtual leased lines*. Reserved resources can be used by one or many applications sending data between the domains connected by a virtual leased line.

There are other frameworks, i.e., IETF Integrated services [78, 36] and RSVP [80], that can set up both unicast and multicast reservations end-to-end. However, the provided service is not the same as for virtual leased lines, since reservations are made for individual data flows (i.e., application data streams). In that model, per-flow reservations are maintained in the routers. This results in scaling problems due to reservation state and packet classification overhead in routers. This model inherently relies on per-flow semantics and therefore it is hard to perform aggregation over different administrative domains. Thus, RSVP and Integrated Services may be too heavy-weight for the backbones.

In differentiated service, on the other hand, there is low overhead in routers. Our goal is to design an architecture that performs scalable resource management. Currently, our service model provides only unicast virtual leased links. When resources are managed by agents according to this service model, there are opportunities to aggregate reservations as they merge toward a destination domain. In this section, we give an overview of the admission control architecture and focus on the aggregation issues.

3.2.1 Direction of admission requests

An admission request can be directed to any agent, e.g., an agent that manages a user's account. Each agent sets up reservations between any two points in the network by invoking other agents. Thus there is *no difference* between reservations for *sending* and *receiving* data, or reservations made by a *third-party*.

Each reservation request contains a bandwidth to be reserved, a source and a destination address. The source address determines the point where the reservation starts (i.e, the point where traffic using the reservation will enter) and the destination address determines the point where it ends. An agent receiving a request first considers whether the starting point is in its domain. If the starting point is *not* in its domain, it finds an agent closer to the starting point and repeats the request with that agent. Technically speaking, it is possible to directly identify the agent that is responsible for the source domain, but in practice clients may prefer using a particular agent. Also, agents may prefer having business relations with only a few adjacent agents. This would require that requests are made in several steps. The result is that accounting information is aggregated, i.e., the mechanism helps keep down the number of separate charges. In figure 3.2, it is shown how a request is issued by node Dx for reserving resources from Ax to Dx.

If an agent finds that the starting point given in the request is in its domain, admission control is performed on the links from the starting point towards the destination, defined by the routing protocol. This is possible since all routers and the agent

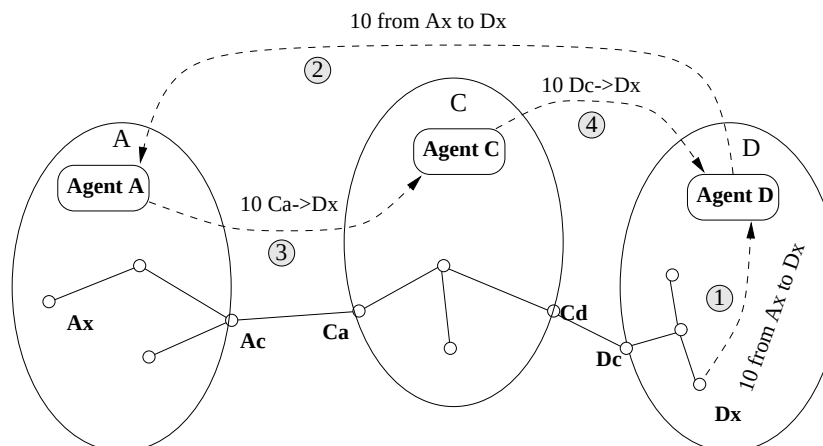


Figure 3.2: A receiver Dx requesting a reservation to receive data from Ax

have the same topology map of the domain. Thus, an agent can perform admission control on the links from the starting point to an edge router in its domain.

An agent cannot perform admission control beyond its domain. Therefore, if the destination is outside of the agent's domain it must request a reservation with the neighboring agent, giving the edge point where traffic will cross the borders as the source address of the reservation (figure 3.2). Without distinguishing between reservations to different destinations, it would be impossible for agents to grant reservations to other domains. This is because there is no way to make sure that there are sufficient reservations along any path in other domains further downstream (unless we accept very low utilization for priority traffic).

3.2.2 Aggregation

If reservations were always done “on demand” there would be signaling between all neighboring agents along the path for each end-to-end admission request. Our architecture supports several ways of aggregating reservation state. Aggregation takes place in the agents because routers are not part of the admission control process. The motivation for aggregation may be less when reservation state is stored in agents with plenty of memory and disk. However, a per-flow reservation model would have scaling problems both in terms of state and processing cost in agents. Therefore it is important to show that agents can provide exact resource reservations with good scaling properties.

In [69] we describe how aggregation is done. The key ideas are the following: first, reservations from different sources to the same destination are aggregated as their paths merge toward the destination. Second, an agent in charge of resources at the

final destination can generalize reservations for specific end points so that they are valid for any end point in the destination domain, thereby allowing more aggregation.

Merging reservations

To grant requests spanning many routing domains, there must be a commitment from all agents along the path involved in the request. When an agent is making a reservation with a neighboring agent, the source address included in the request determines the point (router interface) where the traffic will cross the borders, i.e., enter the domain of the neighboring agent. The neighboring agent does not care where traffic came from originally. Thus, priority packets can use reserved resources as long as the destination address matches the reservation. This allows agents to aggregate traffic with different sources into one single reservation made with a neighboring agent.

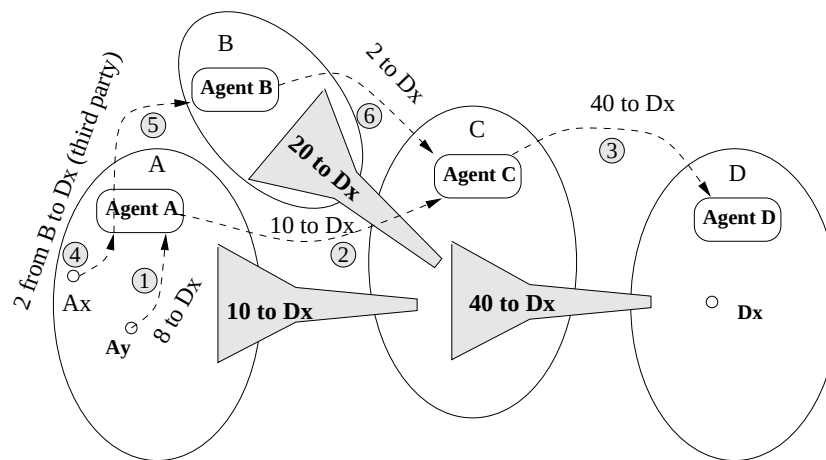


Figure 3.3: Funnel and aggregate reservations for one destination

An end-to-end resource reservation can be seen as a number of consecutive *funnels* (figure 3.3). Priority packets poured into any of these consecutive funnels use reserved resources along the path to the destination. Agents may aggregate several requests with different source addresses into the same funnel, as long as all the requests specify the same destination. Agent C (in fig. 3.3) sets up a funnel starting at its edge by reserving bulk resources (40 units) with neighboring agent D to a distant destination and then aggregate requests for resources between domains C and D into the established funnel. This bulk reservation implies that downstream agents only keep state for the aggregate. The agent that decides to aggregate into a bulk reservation keeps information about its individual commitments constituting the aggregate. Aggregation by merging reservations towards each destination can be done with full control over the resources. The funnels form a so called *sink-tree* with the root towards the destination. At each merging point, the reserved bandwidth on the outgoing link will be equal to, or larger than, the sum of the reserved bandwidth on the incoming links.

Prefix aggregation

Requests that specify different destinations cannot easily be aggregated into a funnel in the general case. If we allow funnels to split into different directions further downstream, upstream agents must keep information about which destinations are involved and how resources are divided between them. This means that the aggregate no longer can be seen as one unit. However, if a set of destinations can be identified by one common identifier, and there are sufficient resources available in each branch to service the whole aggregate, then split outs can be scalable and exact.

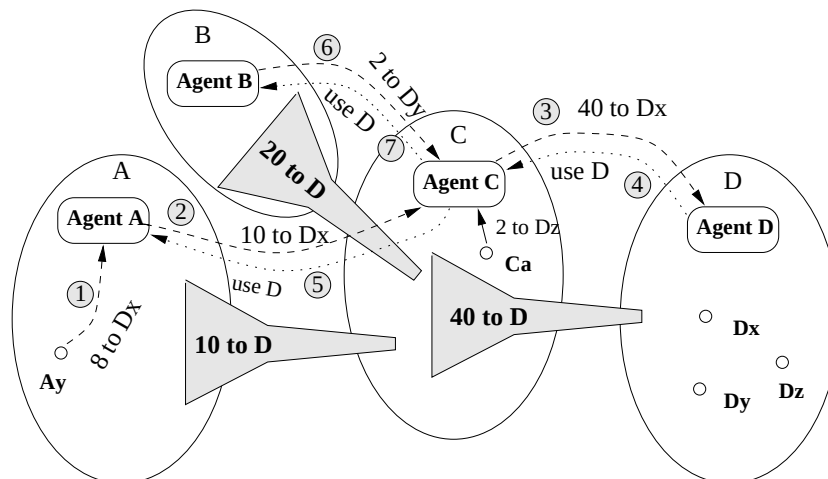


Figure 3.4: Funnels and generalized destinations

With this argument, we allow aggregation to be increased by *generalizing* a reservation as valid for all destinations within the same destination domain. This means that a funnel provides resources to all endpoints within the destination domain, i.e., resources in the destination domain must be sufficient to ensure that priority packets get expected service independently of how they are routed within that domain. It is only the agent responsible for the destination domain that can judge if resources are sufficient for allowing generalization. Consider the example in figure 3.4. Agent D can judge if the resources in D are sufficient to let incoming funnels be valid for the whole domain. Thus, it must be agent D that decides when a reservation request for a particular destination node within D can be replaced with a reservation for any node in domain D.

In IP networks an *address prefix* is often used to represent a domain (individual nodes within the domain are distinguished by the remaining suffix). An address prefix is represented by an IP address and an associated mask telling how many bits are part of that prefix. An agent generalizes a request by replying with an *address prefix* to be used as a destination address for the reservation instead of the original address

included in the request (figure 3.4). The aggregation that results from generalizing a reservation is also called *prefix aggregation*.

Clients or agents that have been granted a generalized reservation can use it for any destination matching the prefix of the reservation. Agents providing generalized reservations must handle the committed bandwidth of priority traffic to any destination in its domain. A conservative agent would therefore generalize reservations up to an aggregated value equal to the narrowest link. In practice we believe that this can be useful for well provisioned local area networks.

In the Internet, agents must be able to handle occasional topology changes (e.g., node and link failures), as well as pathological changes (e.g., route-flaps) [61]. Agents listen to the routing protocol and detect routing changes on average as fast as any router in the domain. When changes occur, agents can schedule resources along new paths.

3.3 Admission control for immediate and advance reservations

To obtain a modular design, advance reservations should, as far as possible, be managed separately from immediate reservations. One way of separating the admission control for immediate and advance reservations would be to partition the resources so that one part is available for immediate reservations and another part for advance reservations. However, this would result in two logically separate networks with the problem of partitioning resources according to current demand. Therefore, we have chosen to share resources between immediate and advance admission control. To allocate resources safely we have extended the immediate admission control to consider the near future concerning the resources that will be allocated for advance reservations. There are three major consequences of these results. First, the resources can be managed efficiently without suffering from badly chosen resource partition sizes. Second, the immediate admission control scheme can be kept fast and simple by not having to consider advance reservations in the far future. Third, we have the architectural freedom of locating the admission control for immediate and advance reservations at separate places in the network without suffering undue communication overhead.

3.3.1 Parameter-based or measurement-based?

Admission control algorithms can be classified as either parameter-based or measurement-based. The metrics used in *parameter-based* admission control are *traffic specifications* and static link resources. By aggregating the given metrics, such a scheme can offer absolute or statistical performance bounds [30, 36], often enforced by the packet scheduler (e.g., WFQ or EDF). Parameter-based admission control can be used for immediate reservations and, by adding timing information, for advance reservations.

The metrics used in *measurement-based* admission control involve also *measurements* of used resources [45] [78]. Such a scheme can detect reserved resources that are not utilized and grant those resources to other requests. The benefit of measurement-based admission control is better resource utilization. On the other hand there is no guarantee that reserved resources are available when needed. The scheme may also be sensitive to synchronization effects. Suppose there are many events scheduled at a given time and many clients start reserving immediate resources just before that time. As there is little traffic, more reservations will be allowed. When all events start, the network will be over-allocated with poor service as a result.

In our agent scheme, we use only parameter-based admission control for a number of reasons. First, measurement-based admission control can be efficiently implemented only in routers or switches (where traffic measurement can be done directly). When admission control is in the agent, the signaling overhead for traffic measurements may be prohibitive. Second, traffic measurements provide little information about the future and are therefore not suitable for advance reservations [19]. Third, we want to provide a service model where reservations give a trusted level of service for people that are prepared to pay for it. The problems of lower utilization for reserved traffic are mitigated in a packet-switched network as unused resources always can be used by best-effort traffic. Thus the overall network utilization does not suffer very much from parameter-based admission control.

3.3.2 Service model

To support advance reservations, information about time must be added to the reservation requests. Therefore, advance reservation requests include a *duration* interval: $I = [t_s, t_e]$, where t_s is the starting time of the requested service and t_e is the finishing time of that service. This information could be automatically collected from a site where scheduled events are advertised, or generated by the client making a reservation. This is indifferent to our architecture.

Immediate reservations are open ended and do not specify a duration. A consequence of allowing open ended immediate reservations together with advance reservations is that there may be cases where resources are over-allocated. This is resolved by *preempting* immediate reservations to free resources for advance reservations. Preemption is used to ensure that there are always resources to honor advance reservations. Flows that lose reserved resources through preemption will be given best-effort service. We have chosen *not* to include a service for immediate requests that are *guaranteed* never to be preempted. Instead, we aim at keeping the preemption probability low, so that the user need not worry about preemption when asking for admission. A fundamental reason for this decision is that indefinite non-preemptible service would block resources indefinitely and no advance reservations of those resources could therefore be made. Furthermore, resources that are reserved in advance for some time in the far future cannot be given to indefinite sessions in the meantime unless we have the option of preempting those sessions when resources are needed. Thus, we have the following service model:

- Durations must be specified for advance reservations. A duration includes the start time and the finish time for the requested service.
- Immediate reservations do not specify durations. Such requests are serviced immediately (if possible) and can be preempted later by flows that reserved in advance.
- Preemption means that a flow loses its service quality; however, it will still be serviced at a best-effort level.
- In order to manage resources safely there may be limits imposed on minimum and maximum bookahead times for advance reservations.
- The admission control for advance reservations may support finding the earliest point for which a requested service can be granted.

There are other proposals for advance reservations that do allow indefinite (open ended) resource reservations. To provide indefinite sessions without having preemption, resources could be set aside for this service exclusively. In the Tenet suite [26], resources are partitioned to support advance reservations and non-preemptible immediate sessions simultaneously. With that solution there will be some fragmentation due to it being hard to maintain the partitioning such that the demand for the two different services are met at each instant. In our architecture we can support indefinite sessions by partitioning resources. However, in this paper, we will investigate whether or not we can avoid some problems with resource fragmentation by sharing resources between immediate and advance reservations and instead allow for statistically rare cases of preemption.

There are also proposals relying on statistical guarantees of immediate and advance reservations where both immediate and advance reservations are preemptible. In [76] a model is presented where the call with the shortest bookahead time is preempted in case of congestion. The model relies on the theory of effective bandwidth to calculate the chance of a call being active. In our model we deal with aggregated bulk reservations rather than per-call reservations. For our model we have chosen to use deterministic guarantees in the admission control for advance reservations. Our belief is that advance reservations, once accepted, should not be preempted. Preemption of advance reservations is acceptable only when there are failures in the basic infrastructure. This consideration is made in a context where there is a lot of unplanned network traffic that use immediate reservations and an even larger amount of bursty traffic that uses no reservation at all. In this context resources will be well utilized.

3.3.3 Bookahead and lookahead

When immediate reservations and advance reservations share resources and the duration of each immediate reservation is unknown, there is a risk of over-allocation that can be resolved with preemption. The probability of preemption can be controlled by

either monitoring immediate requests when admitting advance reservations, or vice versa, by monitoring advance reservations when admitting immediate requests.

In practice, however, the risk of preemption for immediate requests can not be controlled very well by advance admission control. When advance reservations are requested far in advance, the information obtained from monitoring present traffic or present immediate reservations will not be useful [19]. Most of the active flows would have finished at the crucial time anyway. It is only when advance reservations are requested with very short notice that it would be valuable to know about present traffic. Since advance reservations are provided to meet the demand for scheduled sessions primarily, we believe that most reservations will be requested on a notice much longer than the average session length. We denote the time from making an advance reservation until the resources are to be available as the *bookahead time*. The bookahead time is set by the clients asking for admission and varies from request to request. In our admission control we may impose a minimum bookahead time. Then, advance admission control need not consider the immediate reservations state at all.

The risk of preemption can be effectively controlled by immediate admission control. By monitoring advance reservations for the near future when admitting immediate requests, we can control the risk of preemption. If there are insufficient resources available in the near future, the request could be rejected instead of serviced. It is up to the provider to decide whether rejection or preemption is preferable in these cases. Rejecting immediate requests because the resources are reserved in the near future means that the overall utilization will go down to make room for the advance reservations. We denote the pre-allocation time, i.e., the time for starting to set aside resources for advance reservations as the *lookahead time* into the advance reservation state. Setting the lookahead time is an internal issue to the admission control scheme. As shown in simulations (section 3.4), the lookahead time can be tuned to obtain reasonable utilization and preemption probability.

3.3.4 Admission control criteria

Figure 3.5 is a snap-shot of admitted flows in a time/bandwidth diagram. Flows a, b and c are immediate reservations. For immediate reservations we add up the aggregate bandwidth. When a new admission request α arrives, admission is granted if the new reservation would not cause the aggregate bandwidth to exceed the link bandwidth. For advance reservations, the admission conditions consider other advance reservations that overlap with the new flow (d,e,g,h). The conditions are checked at all points where new flows begin (t_s , t_x and t_y). For immediate requests, the admission conditions consider the current aggregate of immediate reservations and the advance reservations that are active within the lookahead time (d,e,f). The following notation is used in describing our admission criteria:

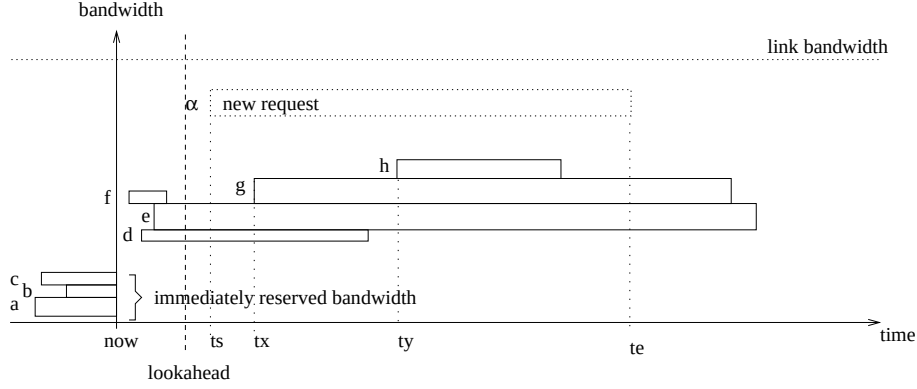


Figure 3.5: Snap-shot of reservations

- R Aggregate bandwidth reserved by current immediate reservations
- $A(t)$ Aggregate bandwidth reserved in advance at time t
- $look$ The lookahead time used at immediate admission
- μ The link bandwidth

Immediate reservations: When a client requests an immediate reservation α with bandwidth r^α , the admission control algorithm performs the following checks:

- Determine if the bandwidth usage, after adding the new load r^α , will exceed the available link bandwidth μ during the lookahead time.

$$\mu > \max_{t \in look} (r^\alpha + R + A(t)) \quad (3.1)$$

Advance reservations: When a client requests an advance reservation α with bandwidth and duration (r^α, I^α) , the admission control algorithm performs the following checks:

- Check whether the starting time is acceptable as a bookahead time. The minimum bookahead time is configurable to any value larger than the lookahead time.
- Check whether the finishing time is acceptable. It should be larger than the starting time. Admission control may also set a limit on how far in the future reservations can be granted.
- Determine whether the aggregate bandwidth of advance reservations exceeds the link bandwidth at some point during the requested duration:

$$\mu > \max_{t \in I^\alpha} (r^\alpha + A(t)) \quad (3.2)$$

3.3.5 Reservation state

As described in section 3.2.2, our model has mechanisms to minimize the amount of reservation state. Nevertheless, it is important to have data structures that manage reservations efficiently and allow quick admission control. For immediate admission control, we need to keep information about the current reserved aggregate bandwidth. We also need access to the advance reservation state within the lookahead time. For advance admission control, we need to keep the aggregated bandwidth reserved in advance over time. For this we have two alternative data structures. Our first data structure keeps exact starting times and finishing times for each admission request. The amount of state is proportional to the number of reservations maintained. The other data structure uses time slots where each reservation is extended to occupy a number of time slots. For each slot, the aggregate reserved bandwidth can be found. This data structure has fixed size and depends only on the maximum bookahead time allowed and the slot length.

Also, we must be able to find a victim for preemption among the immediate reservations. For this we need information about individual immediate reservations. However, potential victims can be chosen when they are admitted. It is enough to record a small number of potential victims corresponding to the amount of resources to preempt.

3.4 Simulations

The objective of our simulations is to explore the effects of allowing reservations in advance. The performance measures are *total utilization*, *rejection probability*, and *preemption probability*. We vary the *fraction* of the total link resources used for *advance reservations* and the *lookahead time* into advance reservation state at admission control for immediate requests. Admission decisions are parameter-based, i.e., based on traffic specifications and static link resources. For simplicity we use the requested bandwidth as the only traffic parameter. This study is for admission control over a single link only.

3.4.1 Simulation parameters

- The duration of immediate requests, i.e., the call hold time, is exponentially distributed with a mean of 300 time units.
- The duration of advance requests, is also exponentially distributed with a mean of 300 time units.
- The bookahead time for advance requests, i.e., how far in advance the advance requests are made, is exponentially distributed with a mean of 20 000 time units, where the tail is cut at 60 000 units (i.e., there is a maximum bookahead time).
- The total link capacity is 45 000 resource units.

- The resources asked for in each request are uniformly distributed in $[1..3000]$ resource units.
- The inter-arrival time of admission requests is exponentially distributed with a mean of 11.76 time units. This number was selected to obtain a reasonable rejection probability, where around 6% of the requests are rejected. The average offered load is 85% of total capacity.
- When preemption is performed, the immediate request with the largest requested capacity is picked as a victim (which would encourage making requests in advance when large capacity is needed).

Note that the same parameters for duration and resource requirements are used for both immediate and advance reservations. This is to isolate the impact of reserving in advance versus reserving immediately. Should we have different parameters for the two classes, other effects would dominate the outcome in terms of utilization, rejection probability and preemption probability (the impact of having other parameters is discussed in section 3.4.6). The bookahead time is chosen to be sufficiently large so it has little impact on the currently active immediate reservations.

The simulations are run many times for each parameter set, with different seeds for the random variables. The 90% confidential intervals are computed and presented as error bars in the plots (although the confidence is often so tight that the error bars appear as dots).

3.4.2 Advance reservation requests

We vary the fraction of the admission requests that are made in advance between zero and one. The upper bound on resources available for advance reservations equals the total link capacity, resulting in advance reservations being seldom rejected when a small fraction of the requests are made in advance (Figure 3.6).

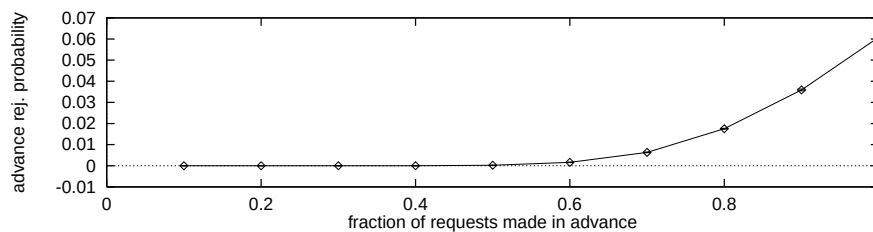


Figure 3.6: Rejection probability for advance requests

Figure 3.7 shows that the average duration of the granted requests goes down as there is a higher fraction of advance reservations. When resources are scarce, advance reservations will result in some fragmentation in time, and there is a greater

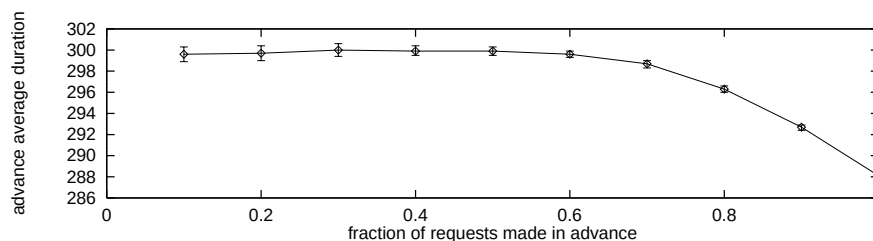


Figure 3.7: Average duration for accepted advance requests

chance that requests for short durations are accepted. This effect does not occur for immediate requests since admission is made without considering durations (in fact, durations are unknown).

A similar effect can be observed concerning the average bandwidth given to the granted requests. If the offered load is increased, the average bandwidth for granted requests will go down as the rejection probability goes up. For immediate reservations clients see a considerable decrease in the chance of succeeding with large reservations as the offered load is increased. However, for advance reservations the rejection probability depends on the bookahead time. Therefore, clients can increase their chance of succeeding with large requests by reserving further in advance. This effect is clearly seen in our simulations. However, as the bookahead time is chosen by clients it is not easy to draw any conclusions on *how* early one should make an admission request. The appropriate bookahead time depends on when other clients choose to book ahead. The only effect that can be claimed for sure is that the chance of having a request granted for a given duration increases strictly with the bookahead time.

3.4.3 Total utilization

We define the *total utilization* as being the ratio of the total reserved bandwidth versus the total bandwidth of the link. This metric considers how large part of the total resources that are reserved. The metric does not include increased link utilization due to best-effort traffic, nor drops in link utilization due to reserved resources not being utilized.

Figure 3.8 shows that the total utilization goes down as the fraction of advance reservations increases. There are two reasons for this. First, we start allocating resources ahead of the starting point of the durations for advance reservations. The interval during which resources are allocated ahead of time is denoted the *lookahead* time. Technically speaking the lookahead time is used by admission control for *immediate* requests to judge whether there are any advance reservations needing the resources within that lookahead time. If so, the immediate request will be rejected. Thus, having a longer lookahead time results in more immediate requests being rejected and thereby a lower utilization. Having a shorter lookahead time means that

the network will more often be over-allocated, resulting in more immediate requests being preempted (as we will show in section 3.4.4).

The second reason for lower utilization is that fragmentation in time is introduced when advance reservations are allowed. There is little chance that other advance requests ask for the small fragments that may be available between already-granted advance reservations. Consequently, a decrease in utilization can be observed as the rate of advance reservations is increased, even though the lookahead time is zero. When the lookahead is zero, admission control for immediate requests does not consider any resources reserved in advance.

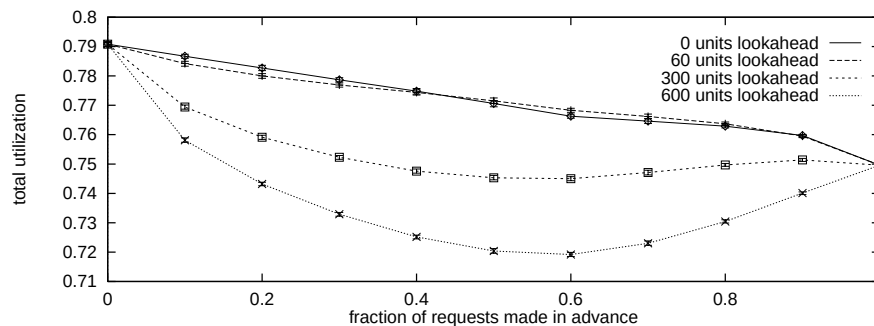


Figure 3.8: Total utilization

This may sound like bad news for providing advance reservations, but in fact similar drops in total utilization would be observed as soon as there are scheduled events introduced in the network, even though users were to make only immediate reservations. Users would still be likely to request resources before events begin to make sure that they have resources allocated in time. The only difference is that the control is with the clients instead of with the network. The degree of over-allocation in time that clients choose would be a consequence of social behavior in an environment with scarce resources. In some situations we can expect that the utilization drops in an uncontrolled way. When there are scheduled events, advance reservations can improve the service and the utilization by ensuring that resources are allocated at an appropriate time.

Measurement-based immediate admission control could alleviate the problem of clients over-allocating in time by measuring traffic on the link. However, we cannot get any useful measurements for scheduled events until the traffic starts. Thus, we must choose between staying with the traffic specification or gradually losing the resource allocation as the measurement procedure finds that there is no traffic. When there are several events scheduled at the same time, there is a risk that allocated resources are insufficient to give acceptable service.

3.4.4 Rejection and preemption probability

Figure 3.9 shows that the rejection probability for immediate reservations increases as the fraction of advance reservations is increased. However, the change in rejection probability for immediate requests is primarily an effect of picking the victims among a smaller set of immediate requests as the rate of advance reservations increases. If we instead monitor the total rejection probability (Figure 3.10) over both immediate and advance requests, we get a much lower risk of rejection (note that the scales are different). Perhaps this is a better reflection of overall user satisfaction.

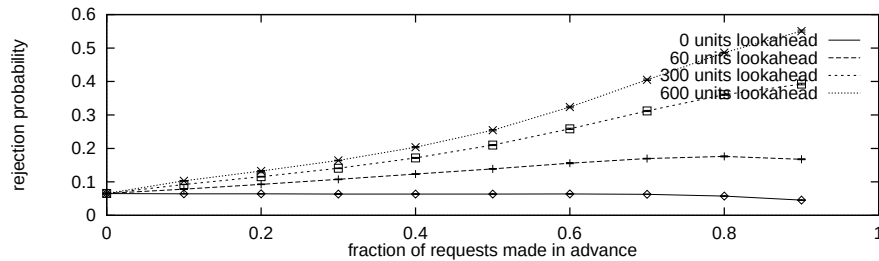


Figure 3.9: Rejection probability for immediate requests

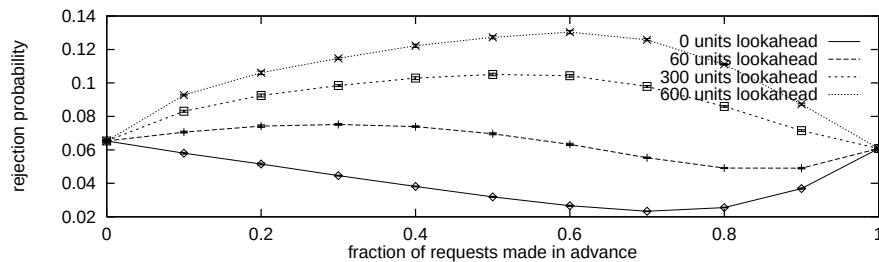


Figure 3.10: Total rejection probability (for immediate and advance requests)

We can see that the rejection probability is about the same when we have only immediate requests as it is when we have only advance requests, i.e., around 6 %. The decrease in total utilization (Figure 3.8) is explained by the fact that the average duration for accepted requests goes down (due to fragmentation in time) with increasing demand for advance reservations (Figure 3.7).

Figure 3.11 shows that the preemption probability increases as the fraction of advance reservations is increased. Again, part of the explanation is that all preemption victims are picked within the immediate requests and as the fraction of immediate requests goes down we get a higher risk of preemption for this service class. In Figure 3.12, we show the preemption probability over the total number of served reservations.

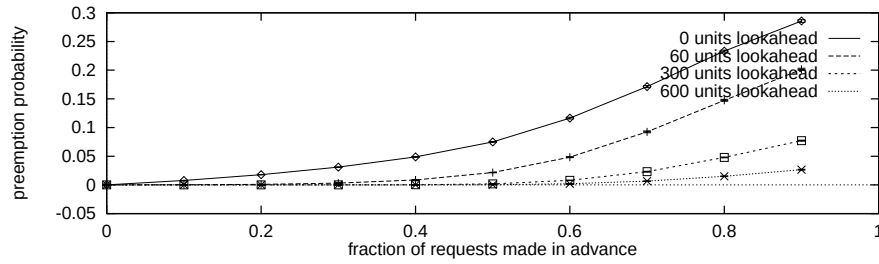


Figure 3.11: Preemption probability for immediate requests

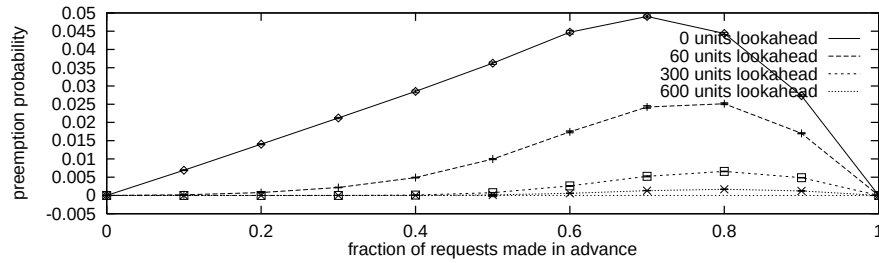


Figure 3.12: Total preemption probability (for immediate and advance requests)

It is clear that increasing the lookahead time makes admission control more conservative, resulting in lower preemption probability at the cost of lower utilization and higher rejection probability. By decreasing the lookahead time, we allow the system to be more over-booked resulting in higher preemption probability. The fact that the system is allowed to be over-booked explains why the total rejection probability (Figure 3.10) goes down when we use zero lookahead and there is a suitable fraction of advance reservations.

3.4.5 Lookahead time

Given a desirable target for preemption probability, the lookahead time can be conservatively computed by considering the bandwidth that must be allocated for each advance reservation and the amount of bandwidth that is returned per time-unit by immediate reservations ending. However, at reasonable load, it is likely that there are some unallocated resources that should be considered as well. In our simulations, between 75% and 79% of the resources are allocated on average, which explains why it is possible to allocate resources for advance reservations with a total preemption probability below 5%, even when using zero lookahead.

In previous examples we kept the lookahead times constant as the fraction of advance reservations was increased. We have also tried to use exponentially distributed lookahead times but we found a strict performance degradation compared with constant lookahead time. If there are problems in meeting the requirements for high utilization and low preemption probability, we may be forced to have a limit imposed on resources available for reservations in advance. As an example, we now fix the advance request rate to 40% of the total number of requests (i.e., the average is bounded and there is no firm bound) and we plot the performance measures as functions of the lookahead time.

Figures 3.13, 3.14, and 3.15, show the effects. It is clear that the preemption probability falls steeply as the lookahead time is increased. In most systems, like cellular phone systems, preemption is considered bad and should be avoided. In a connectionless packet switched network, preemption is not as bad since generally the packets are still served best-effort. We believe, however, that the relative weight of having low preemption is still high. Provided that we have the average duration for resource reservations (assuming exponentially distributed session lengths), the fraction of advance reservations, and the individual weights (set by the provider) for utilization, blocking and preemption, we can find a suitable lookahead time.

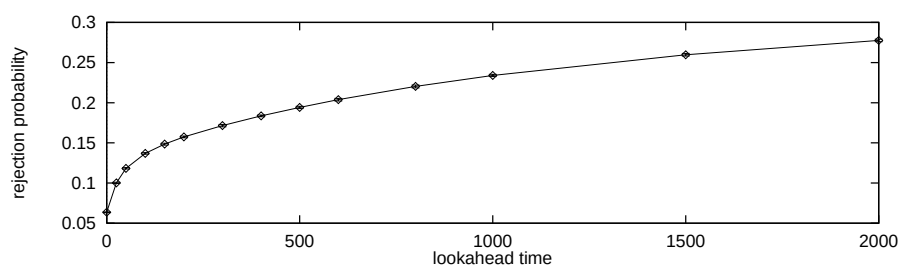


Figure 3.13: Rejection probability for immediate requests, at 40% advance request ratio

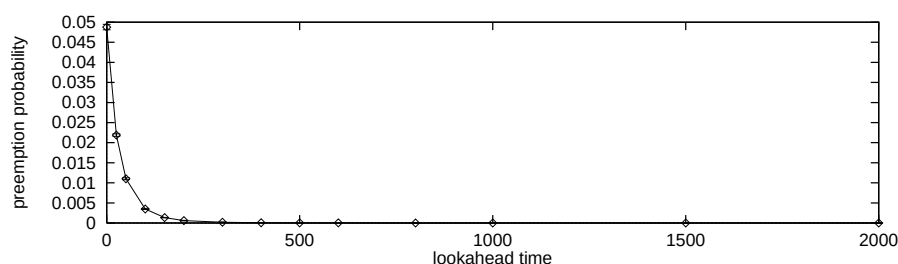


Figure 3.14: Preemption probability for immediate requests, at 40% advance request ratio

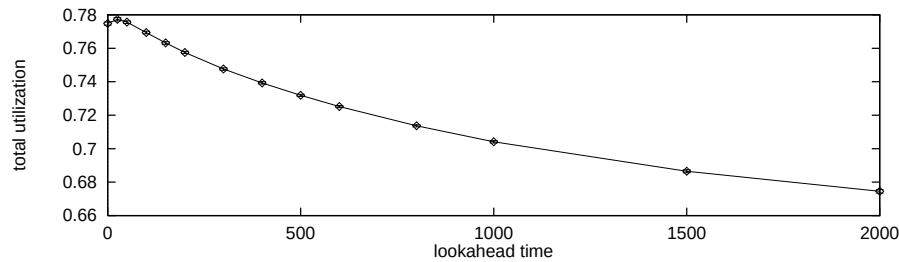


Figure 3.15: Utilization for immediate requests, at 40% advance request ratio

With a 40 % advance reservation request rate, we can, for example, use a lookahead of 50 time units, resulting in 77.6% of the resources being allocated under a 1.1% immediate request preemption probability (i.e., 0.44% total preemption probability). By using a lookahead of 100 time units, we get a utilization of 76.9% at a preemption probability of 0.35% for immediate requests. This is to be compared with 79% of the resources being allocated when there are no advance reservations. The offered load is in both cases 85% of the total capacity and the total rejection probability is around 7%.

If there were reservation agents without supporting lookahead in the immediate admission control algorithm, the results would be comparable with having zero lookahead (the network must support preemption and non-blocking resource allocation that can be used by the advance reservation agent). In that case, we have 5% preemption probability for immediate requests when 40% of the admission requests are made in advance.

3.4.6 Evaluation

Our simulations show the cost of advance reservations when they use the same holding-times and have the same resource demands as immediate reservations. We do this to isolate the impact of reserving in advance versus reserving immediately. By using different parameters for advance and immediate reservations, utilization increases when there are more advance reservations. Given a certain offered load, it is easy to show that there is a benefit from making large reservations in advance. It is generally easier to obtain a large utilization if we start packing the largest requests into the time-bandwidth diagram shown in figure 3.5. In reality, we expect that users needing large resources will tend to make reservations in advance to avoid being blocked.

3.5 Conclusion

In this paper we claim that resources can be more accurately allocated over time by supporting advance reservations for scheduled events. The price of supporting advance admission control is a small but controlled decrease in network utilization. This is compared with users making immediate reservations for their scheduled events. When resources are scarce, users tend to ask for admission ahead of time to make sure that they have resources granted in due time for their own scheduled event. Users make their decision with little information about other scheduled events and with little concern about the overall utilization of the network. In some situations, the effective utilization and the probability of having reservation requests granted can drop without control.

Our simulations show the effects of providing advance reservations. We assume that we allocate resources at some point for each reservation and the utilization is measured in terms of how much we can allocate. We do not consider drops in link utilization due to reserved resources not being used.

In our model, resources are shared between advance and immediate admission control, i.e., there is no pre-partitioning of resources. The most important parameter in the admission control algorithm is the so-called *lookahead* time, i.e., the point at which we actually start making the resources in the routers available for approaching advance reservations. We show how a suitable lookahead time can be found through simulations.

The effects of advance reservations can be expressed in terms of rejection probability, preemption probability, and total utilization. In the simulations presented, we can allocate 79% of the link resources when there are no advance reservations and around 77% of the resources when 40% of the admission requests are made in advance, and there is a 0.4% preemption probability for immediate requests. The offered load is in both cases 85% of the total capacity and the total rejection probability is around 7%.

Acknowledgments

This work was supported by a grant from the Center for Distance Spanning Technology (CDT), Luleå, Sweden, and Ericsson Radio Systems, Sweden.

Part 4

Advance Reservations for Predictive Service in the Internet

Advance Reservations for Predictive Service in the Internet¹

Mikael Degermark², Torsten Köhler^{2 3}, Stephen Pink^{2 3}, and Olov Schelén²

²Dept. of Computer Science
Luleå University
S – 971 87 Luleå, Sweden
{micke, olov}@sm.luth.se

³Swedish Institute of Computer Science
PO box 1263,
S – 164 28 Kista, Sweden
{steve, tk}@sics.se

Abstract

We extend a measurement-based admission control algorithm suggested for predictive service to provide advance reservations for guaranteed and predictive service while keeping the attractive features of predictive service. The admission decision for advance reservations is based on information about flows that overlap in time. For flows that have not yet started, the requested values are used, and for those that have already started measurements are used. This allows us to estimate the network load accurately for the near future. To provide advance reservations we ask users to include durations in their requests. We present simulation results to show that predictive service with advance reservations provides utilization levels significantly higher than those for guaranteed service, and comparable to those for predicted service without advance reservations. Those utilization levels are reached without any preemption of other admitted flows. Finally, we discuss how to setup advance reservations over multiple hops in the Internet using resource reservation setup protocols.

4.1 Introduction

Real time multimedia applications will share future networks with traditional data applications. To provide quality-of-service (QoS) for real time applications, it is likely that resource reservations will have to be made in the network. Current resource reservation protocols allocate resources just before communication begins, e.g., ST-2 [20] and various ATM signaling protocols reserve resources during connection establishment. This model of communication may not fit the needs of future network users [60, pp. 44–45].

¹This work was supported by a grant from the Center for Distance Spanning Technology (CDT), Luleå, Sweden.

Resource reservations should be optional and decoupled from the starting time of the session. One should be able to reserve resources prior to or during a network session depending on when a specific service is needed. Users may know far in advance of their needs and would like to plan their activities by making advance reservations to ensure that they are not blocked by the network's admission control mechanism. Imagine some users with busy schedules in different time zones who want to have an important teleconference on a resource-limited network at an agreed time in the near future. They should be allowed to make an advance reservation given that they know when and for what duration their teleconference will take place.

In this paper we will look at an important candidate for an admission control algorithm originally proposed in [44] and later refined in [45], for a service called predictive service [15]. Predictive service provides quality of service for applications that can tolerate some loss such as real time digital audio and video applications that can adjust their playback points in response to jitter in the network. The efficiency gain of predictive service comes from allowing more flows into the network than guaranteed service, thus providing more sharing and lower cost. The architecture described in [15] supports guaranteed and predictive service, but not advance reservations.

The possibility of making advance resource reservations should be a part of a communication architecture to provide better service to the users. Whether advance reservations are actually needed depends on future resource scarcity. Where resources are plentiful, not even immediate reservations may be necessary, but where resources are scarce enough to justify reservations at all, it makes sense to be able to make them in advance. In this paper we will show that advance reservations can be provided by the network with little overhead.

4.2 Framework

The service model and the admission control algorithm suggested in this paper are extensions of those presented in [15] and [45]. In [15], the proposed network service interface offers guaranteed service, predictive service and best-effort (ASAP) service. The service interface relies on token bucket traffic shaping; the source specifies the bucket size b and the token generation rate r . Guaranteed service provides a minimum transmission rate and therefore the queuing delay bound becomes the bucket size divided by the rate. Predictive service provides K different service classes with widely spaced target delay bounds D_i and it is suggested that the target bounds are spaced by an order of magnitude. The bounded quantity is the queuing delay per hop, so it is necessary to add up the target delay bounds at each hop to find the upper bound on the total queuing delay.

To support this service interface a scheduling algorithm is presented in [15]. The guaranteed service traffic is scheduled with weighted fair queuing (WFQ) [21] so that each guaranteed service client has a separate WFQ flow. All the predictive service flows and ASAP traffic share the spare bandwidth in a pseudo-WFQ flow, called flow 0. The available bandwidth for flow 0 is therefore $\mu - \hat{\nu}_G$ where μ is the link

bandwidth and $\hat{\nu}_G$ is the measured bandwidth usage for all guaranteed flows over the link. Inside flow 0, there are a number of strict priority classes: one class for each target delay bound and ASAP traffic at the lowest priority. The strict priority scheme implies that queuing delay experienced by higher priority classes will be conveyed to lower priority classes.

Admission control is performed in each switch along the path of a flow. Admission requests will be carried to the switches by an end-to-end resource reservation protocol.

Note that our use of the term “guaranteed service” in this paper is adopted from [15]. There are other ways to provide guaranteed service which may give better utilization, e.g., jitter-EDD [27].

4.3 Duration Intervals

To achieve an efficient scheme for advance reservations without preemption we have extended the service interface so that each admission request includes a duration interval: $I = [t_s, t_e]$, where t_s is the starting time of the requested service and t_e is the finishing time of that service. Knowledge of these intervals is necessary for the admission control algorithm to determine which requests overlap and when the reserved resources will be released. As a special case, requests for immediate admission will specify *now* as their starting time. Note that a duration interval refers to the resource allocation only, sending packets outside the interval is possible but as no resources are allocated to those packets they will receive a lower service quality.

A user that does not know how long a session will be should be able to use a default value or a value derived from a personal profile. If a resource allocation turns out to be shorter than needed, it will be possible to renegotiate by calling admission control again. If the new request is granted the duration of the resource allocation is extended. If not, the session is in danger of losing service quality when the resource allocation expires. Default intervals could be renegotiated automatically just before they expire.

If a resource allocation turns out to be longer than needed, other advance reservation requests might unnecessarily be rejected. Fortunately, the unused resources can be given to those who are requesting immediate admission. This is because the measurement procedure of predictive service automatically detects unused capacity once a flow is active. Therefore, over-reservation has little impact on the total utilization as long as there are some immediate requests for admission. In addition, there is an option for users to explicitly cancel the allocation before it expires.

4.4 Admission Control Decision for Advance Reservations

The admission decision for predictive service is based on requested rates for flows that have not yet started and on measured rates for currently active flows. If there are no advance reservations and a request for immediate admission arrives, our extended conditions give the same result as the conditions stated in [45].

Figure 4.1 is a snap-shot of admitted flows in a time/bandwidth diagram. Flows a, b and c are currently active and we have measurements of their rates and maximum delays which are used as predictions of their future behavior. When a new admission

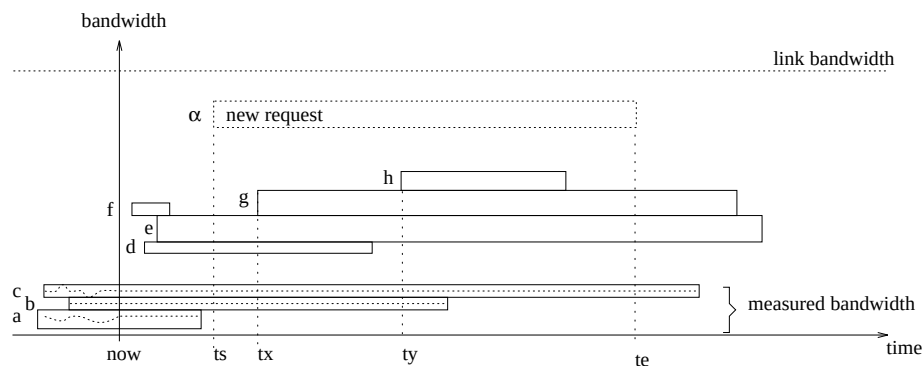


Figure 4.1: Snap-shot of reservations.

request arrives, admission is granted if the new flow would not cause any delay bounds to be violated or bandwidth limits to be exceeded. The admission conditions (section 4.4.1) only consider flows that overlap with the new flow (b,c,d,e,g,h), using measured bandwidth if they have started or, otherwise, bandwidth requests; we call this the *estimated* bandwidth. The conditions are checked at all points where new flows begin (t_s , t_x and t_y).

For reservations in the distant future the number of currently active overlapping flows is small and admission decisions are based mainly on requested rates. In the near future the number of currently active overlapping flows is probably large and admission decisions are based mainly on measured values. So, in the distant future, the admission criteria are conservative, but as time proceeds more overlapping flows will become active and we get better estimates of bandwidth usage. Thus, as we get closer in time to the point at which a flow with an advance reservation is to begin, we have a more accurate knowledge of the network load and more flows can be admitted. Requests for immediate reservation can fill up the remaining bandwidth.

4.4.1 Admission Criteria

A client may request admission for predictive service in one of the classes 1 to K (where class K packets are scheduled at the lowest priority level), or for guaranteed service. The following notation will be used in the formulas² describing our admission criteria:

$\nu_{G(t)}$	estimated bandwidth for guaranteed flows at time t
$\nu_{P(t)}$	estimated bandwidth for predictive flows at time t
$\nu_{P_i(t)}$	estimated bandwidth for flows in predictive class i at time t
$R_{G(t)}$	requested bandwidth for guaranteed flows at time t
$\hat{D}_j$	measured delay in predictive class j
$B_j(t)$	bucket size sum for not yet started flows in predictive class j .

Predictive service: When a client requests service in predictive class k for a flow α , shaped by token bucket filter $(r_k^\alpha, b_k^\alpha, I^\alpha)$, the admission control algorithm performs the following checks:

- Determine if the bandwidth usage, after adding the new load r_k^α , will exceed the available link capacity $v\mu$ during the requested interval I^α :

$$v\mu > \max_{t \in I^\alpha} (r_k^\alpha + \nu_{G(t)} + \nu_{P(t)}) \quad (4.1)$$

The available link capacity, $v\mu$, is determined by the link capacity μ and the link utilization target v , that is tunable.

- Determine whether the worst possible behavior of the new flow and the other flows that have not yet started can cause violation of delay bounds for predictive service classes k through K .

The worst case is when all predictive service flows flush their entire token buckets simultaneously in one burst. The resulting queue will be emptied according to the available bandwidth.

- check the delay bound, D_k , of the same priority level:

$$D_k > \max_{t \in I^\alpha} \left(\hat{D}_k + \frac{b_k^\alpha + \sum_{i=1}^k B_i(t)}{\mu - \nu_{G(t)} - \sum_{i=1}^{k-1} \nu_{P_i(t)}} \right) \quad (4.2)$$

- check the delay bound of the lower priority levels, i.e., D_j where $k < j \leq K$.

$$D_j > \max_{t \in I^\alpha} \left(\frac{\hat{D}_j \left(\mu - \nu_{G(now)} - \sum_{i=1}^{k-1} \nu_{P_i(now)} \right) + \left(b_k^\alpha + \sum_{i=1}^j B_i(t) \right)}{\mu - \nu_{G(t)} - \sum_{i=1}^{k-1} \nu_{P_i(t)} - r_k^\alpha} \right) \quad (4.3)$$

²These formulas are extensions of those presented in [45]

Guaranteed service: When a client requests guaranteed service for a flow α shaped by $(r_G^\alpha, b_G^\alpha, I^\alpha)$, the admission control algorithm first performs the total bandwidth check expressed in (1), then the following checks are performed:

- Determine whether the requested bandwidth of all guaranteed service flows will exceed link capacity:

$$v\mu > \max_{t \in I^\alpha} (r_G^\alpha + R_{G(t)}) \quad (4.4)$$

- Determine that the delay bounds of each predictive service class is still observed when the remaining bandwidth is decreased (estimated bandwidth for guaranteed flows and for predictive classes with higher priority is subtracted from the link bandwidth).

$$D_j > \hat{D}_j * \max_{t \in I^\alpha} \left(\frac{\mu - \nu_{G(now)} - \sum_{i=1}^{j-1} \nu_{P_i(now)}}{\mu - r_G^\alpha - \nu_{G(t)} - \sum_{i=1}^{j-1} \nu_{P_i(t)}} \right) \quad 1 \leq j \leq K \quad (4.5)$$

4.4.2 Operation of Admission Control Algorithm

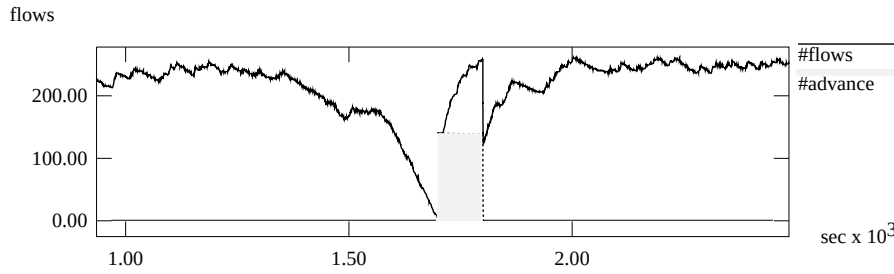


Figure 4.2: Plot of number of active flows when there is a large block of flows admitted in advance.

Figure 4.2 illustrates how our admission control algorithm operates. At time 1700 a large number of sources start. These sources, which were admitted in advance, have reserved all of the available bandwidth and all finish at time 1800. There is a background of sources asking for immediate admission with predictive service. The figure clearly shows that the number of active flows goes down to zero just before time 1700 because admission control rejects new flows to honor the resource commitments to the previously admitted sources, and because active sources finish. At time 1700 the number of active flows increases sharply as the previously admitted sources begin to transmit. As the aggregate traffic is measured and found to consume only part of the resources needed to accommodate the worst case according to the token bucket parameters of the admission requests, more sources can be admitted and the flow count increases further.

4.4.3 Renegotiations

If a request is made for extending the duration of an already active flow it would be very conservative to base the admission criteria on peak-rate values obtained from the token bucket specification. Instead the available measurements of the flow could be used to predict its future behavior. This can be done by checking the conditions (1) to (3), replacing the bucket rate r_k^α with the measured rate and replacing the bucket depth b_k^α with zero (so that it has no effect on the admission decision). This means that the future behavior of any active flow is given by the measured values only. The conditions need to be checked at the points in the requested interval where previously admitted flows will start. This ensures that the extended duration will be granted only if no previously admitted reservations are jeopardized.

4.5 State Requirements

To make the admission control decision, the original admission control algorithms [44] [45] need the current bandwidth use for all guaranteed flows plus maximum delay and bandwidth use for each predictive service class. These are measured values and per-packet processing is needed to maintain them.

The information needed to make advance admission decisions in a switch is summarized in Figures 4.3 and 4.4. Figure 4.3 shows the cumulative requested bandwidth

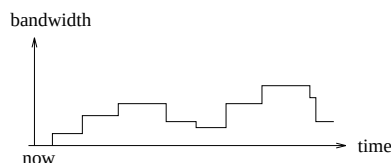


Figure 4.3: Cumulative requested bandwidth

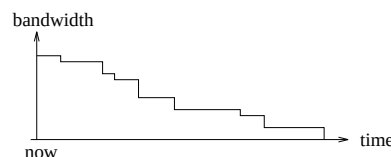


Figure 4.4: Predicted bandwidth use of currently active flows

admitted to flows that have not yet started. Figure 4.4 shows the predicted bandwidth use of currently active flows. Present measurements are used as predictions of future bandwidth use for those flows. We need to keep state corresponding to these diagrams for the guaranteed flows collectively and for each predictive service class individually. Note that the state corresponding to figure 4.3 need to be accessed only when making an admission decision, while the state corresponding to figure 4.4 are measured values and per-packet processing is needed to maintain it.

A straightforward implementation for advance reservation would keep an amount of state proportional to the number of active flows plus the number of flows reserved in advance. Aggregation methods, however, can decrease the amount of state needed for the admission control decision: flows that start or finish at the same time, or nearly the same time, can be treated collectively.

There is a tradeoff between the amount of state saved by aggregating requests and the flexibility of making requests. A simple way to aggregate requests is to use time slots. Duration intervals may then start and finish only at certain points in time. A disadvantage with this scheme is internal fragmentation: clients may have to reserve longer intervals than they will actually use.

Any scheme for resource allocation in a packet switched network requires that switches can find the appropriate resource allocation when a packet is received, so that the packet can be scheduled correctly. It seems inevitable that this mapping requires an amount of state linear in the number of active reservations if resource reservations can be made for individual flows. For instance, the values that identify flows with reservations (addresses, port numbers, flow identifiers, etc) will typically be kept by the switch.

To summarize, regardless of whether advance reservation is supported or not, predictive service requires that a switch keeps, and accesses on a per-packet basis, an amount of state that is no more than linear in the number of active reservations. To support advance reservations, our admission control algorithm needs to keep an additional amount of state not more than linear in the number of admitted flows that have not yet started. This additional state needs to be accessed only when admission decisions are being made.

4.6 Simulations

Our simulations aim to show that adding advance reservation capability to the admission control algorithm for predictive service does not decrease utilization levels very much. This implies that advance reservations for predictive service give significantly higher utilization levels than advance reservations for guaranteed service.

The simulated network topology is a single link, and the source models are simple on/off models with exponentially distributed on and off times. We have done simulations of scenarios with immediate reservations only, and with both immediate and advance reservations. We have also examined the effects of aggregating state for active flows with similar finishing times. Finally, as it is likely that advance reservations will be made a long time in advance and have longer durations than immediate reservations, we have studied how such scenarios affect utilization levels.

4.6.1 Simulated Topology

The simulated topology is a single 10 Mbit/s bottleneck link connecting two routers. A number of sources are connected to one of the routers with links of infinite bandwidth. All sources send data to a sink connected to the other router. Our data comes from the upstream router R (fig 4.5).

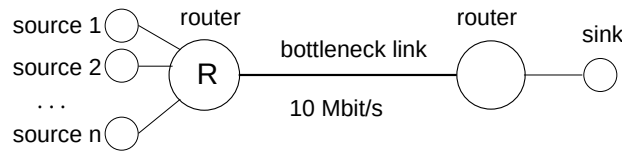


Figure 4.5: Simulated topology

Model Name	Model parameters				Token bucket		Delay bounds	
	p pkts/sec	I msec	N pkts	p/a	r tkns/sec	b tkns	D msec	D_j msec
Exp1	64	325	20	2	64	1	16	16
Exp2	1024	90	10	10	320	50	160	160
Exp3	10^7	684	9	76000	512	80	–	160

Table 4.1: Source model parameters

4.6.2 Source Model Parameters

We use three kinds of sources, all generate packet trains at some peak rate p . The train length is exponentially distributed with mean N . The time between packet trains is also exponentially distributed with mean I . The ratio between the peak and average rate, p/a , can be calculated from those values.

All sources regulate their output with a token bucket filter with token generation rate r and bucket depth b . Each token is worth 1000 bits which is equal to the packet size; sending one packet consumes one token. If the bucket is empty the packet is queued until a token is available. The token bucket filters in table 4.1 are designed to always have a token available when the source wants to output a packet.

All source parameters are listed in table 4.1. In the table, D is the maximum delay for a guaranteed flow, calculated from the token bucket parameters. D_j is the requested delay bound when the source asks for predictive service. The router supports two predictive service classes, one with a delay bound of 16 ms and the other with a delay bound of 160 ms.

4.6.3 Flow Generation

Sources ask for admission according to a poisson process; the times between admission requests are exponentially distributed with a mean of 400 ms. In all simulations, the offered load is about 2.4 times larger than the capacity of the link, so most admission requests are rejected.

In the first set of simulations, the requested duration intervals are exponentially distributed with a mean of 300 seconds. For sources that ask for admission in advance, the times between the admission request and the start of the duration interval,

henceforth called the *booking times*, are exponentially distributed with a mean of 5 min.

In the second set of simulations part of the load, henceforth called the *uniform* load, comes from sources with durations that are uniformly distributed in an interval $\pm 50\%$ of their mean. These sources use booking times that are also uniformly distributed in an interval $\pm 50\%$ of their mean. The offered uniform load is kept constant by decreasing the number of admission requests when durations are large. E.g., when durations have a mean of 30 min the number of admission requests are 6 times fewer than when durations have a mean of 5 min.

A source requests admission by sending a setup packet containing the desired service type and token bucket parameters towards the destination. If all routers along the path grant admission, the source transmits during the requested interval and then it stops.

4.6.4 Measuring Process

The measuring process estimates current bandwidth utilization $\hat{\nu}$ and experienced maximum delay $\hat{D}$ in the same way as in [45]. When deciding whether to admit an



Figure 4.6: Estimated future bandwidth use for currently active flows

advance reservation, the algorithm needs estimates of bandwidth utilization in the future, e.g., at time t in fig 4.6. This is done by continually estimating current bandwidth utilizations and using these as predictions of future utilization. The estimates are obtained by a straightforward extension of the measuring process in [45]. The packet rate of every active flow is sampled; these rates are then used as in [45] to obtain estimates of bandwidth utilization between finishing points of flows. The sum of the rates of the bottom three flows in figure 4.6 are used to estimate $\hat{\nu}$ between $f3$ and $f4$, i.e., the finishing points of flows three and four from the bottom. This procedure ensures that the estimates are conservative in the distant future and accurate in the near future where many currently active flows will still be active.

In a straightforward implementation, calculating the estimates is linear in the number of flows. To avoid keeping track of every individual flow and reduce the overhead in calculating the estimates, we have experimented with aggregating flows that finish at about the same time. The bandwidth utilization of the aggregated flows is then estimated collectively.

Name	Model	Serv	util %	delay (ms)	# sources (avg)			Measuring params		
					act	adv	adm	$T(s)$	$S(s)$	$A(s)$
IMM	Exp1	G	45	2.6	140	—	—	5.0	0.80	—
ADV	Exp1	G	44	2.3	137	137	282	5.0	0.80	—
IMM	Exp1	P	78	2.3	244	—	—	5.0	0.80	—
ADV	Exp1	P	68	1.9	213	160	265	5.0	0.80	—
GRA	Exp1	P	70	1.9	219	164	249	5.0	0.80	32
GRA	Exp1	P	75	2.5	232	153	240	5.0	0.80	128
IMM	Exp2	G	28	9.3	28	—	—	1.0	0.12	—
ADV	Exp2	G	27	8.4	27	27	109	1.0	0.12	—
IMM	Exp2	P	76	37.0	75	—	—	1.0	0.12	—
ADV	Exp2	P	59	13.4	58	37	124	1.0	0.12	—
GRA	Exp2	P	54	11.1	54	37	122	1.0	0.12	32
GRA	Exp2	P	50	11.5	49	33	123	1.0	0.12	128

Table 4.2: Simulation results, durations exponentially distributed

4.6.5 Simulation Results

To verify our simulation environment we first replicated some relevant results from [45] in our simulator. In this first set of simulations, all sources in a single simulation conformed to the same source model and all requested immediate admission for the same type of service. The results of these simulations are summarized in Table 4.2 under IMM. In table 4.2; *util* is the utilization of the bottleneck link and *delay* is the maximum experienced queuing delay. *# sources* are averages of flow counts; *act* is the average number of sources that were transmitting. The *measuring params* are the size of the T and S windows used in the measuring process (see [45]).

The utilization target v was 90% in all simulations. All simulations ran for at least 3000 seconds simulated time. The data in table 4.2 comes from the second half of the simulated time. Visual inspection confirmed that no startup transients remained at that time.

In the simulations with advance reservations, 50% of the sources asked for immediate admission and 50% for admission in advance. The choice was random. All sources conformed to the same model and asked for the same type of service. These simulation results are summarized in table 4.2 under ADV. There, *adv* is the average number of sources that were transmitting and were admitted in advance, and *adm* is the average number of sources that were admitted in advance but have not begun transmitting.

The GRA simulations are similar to ADV, the only difference being the measuring process: all flows that finish within the same A seconds are aggregated and measured collectively. This also implies that for purposes of admission control, finishing times are rounded upwards to the nearest A seconds. In the table, A is the granularity of the measuring process.

4.6.6 Discussion

These simulations clearly show that predictive service with advance reservations provides higher network utilization than guaranteed service with advance reservations. They also show that adding advance reservation capability to predictive service decreases bandwidth utilization. The levels are not much lower though for smooth traffic, but for bursty traffic the utilization level decreases more. When the fraction of sources asking for advance admission is lower the decrease in utilization is lower. E.g., in simulations when 10% of the sources (instead of 50%) ask for admission in advance, the utilization is 69% for the burstier Exp2 traffic.

The reason for the decrease in utilization is that advance reservations will block requests for immediate admission. This blocking effect is larger when the sources are bursty since the token bucket parameters are larger. Moreover, when token buckets are deep the admission decision is based on delay considerations more than on available bandwidth. To make a good admission decision in this case, the algorithm would need to know how much each flow contributes to the current queuing delay. This would enable the algorithm to estimate future delay since it knows which flows will be active at any future time. Instead, the algorithm uses the current delay as an estimate of future delay. Since this is a very conservative estimate, network utilization suffers.

An interesting and somewhat surprising result is that when there was aggregation of flows in the measuring process utilization increased for the smooth traffic generated by Exp1 sources, but decreased for the burstier Exp2 sources. It is easy to see that utilization might decrease since the durations of active reservations are virtually extended by the aggregation in the measuring process. This gives less room for new reservations. The simulation results are not conclusive on why aggregation increases utilization for the smooth Exp1 sources.

An interesting observation is that for guaranteed service, sources asking for immediate admission are almost completely shut out by sources asking for admission in advance. This is due to the fact that for guaranteed service the admission decision is based on requested values only, regardless of estimated bandwidth use. The sources asking for admission in advance are admitted first and so can starve out sources asking for immediate admission since no bandwidth is freed when the sources with guaranteed service begin to transmit.

4.6.7 Booking Time and Duration

These simulations investigate how booking times and durations are related to utilization. Table 4.2 shows that there is virtually no decrease in utilization levels for guaranteed service when there are advance reservations, so we concentrate on predictive service. No effort was spent in finding good parameters for the measuring process, so it is likely that utilizations levels could be improved somewhat. [45] discusses the effects of changing various measurement parameters.

In this and all following simulations, 60% of the load has zero booking times and durations that are exponentially distributed with a mean of 5 min. 40% of the offered

Mean duration (min)	5	10	15	30	45	60	75
Exp1 util (%)	79	81	82	78	78	83	83
Exp2 util (%)	78	79	80	80	81	81	81
Exp3 util (%)	40	42	43	46	45	47	46

Table 4.3: Zero booking time

Mean booking time (min)		5	15	30	45	60	75
Exp1	util (%)	80	79	79	78	79	78
	#adv	162	136	132	133	134	132
Exp2	util (%)	76	76	76	76	77	76
	#adv	36	26	26	25	25	25
Exp3	util (%)	16	19	20	19	21	21
	#adv	30	15	15	15	15	15

Table 4.4: Effect of booking time. Mean duration is 10 min.

load is uniform as explained in 4.6.3. The parameters in tables 4.3, 4.4, and 4.5 refer to the uniform load only. All sources in a simulation obey the same source model, and all request predictive service. Simulations ran for 7500 seconds simulated time and the reported results come from the last 1500 seconds. No delay bounds were violated.

Table 4.3 gives utilization levels for a simulation series where all booking times were zero, i.e., all requests are for immediate reservation. This allows comparison with later results using non-zero booking times. Table 4.3 generally show a tendency of increasing utilization as the durations of the uniform load increase. This is expected as the number of active flows with long durations increase and there are fewer dips when flows leave.

The uniform load in the next set of simulations have duration means of 10 minutes. The booking times vary. Results are reported in Table 4.4, which lists utilization levels, *util*, and average number of active reservations that were booked in advance, *#adv*. Table 4.4 show that utilization levels are independent of the length of the booking time when the booking time is longer than the duration intervals. As there is no overlap with active flows at the time of admission, the number of flows that can be admitted in advance is determined by how much will fit into the link given the token bucket parameters. Simple calculations on the token bucket parameters and comparison with the reported *#adv* numbers confirm this. The utilization levels for the smoother Exp1 and Exp2 models decrease only a few percent compared to the simulations with zero booking time. For the Exp3 model utilization levels decrease dramatically, from 42% to 21% when durations are 75 min. Note, however, that the utilization level would be less than 2.7% if guaranteed service were requested. The utilization decreases because of the blocking that occur just before reservations made in advance become active, see Figure 4.2 for an illustration of this effect.

Mean duration (min)		5	10	15	30	45	60	75
Exp1	util (%)	69	79	82	84	84	85	86
	#adv	130	144	153	187	205	219	224
	#advstarts	628	342	244	152	109	93	65
Exp2	util (%)	71	76	78	79	80	80	82
	#adv	26	26	28	38	50	52	57
	#advstarts	133	69	52	31	24	19	20
Exp3	util (%)	11	19	24	28	30	35	37
	#adv	15	15	19	29	42	50	63
	#advstarts	80	42	29	27	22	21	24

Table 4.5: Effect of Duration. Mean booking time is 15 min.

The final simulations keep the mean booking time fixed at 15 min, and the duration means vary from 5 to 75 minutes. Table 4.5 list the results. There, **#adv** is the number of active flows that were reserved in advance and **#advstarts** is the number of such flows that started within the 25 minute interval.

Table 4.5 show that longer duration intervals increase utilization levels. The reason appears to be that the number of starts of reservations that were booked in advance decrease as the length of duration intervals increase. The number of accompanying dips then also decreases. Utilization levels for the bursty Exp3 sources fluctuate significantly due to the uneven distribution of starting times of advance reservations. A plot of the utilization level and number of starting advance reservations per time unit clearly show this phenomenon, the utilization level drops significantly before a high concentration of starts.

When booking times are shorter than durations, the number of active flows that were admitted in advance will increase. This does not appear to have much effect on utilization. A 20000 second simulation where durations had a mean of 75 min, and booking times were large, resulted in 36% utilization for Exp3 sources. Compare this to the 37% utilization when booking times had a mean of 15 min (Table 4.5).

To conclude, these simulations show that when durations of advance reservations are long, utilization levels are about the same as the levels without advance reservations, unless token bucket parameters are very large in which case the levels decrease somewhat due to blocking before the starting times of reservations booked in advance. When booking times are longer than a few minutes, utilization levels are not related to the length of the booking time. Utilization does depend on the duration of advance reservations, longer duration gives higher utilization. There were no violations of delay bounds in any simulation.

4.7 Setup protocols for advance reservations

To provide an end-to-end advance reservation service in an internetwork, it is necessary to have a protocol which is able to set up reservations in advance across multiple hops. So far, we have been concerned only with admission control into a single hop. There have been some protocols suggested for setting up reservations in a packet switched network, including ST-2 [20], RSVP [80], Q.2931 [63], RCAP [4]. The first two protocols were designed to provide reservations in the Internet. One difference between them is that ST-2 relies on *hard* state and RSVP on *soft* state. Another difference is that in ST-2 the sender makes the reservations and in RSVP it's the receiver. For our purposes, *hard state* means that the reservation state in the routers, once set up, will remain in those routers until explicitly torn down. By *soft state* we mean that reservation state must be refreshed periodically and will time out otherwise. In practice, this means that as routes change, the reservation state in RSVP will time out and be rebuilt along the new path.

4.7.1 Setting up Advance Reservations using RSVP

With some minor changes, RSVP [80] can be used to set up advance reservations in almost the same way as it is used for setting up immediate reservations. To establish an advance reservation for a multi-party session the senders have to announce their session by periodically sending announcement messages (in RSVP terms, "PATH" messages) down a multicast tree. Receivers respond to those announcements by sending reservations toward the senders. The resources have at this point been reserved for some time in the future. At the time the session starts, resources are allocated and the service to each session participant increases from best-effort to the requested quality.

For reservations made far in advance, there is potentially a very large number of path and reservations messages that must be sent before the session begins. To reduce overhead, the frequency of sending these messages should start low and increase as the time of the session approaches. RSVP could support advance reservations efficiently while allowing the admission control algorithm and measuring process to aggregate session state if the following two minor changes are made:

- To support advance reservations the flow specification carried by RSVP path and reservation messages should include session durations. A sender will state a duration for the session and the receivers are free to reserve any interval within that duration. Since senders may lengthen or shorten durations, special wildcard durations can be used by the receivers to follow the changes made by the sender.
- To cancel a reservation, RSVP should provide the original flowspec in the interface between RSVP and the admission control mechanism. This is because our admission control algorithm aggregates requests for sessions of similar duration to save state and for measuring purposes. We propose that RSVP provide the

original flow specification when making a call to the admission control mechanism to delete a session.

Setting up advance reservations with RSVP is attractive because relatively small changes are needed to the protocol. Also, RSVP seems to be a setup protocol that will be widely used in the Internet. However, there are two problems with the scheme.

First, both senders and receivers need to be present when the reservation is made and during the whole period before the reserved interval starts. The actual application need not be present, but some agent on each involved host needs to join the multicast group and tell the local RSVP agent to start sending and receiving path and reservation messages. If hosts fail or are temporarily cut off from the network, the reservation may time out. Even worse, the sender and/or receiver hosts may not even be present when the reservation needs to be made³. A truly flexible resource setup protocol should allow resource reservations by third parties as well as receivers and/or senders.

Second, RSVP currently does not prevent route changes. Changed routes will cause the reservation messages of existing reservations to pass through new routers, which may or may not have sufficient resources to grant the reservations. If not, some reservations will be rejected, which introduces an ordering problem in rebuilding advance reservation state. Thus, earlier reservations can be shut out by later reservations whose request happens to arrive first at the router in the new path. Such behaviour will decrease the utility of advance reservations. If you cannot trust an advance reservation to be present at session time, why bother to make it? There appears to be a strong connection between a resource reservation and the path over which it holds.

4.7.2 Soft versus Hard State Setup Protocols

Hard-state-based resource reservation protocols avoid the problem of changing routes but at the cost of flexibility. Some routing changes that are due to failures are unavoidable. When a hard-state based resource reservation protocol encounters such a routing change, its only available action might be to reestablish the connection on a new route or tear down the connection. Reestablishing the connection on the new route has the same consequences for the user as RSVP's way of rebuilding state on the new path: reservations could be rebuilt in such a way as to deny service to users that had earlier reservations. On the other hand, tearing down the connection completely and not reestablishing a new one seems an inflexible way of handling the situation if a new route in fact exists. Only an "all-or-nothing" user would desire such a service.

Route "pinning" has been suggested for RSVP to avoid situations where the route protocol "flaps" between two different paths without a failure on either one. This could happen, for example, when while routing table calculations attempt to converge after a change. Such "flapping" would needlessly disturb the quality of service for an application. Route pinning would allow RSVP to keep a path stable after a reservation

³This point was made by David Clark

is made by continuing to forward packets on the initial path, even though the route has changed. This might help the situation where advance reservations would otherwise need to be rebuilt along new paths. In a proposed scheme, a *pin* flag is added to the reservation messages of RSVP. When the pin flag is present, the path towards the source is pinned down as long as pin state is refreshed. If real routing failures occurred, pinned routes would be freed and the RSVP path would migrate to the new route. Thus, RSVP would be able to take advantage of the robust features of Internet routing (such as being able to route around “damage”) while preserving quality of service in the face of “flapping” routes.

One way to avoid the situation of having resource reservations rebuilt in a new order and denying service unfairly is to have reservations timestamped to reflect the original time reservations are made. Although this would add more complexity, it could in principle be implemented in a soft-state based as well as hard-state based setup protocol.

Thus, RSVP’s soft-state style of resource reservation would seem to have advantages over setup protocols that use hard state. The major problem for setting up resource reservations in advance with such protocols is the demand that senders and/or receivers must be present at booking time for these schemes to work. This is a problem for both hard and soft-state-based setup protocols. Thus, a third-party approach to making advance reservations on behalf of potentially absent senders or receivers would seem to be needed.

4.8 Related Work

The distinguishing feature of our work is that we offer an admission control algorithm that allows advance reservations for predictive service without preemption. The algorithm results in significantly higher utilization levels for predictive service than for guaranteed service.

There are few papers on advance resource reservation in the literature. The earliest mention of advance reservations in packet switched networks we have found is from 1992 [60].

The work closest to ours is [26], where advance reservations are for a kind of guaranteed service. Their simulations show network wide utilization gains since conference sessions (that share resources) get priority by being reserved in advance. All schemes for advance reservation where reservations are merged, including ours if used together with, e.g., RSVP, would show similar gains. In their scheme a separate partition of the link is allocated to advance reservations. If such partitioning were used for predictive service there would be small or no utilization gains compared to guaranteed service, because partitioning prevents allocating unused resources to immediate requests.

[77] describes a general model for resource reservation in advance. Admission control algorithms are not suggested.

A new scheme called RBONE for making advance reservations in the Mbone (i.e., the Internet multicast backbone) has been presented recently [10]. The key idea is to assign priorities to sessions that are pre-registered. Resource reservation is done as the flow becomes active and RSVP reservation requests carry priorities. The receivers learn what priority to use by an out-of-band mechanism. The routers give resources to the requests with the highest priorities, possibly by preempting reservations with lower priorities. The scheme is independent of routing and routers do not keep much state for advance reservations, except for what is needed to authenticate reservation requests. A disadvantage is that preemption is necessary. The scheme is geared towards large and dense sessions. When few hosts participate in each session, it is necessary that the agents assigning priorities know the topology of the network in order to provide an adequate response to registrations. Users, for example, may want to be notified if their registrations are likely to be preempted.

4.9 Further work

We have used simple on/off source models. Recent studies [34, 50, 62] have shown that real data traffic and VBR video traffic are self-similar, which means that our simple traffic models can give results that are too optimistic. Jamin et al [45] show by simulation that predictive service works well for some self-similar traffic models when flows enter and leave frequently. However, when reservations are made a long time in advance and traffic is self-similar, current estimates of bandwidth and delay can be poor estimates of the future. A natural continuation of our work is to simulate using self-similar traffic.

Another continuation would be to study larger topologies. This has been done by Jamin et al for the original predicted service admission control algorithm. The concern here is that it is possible for existing advance reservations in parts of a multihop path to block new admission requests along that path so that utilization decreases intolerably. The reservation setup protocol and its timing parameters are important here since partially installed reservations that are later rejected in another part of the path could introduce severe blocking under high load [53]. This problem is not specific to predicted service since reservations for other kinds of service would also suffer from such phenomena to some degree.

4.10 Conclusions

We have shown how the predictive service admission control algorithm developed in [44] and [45] can be extended to support advance reservations provided that requests for admission specify the duration of their reservation. Our work shows that it is possible to have advance reservations without preemption.

The extended admission control algorithm proposed in this paper relies on knowledge of which flows overlap in time with the flow that requests advance reservation,

measuring those overlapping flows that are active, and assigning the requested rate to the flows that have not yet started. Thus, more requests for a certain duration of time can be granted as we get closer to that duration of time increasing sharing and lowering cost for those flows that occupy that duration. We have also suggested ways to minimize the amount of state information necessary to provide advance reservations and to simplify the measuring process that estimates future bandwidth use.

Our simulations show that predictive service with advance reservations provides higher network utilization than guaranteed service with advance reservations. They also show that adding advance reservation capability to predictive service decreases bandwidth utilization. The levels are not very much lower though, and they are still significantly higher than for guaranteed service. The decrease in utilization is due to the fact that advance reservations will block sources asking for immediate admission. This blocking effect is larger for bursty sources which request more resources and when durations of advance reservations are short.

We have also discussed how to set up advance reservations in the Internet. RSVP can, with minor changes, be used to set up reservations in advance. There are, however, problems with this solution and we suggest that a more general mechanism might be more appropriate.

Part 5

Performance of Admission Control Agents

Performance of Admission Control Agents

Andreas Nilsson, Joakim Norrgård and Olov Schelén

Computer Science and Electrical Engineering

Luleå University of Technology, SE – 971 87 Luleå, Sweden

{andreas, nogge, olov}@cdt.luth.se

Abstract

We have designed an agent-based architecture for network resource reservations. For each link-state routing domain in the network there is an agent responsible for admission control. The architecture provides resource reservations for uni-directional virtual leased lines. In this paper, we present performance measures for a prototype agent. We evaluate two different data structures for advance reservations. We also compare on-demand route computations with pre-computation of routes. The objective is to evaluate the performance of end-to-end admission control within a link-state routing domain. In a domain with 15 routers, 28 transition networks and 64 stub networks, our prototype can perform approximately 25000 end-to-end admission decisions per second. The results show that an ordinary PC can be used for running a reservation agent that performs path-sensitive admission control and maintains per-link resource reservations in a link-state routing domain.

5.1 Introduction

We have designed a resource reservation architecture where clients make admission requests through *agents* [67, 68, 69, 70]. For each routing domain in the network there is an agent responsible for admission control. Each agent knows the topology and static link resources in its domain. The agent is an end-system that is configured for passively participating in a link-state routing protocol (e.g., OSPF) where each participating router has an identical topological database over the domain. Agents retrieve link properties, such as static bandwidths, by querying routers seen in the topological database. For this, we use a network management protocol (e.g., SNMP). Agents use the resource map to perform admission control in their domain.

The architecture offers a service comparable to unidirectional *virtual leased lines*. Admission requests contain the bandwidth to be reserved, a source and a destination address. Path-sensitive admission control is performed by agents for the links that

will be used. In this paper we describe a prototype implementation. We focus on obtaining performance measures for end-to-end admission control within a link-state routing domain.

5.2 Prototype Setup

5.2.1 Framework

We have implemented a prototype agent as an application program for the FreeBSD operating system [32]. All experimental results presented in this paper are obtained when running the software on an Intel Pentium II, 300 MHz, with 128 MB of memory. The machine is configured as a passive router to obtain the topological database for its link-state routing domain (i.e., the machine does not advertise any routes but it receives link-state advertisements from other routers). For this, we use the OSPF [55] implementation as provided in *gated* [35]. Link bandwidths are collected by an agent from the routers by using SNMP [38]. We use an SNMP implementation provided by ISODE [42].

5.2.2 Topology

The prototype runs in the network at Luleå University of Technology. The OSPF domain comprises 15 routers, 28 transit networks, and 64 stub networks. Transit networks and stub networks are mostly 100 Mb/s or 10 Mb/s Ethernets that may be switched or hubbed internally. To OSPF, the transit networks are perceived as clouds interconnecting at least two routers while stub networks connect one router to a number of end systems. The bandwidths available for admission control are determined by the speed of the interfaces at the routers (as found through SNMP).

The domain is flat (i.e., there are no OSPF sub-areas) and therefore one agent can perform admission control within the whole domain. In a larger OSPF domain with sub-areas, each sub-area would have an admission control agent (i.e., there would be a two level hierarchy).

5.3 Scope

The prototype presented here contains functionality for:

- obtaining a topological database through the link-state routing protocol,
- obtaining link bandwidths through SNMP,
- performing path-sensitive end-to-end admission control within a single link-state routing domain.

We focus on the following performance measures:

- Single-hop admission control speed. This includes comparing the admission decision speed of two different data structures for advance reservations. We compare their sensitivity to the total number of reservations, the maximum book-ahead time, the length of the requested durations, etc.
- Multi-hop admission control speed. In addition, this includes comparing the time for routing decisions when paths are computed on-demand or pre-computed. We also evaluate the protocol processing cost for receiving admission requests over the network.

In the experiments, we cover both immediate and advance reservations. Immediate reservations start as soon they are granted and are open-ended (i.e., clients may hold on to them as long as they like). Advance reservations are time-limited (i.e., they have a starting time and a finishing time). We use bandwidth as the quality-of-service metric.

5.4 Advance admission control

In our model, advance admission control is a two-dimensional problem involving bandwidth and time. Admission control involves fitting boxes (i.e., rectangles) in a two-dimensional time-bandwidth diagram (figure 5.1). For this we have implemented two different data structures supporting continuous or slotted time.

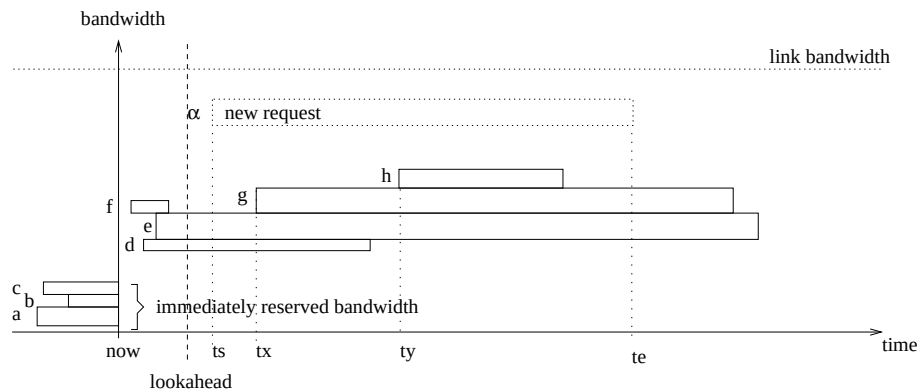


Figure 5.1: Advance reservations in a time-bandwidth domain

5.4.1 Continuous or slotted time

The overhead for advance reservations depend on whether we use continuous time or slotted time. With continuous time, we allow reservations to start and end at

at any time. With slotted time, we enforce a minimum granularity on the starting and finishing times. Slotted time allows more aggregation of reservation state. This results in faster admission control and less memory usage.

In our experiments we have used two different data structures. The first one (section 5.4.2), supports continuous time for reservations, but benefits from having slotted time. The second data structure (section 5.4.3), relies on having slotted time. This data structure was first defined and analyzed in [58], where also other data structures are analyzed (including our data structure supporting continuous time). Below we give a short overview of the two data structures used in our experiments.

5.4.2 A binary search tree over time

The first data structure is a binary search tree over time. Every reservation is represented by at most two nodes in the tree: one for the starting time and one for the finishing time. The node at the starting time contains a positive bandwidth δ and the node at the finishing time contains a corresponding negative bandwidth δ . If there are several reservations starting or finishing at the same time, the sum of bandwidth δ s is stored in a node representing that point in time.

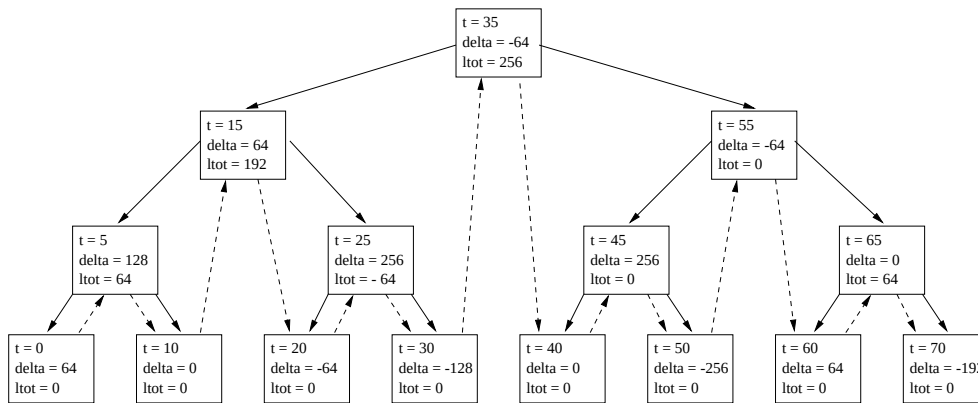


Figure 5.2: A binary search tree over time

To obtain efficient admission decisions, we have added some information to the nodes (figure 5.2). First, the aggregate bandwidth reserved in the left child of each node is stored ($ltot$). This value is updated as new elements are inserted in the left child. Second, there are cross-links (dashed arrows) in the tree so that the list of consecutive nodes in time can be obtained starting from any point in time.

Admission control involves a binary search to find the starting time of the requested duration. In this traversal, the total aggregate bandwidth reserved in left children is computed (i.e., for each step to a right child, the aggregate bandwidth of not traversed left children is updated). Thus, as the starting time is found the total aggregate bandwidth reserved at that time is known. Admission control involves

a linear search through the requested duration. At each point of change, the total aggregate is updated by adjusting for the delta. At each positive delta, we check whether the total aggregate plus the new requested bandwidth is still below the link bandwidth.

The complexity for finding or inserting a given time in this data structure is $O(\lg(n))$, where n is the number of nodes in the tree. The complexity for searching through the duration spanned by a new request is in $O(m)$, where m is the number of nodes in the requested duration. Thus, the complexity of admission control is $O(m + \lg(n))$. However, when slotted time is used, n is bound by the total number of slots and m is bound by the length of the duration divided by the granularity of slots (i.e., by the number of time slots spanned by the requested duration).

5.4.3 A segment tree over time

The second data structure, described in [58], is based on Segment Trees. Each node contains a time frame (duration) and the amount of reserved bandwidth during that time frame (figure 5.3). The time frame of the top node equals the whole period of time that is covered for advance reservations. Each time frame is recursively divided into smaller, equally sized, time frames. The time frame of each leaf is equal to one time slot. Thus, the height of the tree is decided by the time frame of the top node, the branching degree, and the granularity of the leaf nodes.

Each node stores the aggregate bandwidth of all reservations spanning over the whole time frame represented by the node. This value is denoted the *node value*. Each node also stores the maximum sum of node values in any of the branches (i.e., paths) below the current node. This value is denoted the *max value*. An admission request can be viewed as a block with a given duration that falls through the data structure (figure 5.3). The admission operation is performed as follows.

- If the duration of the falling block is within the time frame of the node, the block falls through to the level below.
- If the duration of the block intercepts with the time frames of two or more nodes at a given level, the block is split at the interception points before it falls to the next level.
- However, if the duration of a block fits perfectly within a time frame of a node (before or after a split) it does not fall further.

While a block falls through, the bandwidth reserved at the levels it falls through are added up. When a block fits perfectly in the time frame of the current node, the admission decision for that block is made. This is done by adding the reserved bandwidth in the fallen through levels, the bandwidth reserved in the current node, the max value of the children of the current node, and the bandwidth requested by the falling block. If this sum is smaller than the link bandwidth, the request can be granted (for a sub-duration) and inserted.

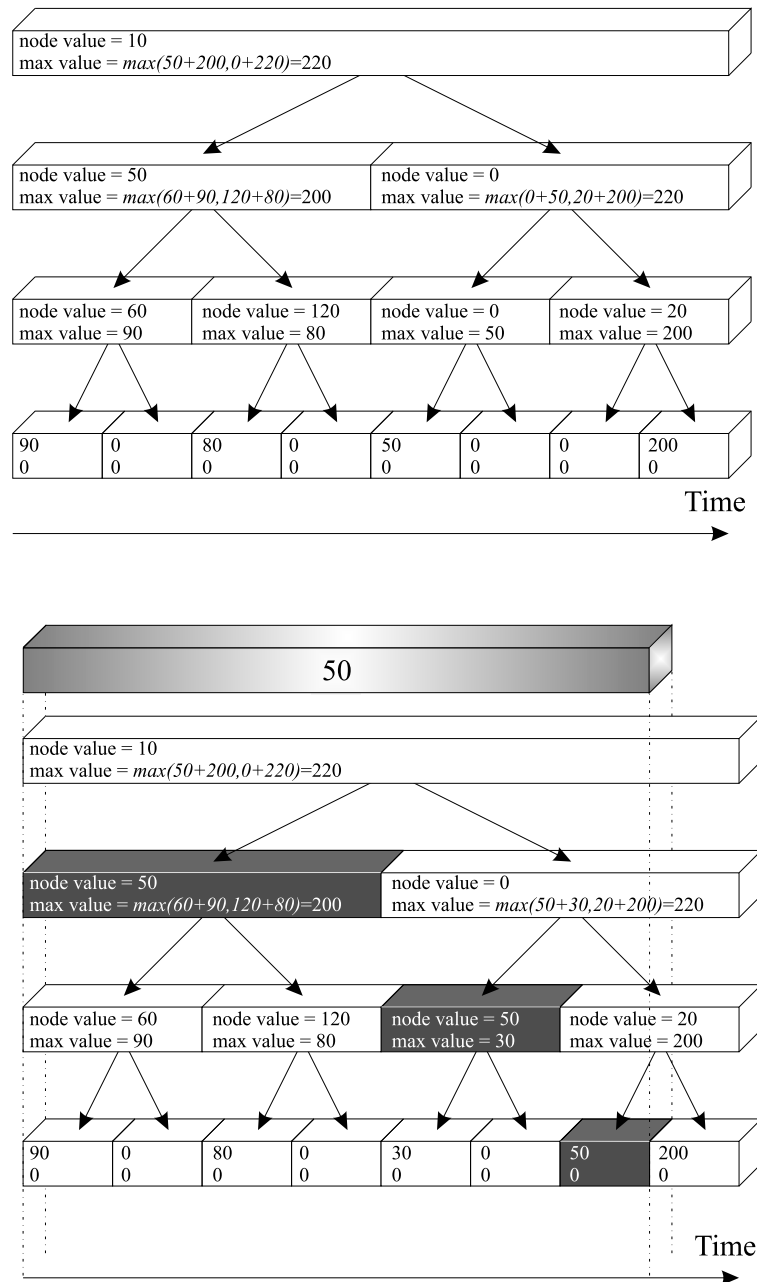


Figure 5.3: A segment tree before and after insertion

The details of the operations as well as some optimizations are found in [58]. All operations can be performed in constant time, i.e., they are independent of the number of reservations stored in the tree. The time depends, however, on the number of time slots and the length of the requested duration (see section 5.6).

5.5 Immediate admission control

Immediate reservations are open ended (i.e., they start immediately and can be held on to by clients as long as they like). Admission control for immediate reservations is performed by recording the aggregate bandwidth of all immediate reservations. In addition, admission control looks ahead (see figure 5.1) a short period of time into the advance reservation state to find out about resources needed for advance reservations in the near future [68]. Thus, an admission decision is taken by adding the requested delta to the aggregate of current immediate reservations. Then, the lookahead interval is checked for sufficient resources so that both advance reservations and immediate reservations can be served. This operation is similar to performing advance admission for the lookahead duration, with the exception that granted bandwidth is added to the aggregate immediate reservations instead of being recorded as an advance reservation.

5.6 Evaluation of single-hop admission control speed

We have performed experiments by using both the binary search tree (section 5.4.2) and the segment tree (section 5.4.3). Figure 5.4 shows the results when performing admission control over a single link where there are up to approximately 800000 different reservations spread over 30 days. The following parameters are used in this example:

- the link speed is 100 Mb/s,
- each request asks for a bandwidth of exactly 64 kb/s (we have chosen a small value to obtain a large number of reservations),
- requested durations are uniformly distributed in [20..180] minutes.
- requested starting times are uniformly distributed over 30 days (i.e., 43200 minutes)
- time is slotted with a granularity of five minutes

Each point plotted in figure 5.4 is obtained by averaging over 1000 admission requests. In this case, it is clear that advance admission control on average can be performed in less than $10\mu s$ when using the search tree implementation and less than $7\mu s$ when using the segment tree implementation. These results are for admission control over a single link (including decision time and storage time). When a reservation involves several hops, the corresponding time will be spent for admission control over each hop.

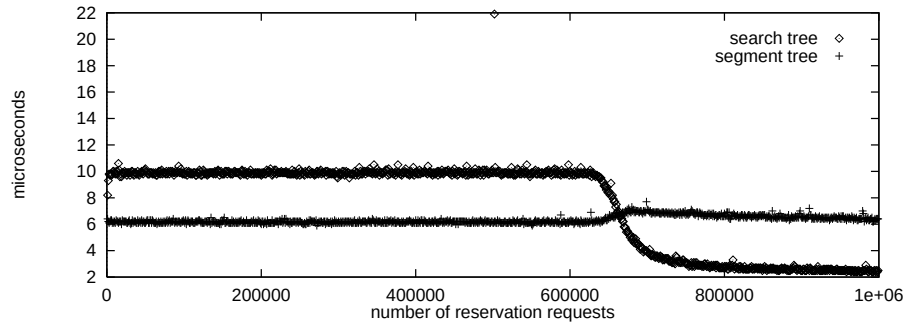


Figure 5.4: Admission decision speed for a single link

5.6.1 Dependence on the number of rejections

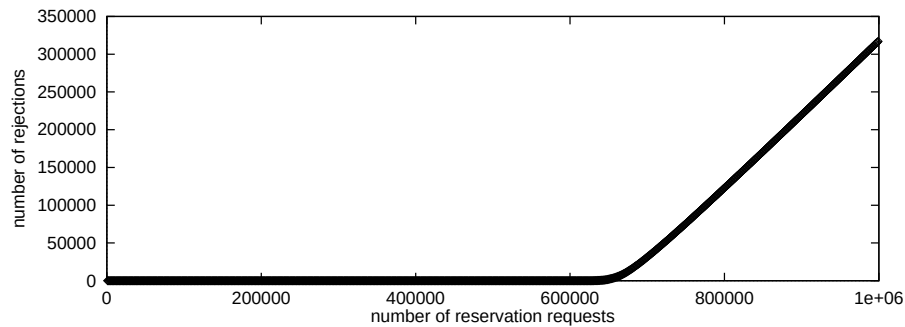


Figure 5.5: Rejections for a single link

In figure 5.5, we show that resources are reserved up to the point where we have rejections. The knee shows at which number of reservations we start getting rejections. From figure 5.4, it is clear that admission control speed changes when there is a large number of rejections. In the binary search tree, admission decisions are taken before the corresponding reservations are stored. Also, admission decisions end as soon as one case of insufficient resources is found. Therefore, the admission control time goes down when there are rejections. In the segment tree, traversed segments are updated as an admission decision proceeds along a branch of the tree. If there is a rejection, the reservation state has to be restored. Therefore, execution time increases when there are rejections. The major difference is explained by the way we have designed our operations. There is a choice between optimizing either for fast acceptance or for fast rejection.

5.6.2 Dependence on the number of reservations

It is somewhat surprising that the binary search tree appears to have an almost constant execution time (figure 5.4). Intuitively, the execution time should have an $O(\lg(n))$ component for finding the starting time and an $O(m)$ component for admission control through the requested duration. However, these effects are visible only in the range of very few reservations (e.g., < 10000). As the number of reservations grow large, the data structure grows to a maximum number of nodes. In this case there are never more than 8640 nodes in the tree. This is because there are 8640 slots of five minutes in a period of 30 days. Once the tree has reached the maximum number of nodes, each set of admission requests will on average take the same time even if the total number of reservations increase. Thus, aggregation of reservation state results in both data structures having execution times that are independent of the number of reservations stored (when the number of reservations is reasonably large).

5.6.3 Dependence on insertion order

The segment tree is always fully balanced independently of the insertion order. The search tree, on the other hand, may get unbalanced if the requested durations are ordered. In the worst case (i.e., when insertions are totally ordered), all nodes in the structure must be traversed. In our experiments here presented, the requested durations are generated randomly with equal probability over time. Thus, the search tree is well balanced. If requested durations sweep over time (e.g., if most reservations are made one day ahead while current time moves forward), the search tree will tend to be filled from the left hand side to the right hand side. Consequently, the admission decisions will take longer time.

5.6.4 Dependence on the reservable time frame

The search time depends on the total number of slots (i.e., nodes) stored in the data structure. This number is determined by the reservable time frame, divided by the slot size. Using a slot size of five minutes, figure 5.6 shows how the search time depends on the reservable time frame. In this graph, the dependence of the reservable time frame is isolated by keeping the distribution of durations fixed (in $[20..180]$ minutes). We measure the average admission decision time over granted requests only.

For both data structures, the search time is in $O(\lg(n))$, where n is the total number of slots. However, the constants for admission control over the given durations are smaller for the segment tree.

5.6.5 Dependence on the length of requested durations

The number of time-slots spanned by a requested duration is directly proportional to the length of that duration. In the search tree implementation, there is a linear search through the time slots spanned by the duration. This results in admission control time

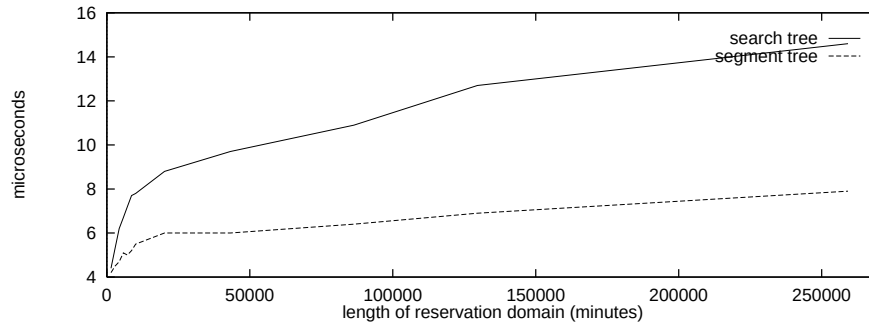


Figure 5.6: Decision time and reservable time frame

for the search tree being directly proportional to the length of the requested duration, as verified in figure 5.7.

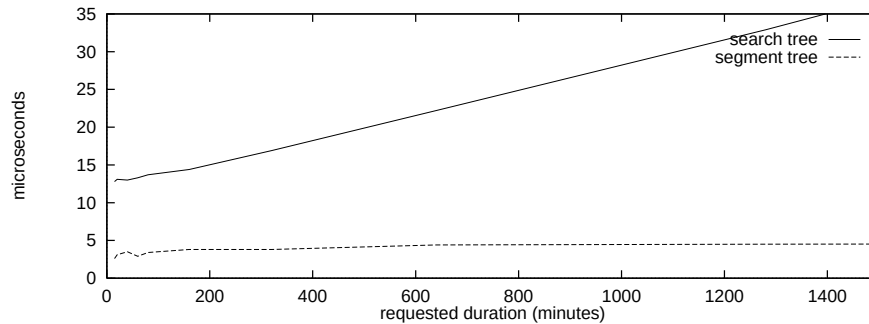


Figure 5.7: Decision time and length of requested durations

The segment tree shows a logarithmic curve (figure 5.7). The reason for this is that longer durations are more likely to be split at several levels of the tree (see section 5.4.3). There is some computational cost associated with each split. The worst case is that each end of the falling block is split exactly once at each level of the tree (where the top level is excluded). Thus the worst case is $2 * (d - 1)$ splits, where d is the depth of the tree. As described in section 5.4.3, the depth is in $O(\lg(n))$, where n is the total number of slots in tree.

5.6.6 Memory requirement

Our segment tree implementation has one leaf per time slot. The branching degree is configurable (individually for each level). Assuming the worst case, i.e., a binary tree, there is a total of $2n - 1$ nodes, where n is the number of time slots spanned. Each node contains one pointer to where the children are stored and four integers (start time, stop time, node value, max of children). Assuming four bytes per pointer and

four bytes per integer, each node requires 20 bytes. When spanning 30 days using 5 minute slots (i.e., when there is a total of 8640 time slots), the memory requirement adds up to $(2 * 8640 - 1) * 20 = 345580$ bytes. A higher branching degree decreases the memory requirement.

The search tree implementation requires one node per time slot in the worst case. In the current implementation, each node contains four pointers (left child, right child, next, previous) and four integers (time, aggregation count, aggregated bandwidth delta, bandwidth of left child). This means that each node takes 32 bytes. For 8640 time slots, this adds up to $8640 * 32 = 276480$ bytes. However, by allocating the pointers in a chunk and have one pointer to that chunk (as we did in the segment tree implementation), we would need only 20 bytes per node. Then, the total memory requirement for 8640 time slots would be $8640 * 20 = 172800$ bytes. Note also that the search tree requires less memory if there are few reservations.

5.6.7 Discussion

Our experiments show that the segment tree generally has better performance than the search tree. Especially, the segment tree is the best choice for large backbone links where there is a large number of requests with long durations, e.g., for reserving trunk bandwidth. Also, the segment tree is insensitive to the insertion order. The drawbacks of the segment tree are that it requires slotted time and that it uses more memory. Having slotted time seems natural for advance reservations and should not be a problem. The higher memory consumption for the segment tree might be hard to motivate for narrow bandwidth links where there can only be a few reservations at any time. In that case, the search tree implementation is more efficient.

The segment tree must be pre-configured for covering a certain period of time (e.g., by having a different tree for each month). The search tree, on the other hand, can continuously adapt as time proceeds (with the risk of being somewhat unbalanced).

5.7 Routing calculations

Our architecture requires that link-state routing is used in agent domains. The server on which the agent is running executes the routing protocol to obtain the topological database for the domain. Our prototype runs OSPF as implemented in the program *gated*.

The following basic functionality is part of OSPF [55]. Each OSPF node maintains an identical database describing the topology of the domain. From this database, a routing table is calculated by constructing a shortest-path tree. OSPF recalculates routes quickly in the face of topological changes, generating small amounts of routing protocol traffic. In an OSPF domain, each router generates its routing table from this graph by calculating a tree of shortest paths with the router itself as root. Obviously, the shortest-path tree depends on the router doing the calculation. The tree gives the entire path to any destination network or host in the domain. In routers, however, only the next hop to the destination is used in the forwarding process. As a result, packets sent from a node will follow the paths found in the shortest-path tree of that node.

An admission control agent must be able to calculate shortest-path trees starting at any router in the domain. For this, we use a modified Dijkstra's shortest path algorithm [16] (i.e., with a binary heap implementation for the priority queue). The complexity of this algorithm for computing a shortest-path tree from a given node is in $O(e * \lg(v))$, where e is the number of links (edges) and v is the number of routers (vertices). Input for the algorithm is obtained from the OSPF topological database.

5.8 Evaluation of multi-hop admission control speed

End-to-end admission control within a link-state routing domain involves finding the path between the end points and performing admission decisions over each hop in the path. Here, we compare the cost for on-demand route calculations (i.e., performed for each admission request) with the cost for accessing pre-calculated routes.

We evaluate the average time to process an admission request within the topology used at Luleå University of Technology (section 5.2.2). For the end-to-end admission requests generated, the average number of hops is 3.2. In these experiments, admission control includes both finding the shortest-path and performing admission decisions over each hop in the shortest path. In our implementation, admission decisions are performed for each router interface along the path, i.e., twice for each intermediate hop and once for each of the two end hops. A path of h hops therefore involves $2 * (h - 1)$ admission decisions. An average path of 3.2 hops, as in our experimental domain, involves 4.4 admission decisions.

5.8.1 On-demand route computation

Figure 5.8, shows the performance when routes are computed on-demand. For the 100 Mb/s links used in this topology, the time needed for admission decisions depends on which data structure we use. As shown in section 5.6, the time needed for admission decisions per-link in the segment tree is at most $7\mu s$. Thus, the average time spent for admission decisions over a path is $4.4 * 7 \approx 31\mu s$. The rest of the time (approximately $30\mu s$) is spent on shortest path first routing to find out which links are involved in the request.

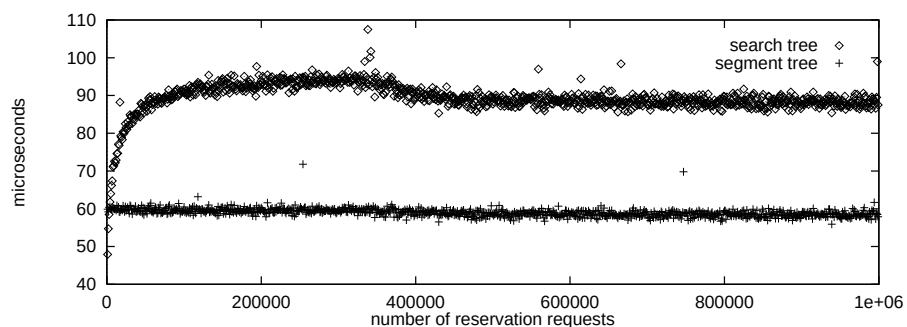


Figure 5.8: Admission decision speed within the LTU domain

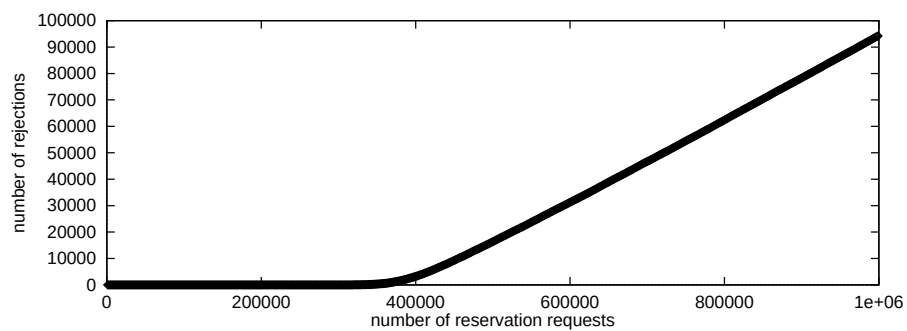


Figure 5.9: Admission rejections within the LTU domain

In figure 5.9, we show that resources are reserved up to the point where we have rejections. From figure 5.8, it is again clear that the admission control speed does not suffer from having a large number of rejections.

We have implemented multi-hop admission control using a one-pass mechanism with cleanup. We reserve resources hop-by-hop and if there is a rejection at one hop, we reset the reservations at all previous hops. Since rejections on average happens halfway to the destinations, a rejection means setting up reservations along half the way and then tearing down those reservations. This is about the same cost as setting up a successful reservation end-to-end. In fact, when there is an extreme load of requests the rejection is likely to happen earlier and the average admission control time becomes shorter. This method is robust and amenable to parallelization. A two-pass solution where we first check availability along the whole path before reserving would be faster when there are many rejections.

5.8.2 Pre-computed paths

One way of saving time for route calculations would be to cache the most recently used shortest-path trees. However, caching works best if there is locality in the reservation requests. As we currently have no real data indicating locality for resource reservations this is left for further studies. Another solution is to pre-compute all shortest path trees. Then, we can save most of the processing time for route calculations.

In figure 5.10, we find that admission control over 4.4 interfaces can be performed in approximately $40\mu s$ when routes are pre-computed. The admission decisions take $4.4 * 7 \approx 31\mu s$ and finding the pre-computed route takes approximately $9\mu s$.

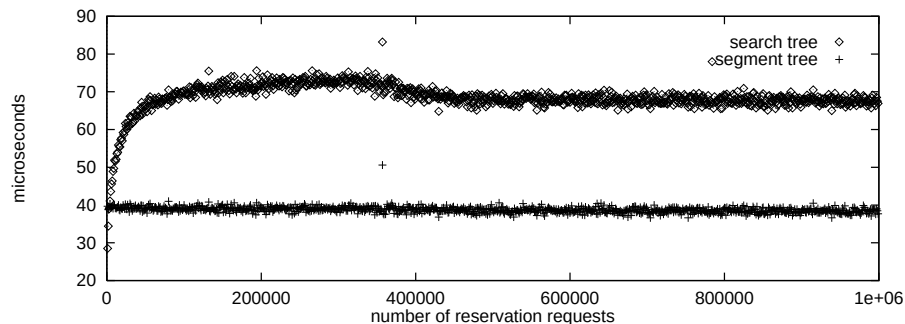


Figure 5.10: Admission speed using pre-computed paths within the LTU domain

5.8.3 Memory requirement

Storing all pre-computed shortest-path trees in our current implementation involves a total of n^2 nodes, where n is the number of routers in the domain. Each node in the shortest-path has two pointers to interface structures and one pointer to the

parent node in the tree. In our example domain with 15 routers, this adds up to $15^2 * 12 = 2700$ bytes for shortest-path trees.

The 15 routers in the domain have a total of 48 interfaces. For each interface, there is an instance of the reservation data structure. Using segment trees, each instance requires 345580 bytes when supporting reservations with five minutes granularity over 30 days. The total memory required for all 48 admission control data structures is therefore $48 * 345580 \approx 16200$ K.

By measuring the memory consumption of the prototype agent we obtained a maximum of 20400 K in this topology (this involves more information about interfaces, runtime stack, debug strings associated with all objects, etc).

5.9 The cost of request-reply transactions

In the experiments presented, admission requests are generated in the computer where the agent runs. In reality, an agent would receive reservation requests by using a reliable communication protocol. For this, we suggest using T/TCP [9, 72] which is an extension to TCP for handling short reliable transactions efficiently. In T/TCP, clients can establish and close a virtual connection in the same packet as the request is sent. Also, the server can close its part of the virtual connection as the reply is returned. Therefore, the total delay from sending an admission request until receiving the reply equals the round-trip time plus the processing time of the admission request in the server (provided no packets are lost).

To evaluate the performance of an agent, we focus on the protocol processing time in the server only (i.e., we do not consider processing time in the client or round-trip delay). When admission requests are received from the network, the total time for processing an admission request must include local T/TCP protocol processing time. We have measured the T/TCP protocol processing time on our machine by running a T/TCP-capable dummy-server that immediately returns a reply to each incoming request. By running the program *tcpdump* [43], comparing the arrival time of requests with the departure time of replies, we have found the protocol processing times to be on average between $100\mu s$ and $110\mu s$, for data sizes between 400 bytes and 1400 bytes.

Thus, with our current implementation of admission control, protocol processing time for admission requests is around twice as large as the time needed for performing end-to-end admission control in our example domain. However, the protocol processing time is independent of the size of the link-state routing domain, while admission control time is proportional to the path length. For clients making frequent admission requests it is more efficient to use a TCP connection for streaming several admission requests, especially if connection setup involves authentication.

5.10 Conclusion

We have evaluated two data structures for advance reservations, i.e., a segment tree and a binary search tree. The segment tree is faster and more indifferent to changing parameters such as average durations requested, insertion order, etc. The only drawback of the segment tree is that it requires more memory than the binary search tree. With the segment tree, an admission decision takes less than $7\mu s$ per hop when supporting reservations over a 30 day period with 5 minutes granularity.

We have also evaluated the time required for finding end-to-end paths in a link-state domain. Using on-demand route computations, the time is on average $30\mu s$ in our example domain. Using pre-computed routes, the time for finding a path in our example domain is on average $9\mu s$. Using pre-computed routes and segment trees for admission control over each link in a topology with 15 routers, 28 transition networks and 64 stub networks, admission control takes approximately $40\mu s$. Thus, in this topology an agent can process around 25 000 end-to-end admission requests per second. We have shown that protocol processing time in the server for receiving admission requests through T/TCP is substantial. Currently, the protocol processing time is more than $100\mu s$ per request.

The results show that it is feasible for an agent to manage per-link resource reservations and perform path-sensitive admission control in a link-state routing domain. A general purpose computer is powerful enough to run the agent software.

Bibliography

- [1] W. Almesberger, J.-Y. L. Boudec, and T. Ferrari. Scalable resource reservation for the Internet. In *Proc. of IEEE Conference Protocols for Multimedia Systems – Multimedia Networking (PROMS-MmNet)*, Santiago, Chile, November 1997. Technical Report 97/234, DI-EPFL, Lausanne, Switzerland.
- [2] ATM User-Network Interface (UNI) signalling specification, version 4.0. Technical specification, ATM Forum, July 1996.
- [3] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT). In *Proceedings of SIGCOMM*, pages 85–95, San Francisco, California, September 1993. ACM. also in *Computer Communication Review* 23 (4), Oct. 1992.
- [4] A. Banerjea and B. Mah. The real-time channel administration protocol. In *Proceedings 2nd International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'92)*, pages 160–170. Springer-Verlag, November 1992.
- [5] A. Banerjea, D. Ferrari, B. A. Mah, M. Moran, D. C. Verma, and H. Zhang. The tenet real-time protocol suite: Design, implementation, and experiences. Technical Report TR-94-059, University of California at Berkeley, Berkeley, California, November 1994. Also *IEEE/ACM Transactions on Networking*, vol. 4, n. 1, pp. 1-10, February 1996.
- [6] T. Barzilai, D. Kandlur, A. Mehra, and D. Saha. Design and implementation of a RSVP based quality-of-service architecture for the internet protocol suite. *IEEE Journal on Selected Areas in Communications*, 16(3):397–413, April 1998.
- [7] S. Berson, R. Lindell, and R. Braden. An architecture for advance reservations in the Internet. Technical report, USC Information Sciences Institute, July 1998.
- [8] S. Berson and S. Vincent. Aggregation of Internet service states. In *Proceedings of Sixth International Workshop on Quality of Service (IWQoS'98)*, Napa, California, May 1998.
- [9] B. Braden. T/TCP – TCP extensions for transactions functional specification. Request for Comments (Experimental) 1644, Internet Engineering Task Force, July 1994.

- [10] B. Braden and S. Casner. RBONE — resource administration for the MBONE. Presentation at 32nd IETF, Danvers, Massachusetts, April 1995.
- [11] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 functional specification. Request for Comments (Standards Track) 2205, Internet Engineering Task Force, September 1997.
- [12] L. Breslau and S. Shenker. Best-effort versus reservations: A simple comparative analysis. In *Proceedings of SIGCOMM*, pages 2–4, Vancouver, Canada, September 1998. ACM.
- [13] A. Campbell, C. Aurrecochea, and L. Hauw. A review of QoS architectures. In *Proceedings of 4th IFIP International Workshop on Quality of Service (IWQoS'96)*, Paris, France, March 1996.
- [14] D. D. Clark and W. Fang. Explicit allocation of best effort packet delivery service. Technical report, MIT Lab for Computer Science, Boston, Massachusetts, November 1997.
- [15] D. D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *Computer Communication Review*, 22(4):14–26, October 1992. *Proceedings of ACM SIGCOMM '92*.
- [16] T. H. Cormen, C. E. Leiserson, and R. R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1991.
- [17] S. Deering, C. Partridge, and D. Waitzman. Distance vector multicast routing protocol. Request for Comments (Experimental) 1075, Internet Engineering Task Force, November 1988.
- [18] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. An Architecture for Wide-Area Multicast Routing. *Computer Communication Review*, 24(4):126–135, October 1994. *Proceedings ACM SIGCOMM '94 Conference*.
- [19] M. Degermark, T. Köhler, S. Pink, and O. Schelén. Advance reservations for predictive service. In *Proceedings of NOSSDAV*, Lecture Notes in Computer Science, pages 3–14, Durham, New Hampshire, April 1995. Springer.
- [20] L. Delgrossi and L. Berger. Internet stream protocol version 2 (ST2) protocol specification - version ST2+. Request for Comments (Experimental) 1819, Internet Engineering Task Force, August 1995. (Obsoletes RFC1190).
- [21] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. *Journal of Internetworking: Research and Experience*, 1(1):3–26, January 1990.
- [22] Differentiated services (diffserv) working group. <http://www.ietf.org/html.charters/diffserv-charter.html>.

- [23] A. Eriksson. Resource reservation in a connectionless network. In *Proceedings of Performance of Information and Communications Systems (PICS'98)*, Lund, Sweden, May 1998.
- [24] A. Eriksson and C. Gehrmann. Robust and secure light-weight resource reservation for unicast ip traffic. In *Proceedings of Sixth International Workshop on Quality of Service (IWQoS'98)*, Napa, California, May 1998.
- [25] G. Fankhauser, B. Stiller, and B. Plattner. Arrow: A flexible architecture for an accounting and charging infrastructure in the next generation Internet. Technical report, ETH Zürich, Switzerland, October 1997. Presented at First Berlin Internet Economics Workshop, Berlin, Germany. October 24 - 25, 1997.
- [26] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. In *Proceedings of NOSSDAV*, Lecture Notes in Computer Science, pages 15–26, Durham, New Hampshire, April 1995. Springer.
- [27] D. Ferrari. Real-time communication in an internetwork. *Journal of High Speed Networks*, 1(1):79–103, 1992.
- [28] D. Ferrari. Multimedia network protocols: Where are we? *ACM/Springer Journal of Multimedia Systems*, 4(6):299–304, 1996.
- [29] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. *ACM/Springer Journal of Multimedia Systems*, 5(3):187–198, 1997.
- [30] V. Firoiu, J. Kurose, and D. Towsley. Efficient admission control for EDF schedulers. In *Proceedings of IEEE Infocom*, San Francisco, California, March 1996.
- [31] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. Technical report, Lawrence Berkeley Laboratory (Under submission to IEEE/ACM Transactions on Networking), February 1998.
- [32] FreeBSD project home page. <http://www.freebsd.org/>.
- [33] E. Fulp, M. Ott, D. Reininger, and D. Reeves. Paying for QoS: An optimal distributed algorithm for pricing network resources. In *Proceedings of Sixth International Workshop on Quality of Service (IWQoS'98)*, Napa, California, May 1998.
- [34] M. W. Garrett and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. *Computer Communication Review*, 24(4):269–280, October 1994. *Proceedings ACM SIGCOMM '94 Conference*.
- [35] GateD consortium home page. <http://www.gated.merit.edu/>.
- [36] R. Guerin, C. Partridge, and S. Shenker. Specification of guaranteed quality of service. Request for Comments (Standards Track) 2212, Internet Engineering Task Force, October 1997.

- [37] A. Gupta, W. Howe, M. Moran, and Q. Nguyen. Resource sharing for multi-party real-time communication. In *Proceedings of IEEE Infocom*, Boston, Massachusetts, April 1995.
- [38] D. Harrington, R. Presuhn, and B. Wijnen. An architecture for describing SNMP management frameworks. Request for Comments (Standards Track) 2271, Internet Engineering Task Force, January 1998. (Obsoletes RFC2261).
- [39] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an autonomous system (AS). Request for Comments (Best Current Practice) 1930, Internet Engineering Task Force, April 1996.
- [40] S. Herzog. RSVP extensions for policy control. Internet Draft, Internet Engineering Task Force, March 1998. Work in progress.
- [41] Integrated services (intserv) working group. <http://www.ietf.org/html.charters/intserv-charter.html>.
- [42] Isode home page. <http://www.isode.com/>.
- [43] V. Jacobson, C. Leres, and S. McCanne. tcpdump, the protocol packet capture and dumper program. <ftp://ee.lbl.gov/>.
- [44] S. Jamin, D. Clark, S. Shenker, and L. Zhang. Admission control algorithm for predictive real-time service. In *Proceedings 3rd International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'93)*, pages 349–356, Berlin, November 1993. Springer-Verlag. Workshop details: Nov. 12–13, 1992, La Jolla, Calif.
- [45] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated services packet networks. *Computer Communication Review*, 25(4):2–13, October 1995. *Proceedings of ACM SIGCOMM '95*. Also, *IEEE/ACM Transactions on Networking*, 5(1):56–70, February 1997.
- [46] M. Karsten, J. Schmitt, L. Wolf, and R. Steinmetz. An embedded charging approach for RSVP. In *Proceedings of Sixth International Workshop on Quality of Service (IWQoS'98)*, Napa, California, May 1998.
- [47] S. Keshav and R. Sharma. Achieving quality of service through network performance management. In *Proceedings of NOSSDAV*, Lecture Notes in Computer Science, Cambridge, United Kingdom, July 1998. Springer.
- [48] K. Kilkki and J. Ruutu. Simple integrated media access - an Internet service based on priorities. In *6th International Conference on Telecommunication Systems*, Nashville, TN., March 1998.
- [49] J. F. Kurose. Open issues and challenges in providing quality of service guarantees in high-speed networks. *Computer Communication Review*, 23(1):6–15, January 1993.

- [50] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic. *Computer Communication Review*, 23(4):183–193, October 1993. *Proceedings ACM SIGCOMM '94 Conference*.
- [51] I. W. M. Handley, J. Crowcroft. Hierarchical protocol independent multicast (HPIM). Technical report, University of Sussex, October 1995.
- [52] A. Mankin, F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, and L. Zhang. Resource reservation protocol (RSVP) – version 1 applicability statement some guidelines on deployment. Request for Comments (Informational) 2208, Internet Engineering Task Force, September 1997.
- [53] D. J. Mitzel, D. Estrin, S. Shenker, and L. Zhang. A study of reservation dynamics in integrated services packet networks. In *Proceedings of IEEE Infocom*, San Francisco, California, March 1996.
- [54] M. Mowbray, G. Karlsson, and T. Köhler. Capacity reservation for multimedia traffics. *Distributed Systems Engineering*, 5:12–18, 1998.
- [55] J. Moy. OSPF version 2. Request for Comments (Standards Track) 2178, Internet Engineering Task Force, July 1997. (Obsoletes RFC1583).
- [56] K. Nahrstedt and J. M. Smith. The QoS broker. *IEEE Multimedia Magazine*, 2(1):53–67, Spring 1995.
- [57] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the Internet. Internet draft, Bay Networks, LBNL and UCLA, November 1997.
- [58] A. Nilsson, C. Jingsen, and S. Carlsson. An efficient data structure for advance bandwidth reservation on the Internet. Technical report, CSEE, Luleå University of Technology, 1998. Work in progress. <http://www.sm.luth.se/~andreas>.
- [59] P. Pan and H. Schulzrinne. YESSIR: A simple reservation mechanism for the Internet. In *Proceedings of NOSSDAV*, Lecture Notes in Computer Science, Cambridge, United Kingdom, July 1998. Springer.
- [60] C. Partridge and S. Pink. An implementation of the revised Internet Stream Protocol. *Journal of Internetworking: Research and Experience*, 3(1):27–54, March 1992.
- [61] V. Paxson. End-to-end routing behavior in the Internet. In *Proceedings of SIGCOMM*, pages 25–38, Palo Alto, California, August 1996. ACM.
- [62] V. Paxson and S. Floyd. Wide-area traffic: the failure of Poisson modeling. *Computer Communication Review*, 24(4):257–268, October 1994. *Proceedings ACM SIGCOMM '94 Conference*.

- [63] Broadband Integrated Service Digital Network (B-ISDN) Digital Subscriber Signaling System no.2 (DSS2) User Network Interface Layer 3 Specification for Basic Call/Connection Control. ITU-T Recommendation Q.2931, International Telecommunication Union, Geneva, 1994.
- [64] W. Reinhardt. Advance reservation of network resources for multimedia applications. In *2nd International Workshop on Advanced Teleservices and High Speed Communication Architectures (IWACA'94)*, Heidelberg, Germany, October 1994.
- [65] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). Request for Comments (Standards Track) 1771, Internet Engineering Task Force, March 1995. (Obsoletes RFC1654).
- [66] Resource reservation setup protocol (RSVP) working group.
<http://www.ietf.org/html.charters/rsvp-charter.html>.
- [67] O. Schelén and S. Pink. An agent-based architecture for advance reservations. In *IEEE 22nd Annual Conference on Computer Networks (LCN'97)*, Minneapolis, Minnesota, November 1997.
- [68] O. Schelén and S. Pink. Sharing resources through advance reservation agents. In *Proceedings of IFIP Fifth International Workshop on Quality of Service (IWQoS'97)*, New York, May 1997. Also: Resource Sharing in Advance Reservation Agents. To appear in Journal of High Speed Networks, Special issue on Multimedia Networks in 1998.
- [69] O. Schelén and S. Pink. Aggregating resource reservations over multiple routing domains. In *Proceedings of IFIP Sixth International Workshop on Quality of Service (IWQoS'98)*, Napa Valley, California, May 1998.
- [70] O. Schelén and S. Pink. Resource reservation agents in the Internet. In *Proceedings of NOSSDAV*, Lecture Notes in Computer Science, Cambridge, United Kingdom, July 1998. Springer.
- [71] S. Shenker and L. Breslau. Two issues in reservation establishment. In *Proceedings of SIGCOMM*, Cambridge, Massachusetts, September 1995.
- [72] W. R. Stevens. *TCP/IP Illustrated, Volume 3*. Addison-Wesley, 1996.
- [73] B. Stiller. A survey of UNI signaling systems and protocols for ATM networks. *Computer Communication Review*, 25(2), April 1995.
- [74] I. Stoica and H. Zhang. LIRA: An approach for service differentiation in the Internet. In *Proceedings of NOSSDAV*, Lecture Notes in Computer Science, Cambridge, United Kingdom, July 1998. Springer.
- [75] A. Terzis, L. Zhang, and E. N. Hahne. Reservations for aggregate traffic: Experiences from an RSVP tunnels implementation. In *Proceedings of Sixth International Workshop on Quality of Service (IWQoS'98)*, Napa, California, May 1998.

- [76] D. Wischnik and A. Greenberg. Admission control for booking ahead shared resources. In *Proceedings of IEEE Infocom*, San Francisco, California, March 1998.
- [77] L. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller, and H. Wittig. Issues of reserving resources in advance. In *Proceedings of NOSSDAV*, Lecture Notes in Computer Science, pages 27–37, Durham, New Hampshire, April 1995. Springer.
- [78] J. Wroclawski. Specification of the controlled-load network element service. Request for Comments (Standards Track) 2211, Internet Engineering Task Force, October 1997.
- [79] J. Wroclawski. The use of RSVP with IETF integrated services. Request for Comments (Standards Track) 2210, Internet Engineering Task Force, October 1997.
- [80] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource ReSerVation Protocol. *IEEE Network Magazine*, 7(5):8–18, September 1993.
- [81] Z. Zhang, C. Sanchez, B. Salkewicz, and E. Crawley. Quality of service extensions to ospf. Internet draft, Internet Engineering Task Force, September 1997.