Advance Reservations of Bandwidth in Computer Networks

vorgelegt von
Diplom-Informatiker
Lars-Olof Burchard
aus Berlin

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Professor Dr.-Ing. A. Wolisz
Berichter:	Professor Dr. H.-U. Heiß
Berichter:	Professor Dr. L. Wolf

Tag der wissenschaftliche Aussprache: 14. Juli 2004

Berlin 2004
D 83

Abstract

In this thesis, the impact of using advance reservations of bandwidth in a computer network on the performance for both clients and operators of the network is examined. Based on an architecture that uses *multi-protocol label switching* (MPLS) controlled by bandwidth brokers, a number of services that - compared to todays best-effort or immediate reservation networks - provide an enhanced functionality for clients were developed. These services allow clients to specify requests in a less stringent way than currently necessary, for example, it is possible to define only the amount of data to be transmitted between two network endpoints and the management system then determines suitable transmission parameters such as start and stop time and transmission rate. This functionality provides reliable feedback to clients and can serve as a foundation for providing service-level agreements, e.g., guaranteeing deadlines for the transmission of a certain amount of data.

The additional services can also be used by network operators to improve the overall utilization of the network. In addition, the various opportunities of using the additional temporal dimension of the advance reservation service are suitable to improve the network performance. It can be shown that the amount of blocked requests and bandwidth can be considerably decreased making use of both services and the additional information available in the given environment.

Besides the achievable throughout and amount of admitted requests, the term *performance* in the context of advance reservation systems also covers other aspects such as failure recovery strategies and the processing time required by the network management system. In the thesis, several strategies to be applied in case of link failures are outlined and examined with respect to their applicability and achievable performance. For example, it can be shown that it is worthwhile to consider not only flows which are active at the time a failure occurs but also to take inactive but already admitted flows into account in order to achieve the best possible performance.

In addition to failure recovery, also the processing speed of the management system is of importance. For that purpose, in particular the data structures used to store the current and future network status need to be examined since they dominate the processing time of the management system. Two data structures, arrays and a tree which was especially designed for this purpose were examined, showing that arrays are superior with respect to processing speed and memory consumption in almost any environment.

Acknowledgments

I wish to thank especially my supervisor Professor Dr. Hans-Ulrich Heiß, who gave me the opportunity to write this thesis, for the constant support and encouragement throughout the last years. In particular, I appreciated the freedom and independence I experienced during the work in his research group.

I am also very grateful to Professor Dr. Lars Wolf, TU Braunschweig, for serving as a reviewer for this thesis.

Special thanks to my colleagues Barry Linnert and Arthur Lochstampfer for their questions, answers, the great cooperation, and the opportunity to discuss with them many of the issues raised in this thesis. I enjoyed working with them. Furthermore, I wish to thank Marc Droste-Franke for the interesting discussions we had about failure recovery.

Contents

List of Figures

Chapter 1

Introduction

In many different environments, in order to allocate resources, the allocations are made a long time before the resources are actually required. Examples are flight or hotel booking, as well as seat reservations in cinemas or theaters. This very natural way of reservation guarantees that the requested resources are available when they are needed, provided that the reservation is made sufficiently early. Besides this advantage for the prospective user of a given resource, reservations made in advance are also attractive for the owner or manager of resources since they permit of a better planning and scaling of their resources such that a sufficient amount of resources is available when and where needed most. An example is the major German railway company which recently introduced a new pricing scheme, that rewards customers buying tickets at least 3 days before their voyage while those buying tickets immediately before the journey were punished with higher prices. This means, reservations in advance were not only allowed but also forced in some sense. One of the reasons was that such reservations in advance allow the management to more accurately predict the requested amount of coaches and thus a better resource utilization. Although it must be mentioned that this new pricing scheme was rather unpopular among the customers, the example shows that also providers of a service have a strong motivation to implement mechanisms that allow customers to reserve in advance.

In general, two basic types of reservations can be distinguished: reservations that are made in a just-in-time manner directly before the resources are needed (*immediate reservations*) and those that are issued a longer period of time (e.g., minutes, days, or even months) before the resources are actually required (*advance reservations*). While advance reservations dominate in "real" world resource allocations and are also used in some fields of computer science, such as allocations on large-scale cluster and parallel computers, in the area of computer networks this reservation type did not gain significant dissemination and did not even attract much attention of the research community.

The reasons are plentiful, one of the major drawbacks is the lack of control over the network in the common Internet environment which is a result of its central design property as self-organizing network. However, with recent developments in network technology such as the introduction of the *multiprotocol label switching* (MPLS) technology, which allows a much higher degree of control over the network and its components, the situation changed. Furthermore, with the evolving *grid computing* efforts to allow connecting resources on a potentially

global scale and provide functionality for collaboration and co-allocations of very different resource types over a network, advance reservations may become a useful way to ensure the availability of communication lines between those different resource types scattered across the globe. In contrast, data intensive high-performance e-science applications, e.g., in the context of grid computing, require large amounts of bandwidth for the transmission of data. For such applications, the lack of quality-of-service (QoS) availability poses a problem which is addressed by specialized network infrastructures such as the TransLight *LambdaGrid* [DdLM+03] where one wavelength of a fiber is dedicated to a grid environment. In particular such networks should be equipped with a management system that provides not only advance reservations as required in the grid environment, but furthermore allows to implement QoS mechanisms that provide more than the current, relatively simple, QoS functionality, i.e., bandwidth reservation or delay constraints. Future networking applications, e.g., in the field of grid computing, demand also for a more general notion of QoS, which includes definitions of the temporal constraints of a job, e.g., a deadline, or guarantees in case of failures. For this purpose, the usage of service level agreements (SLA) is an approach to negotiate such enhanced QoS parameters. As outlined in [BHK+04], SLAs for computing applications are an important requirement which allow to negotiate transmission rates for large bulk transfers or the transmission times. The content of SLAs also includes guarantees such as deadlines for computations and the respective network transmissions. These functionalities can only be implemented on top of an advance reservation system, the current immediate reservation mechanisms do not suffice for this purpose.

The nature of resource reservations in such systems differs largely from those of current requirements for instantaneous calls such as video streaming of short clips which have many similarities with phone calls, e.g., in the sense that their duration is usually not known in advance and hence cannot not be reliably planned. This obstructs the application of advance reservations in such cases. In contrast, allocations in grid computing environments usually require large amounts of resources which must be allocated in advance. In these cases, also the network connections must be guaranteed and furthermore the start and duration of the allocations of the different resources are known.

Besides the application in the field of grid computing, advance reservations provide additional opportunities for users and operators of computer networks in various aspects as expressed by Degermark, Pink, Köhler, and Schelen in [DKPS95]:

> "When resources are plentiful, not even immediate reservations may be necessary but when resource are scarce enough to justify reservations, it makes sense to be able to make them in advance."

In the following chapters, the properties of advance reservations in computer networks will be examined and it will be shown that the above cited statement is correct from very different perspectives of clients (users, customers) and network operators. The previous citation with its general scope can be expressed even more restrictive: in some cases it is even necessary to reserve in advance, for example, when co-allocation of different resources is required as in the field of grid computing.

In general, the scarcity of resources is not a realistic assumption for a variety of applications in computer networks. For example, access to web pages does not require large amounts of network bandwidth and for such applications, todays network capacities are sufficient. In addition to the large amount of spare network capacity, technological measures such as content distribution networks help to avoid bottlenecks for these applications. However, increasing network traffic and new, additional applications such as grid computing with potentially huge amounts of data to be transmitted across a global network demand for QoS, which is understood more generally than just as a definition of bandwidth availability on network links.

In this context, the design and management of network services needs to consider the applications that are run on the network. In [ABW04], it is stated that it is "imperative that we go back to the future [...] where both network and application research were undertaken simultaneously on the same infrastructure" in order to study and understand the necessity for not only traditional, immediate bandwidth reservation-centered quality-of-service mechanisms, but also "security, reliability, resilience, virtual environments, and network management and monitoring". Generally applicable for a variety of applications from on-line gaming to video conferencing and grid computing, advance reservations may serve as one building block for a future resource management platform. In sight of the convergence of various compute resources into a single, global, multi-purpose infrastructure with support for various user-oriented services, the advance reservation service supports in particular these new application environments that require more functionality than the current network infrastructures can offer. An advance reservation service, independent from its technological basis (MPLS or any other suitable technology), is one important step towards achieving this goal.

So far, only a small subset of topics related to advance reservations have been investigated. One of the reasons is the lack of a suitable management architecture that can be easily applied in a network without changing the whole soft- and hardware. An opportunity to overcome this problem is the MPLS standard that allows to influence the network behavior with nearly arbitrary granularity. Based on this technology, the implementation of an advance reservation service in a computer network such as the Internet may eventually become reality.

Using the technological foundation of MPLS, the different aspects of performance of an advance reservation service will be examined in this thesis. For this purpose, a network management system (*bandwidth broker*) was implemented and used as simulation environment for the algorithms and mechanisms needed to offer an advance reservation service in a computer network. This includes a variety of different services and mechanisms that have the potential to influence the performance of the network for customers and operators. For a customer, performance relates to those properties of a network that deal with the individual requests. For example, besides the probability that a request is accepted, which is of major interest for a customer, the speed of the management system is an important factor in this context. Likewise, the behavior of the network in case of failures is essential for the perceived performance of a network. In the ideal case, a customer does not even notice when a failure occurs. For the operator, the main performance aspect is the profit generated by the network. This is directly related to the utilization of the network which is intended to be maximized, but also other factors count, such as customers satisfaction or

the costs of a management infrastructure. For obvious reasons, in this thesis only those performance aspects are considered that can be measured without making assumptions about the business model of a network operator. Thus, the focus concerning the performance as perceived by operators is on the amount of traffic that can be accommodated by the network and the amount of flows that can be admitted.

The most interesting question concerning the performance of an advance reservation network is how to use the additionally available information about the future status of a network to improve the performance. On the one hand, this allows to define *additional services* for customers that cannot be implemented in an immediate reservation environment, such as searching for the first feasible transmission interval for a given amount of data. Network operators on the other hand need to adapt not only their management infrastructure in order to keep information about allocations in the future but also need to implement new functionality in order to fulfill the basic management tasks, i.e., admission control and routing, and to implement the additional services. Moreover, the available information about the future can be used by the operator to improve the performance of the network. This impacts the routing strategy where the opportunity arises to switch traffic flows onto different paths during their lifetime. When a given amount of data must be transmitted between two network end points, and the network management is requested by the client to determine suitable transmission parameters, this degree of freedom can be used to optimize the network performance by scheduling such transmissions in a way which increases the overall network performance.

The network performance is closely related to the *fault tolerance* properties, i.e., the question of how to react in case of link or router failures. Especially in the field of advance reservations this is an important issue since reservations can be made long before they are actually required. Although many strategies for dealing with link failures in immediate reservation scenarios exist, so far this aspect was not considered for advance reservation environments. In chapter 6, strategies for dealing with link failures are presented which also take the available information about the current and future status of the network into account. Their basic feature is to reroute flows in advance when it is likely that they will be affected by a link failure in the future. This pro-active behavior of the network allows the bandwidth broker to react to link failures as soon as possible even when the impact of a link failure did not yet become visible in terms of preemption or termination of flows.

Another important performance aspect is the speed of the admission control process in an advance reservation environment which is a critical aspect for end-users. Because of the large amount of status information to keep, it is necessary to implement the reservation and allocation mechanisms such that the response times of the reservation system is minimized. This means, each client's request can be processed without delays in an on-line fashion, delivering the response with the admission decision as soon as possible after the request was made. In order to efficiently implement this admission control process, data structures are required which provide fast access to the information about the future while supporting the additional services implemented in the advance reservation environment, e.g., the search for suitable transmission parameters for a particular request. In the prototype bandwidth broker implementation, about 50% of the processing time of each request is spent in these data structures. In

contrast to previous studies which used specially designed trees for this purpose, it will be shown that arrays are superior in many environments concerning both admission speed and memory consumption. Furthermore, arrays can be implemented much easier and better support the additional services offered in the advance reservation environment.

In order to categorize the different mechanisms discussed in this thesis, two main approaches can be distinguished: the first approach uses the properties that are introduced by the needs of the clients and applications using an advance reservation network service (application-oriented, *services*). This category covers, for example, the search for suitable transmission intervals and transmission rates. On the other hand, a number of techniques, which constitute the second type of approach, can and in some cases must be implemented transparently for the applications (operator-oriented, *management operations*). Examples for such approaches are the routing strategy and the selection of failure recovery mechanisms. These mechanisms are developed in this thesis and examined with respect to their implications for the performance of the overall advance reservation service.

The remainder of this document is organized as follows: firstly, in chapter 2 the foundations of this work are described. This includes an overview and a comparison of existing approaches to implement advance reservations, showing that especially the performance aspects of advance reservations have so far not been exhaustively examined. The discussion of possible application environments show that a number of applications benefit from or even require advance reservations and additional services that are not implementable in immediate reservation networks.

In chapter 3, the opportunities to implement these services that go beyond those available in immediate reservation networks are presented, such as the search for suitable transmission intervals for a large amount of bulk data. In addition, various opportunities to improve the performance of the network using the properties of advance reservations, i.e., the availability of temporal information about duration, start, or stop times of requests, are discussed. Based on this foundation, the design and components of the reservation framework is outlined. The architecture is based on domain-central bandwidth brokers which handle requests issued by clients and set up the network components accordingly. The impact of advance reservations on the overall performance of a computer network is discussed in chapter 4. The main drawback of this reservation type is a fragmentation of the available resources. Although this influences the overall performance of the network, the benefits for individual clients of a network, i.e., additional services and an improved admission probability, remain. The actual extend of the possible performance degradation is evaluated using extensive simulations with different network topologies. It can be shown that mainly the amount of flows that can be accommodated by the network is affected while the utilization remains less influenced.

Based on these results, in chapter 5 the basic techniques to deal with advance reservation requests in a computer network are presented. This includes the routing algorithms which can be derived from those used in immediate reservation QoS routing. In addition to these basic routing strategies, the exploitation of the additional temporal information in the advance reservation environment is of main interest. In particular, the opportunity to deal with requests that do not specify fixed parameters such as start and stop time but allow (and some-

times even request) the management system to determine suitable parameters for a particular transmission request can be used to significantly improve the overall performance of a network. This leads to a number of different scheduling strategies for those requests with variable parameters. The simulations with the different approaches show, that significant differences exist among them and hence, the actual strategy must be carefully chosen. Further opportunities to improve the network performance are transmissions using different paths during their lifetime or off-line optimizations based on solving multi-commodity flow problems. Using these techniques leads to a significant performance gain which can close the gap between immediate and advance reservations and sometimes results in better performance of the advance reservation scenario. This means, the performance degradation is avoidable when wisely using the properties of the advance reservation framework.

The impact of link or router failures is examined in chapter 6, where a concept to deal with link failures in the presence of advance reservations is presented. Data structures used to store link status information about future allocations are analyzed in chapter 7. It turns out, that arrays are well suited for this purpose although specially designed tree structures exist. However, these trees have significant disadvantages resulting from a distribution of the status information across several tree layers. The superiority of arrays can be shown using both analysis and measurements. After briefly discussing multi-domain issues of the bandwidth broker architecture, the thesis is concluded with some final remarks and a discussion of possible extensions in future work.

Chapter 2

Foundations

The advance reservation service architecture presented in this thesis is related to various issues of managing network and compute resources. The implementation of advance reservations requires a number of already existing architectures as building blocks for its application in a realistic environment. Since the bandwidth broker design described in the following chapters concentrates on functionalities on the flow level, it is important to connect these functionalities to lower layer implementations, e.g., those supporting network QoS and hence, the discussion in the following sections includes an overview of the currently available techniques to support QoS in current IP networks.

In this chapter, the different approaches for implementing network QoS are described and examined with respect to their suitability to support advance reservations. Based on this examination, the approaches made in the past to implement advance reservations and the different aspects of this particular reservation type, as far as being subject of research so far, are presented. Furthermore, examples of actual applications are given that benefit from advance reservations or even require them. The discussions and examinations of advance reservations in previous work usually did not take the actual requirements of such applications into account, but only concentrated on the network layer and its properties. This gap will be closed with the implementations described in this thesis.

2.1 Quality-of-Service in Computer Networks

In order to implement an advance reservation service in a computer network, it is important to examine the available protocols, architectures, and methods that are available in order to establish network QoS The vast variety of developments in this area cannot be summarized and presented here, however a number of different architectures are worth mentioning, as they have been used to implement various advance reservation mechanisms in networks. Many of those techniques, such as queuing, QoS routing, signaling etc. have to be applied together in order to build a QoS framework that provides useful guarantees for an application. For example, in [PD00] many of those techniques are described in a much more detailed way than possible in this thesis.

Two major approaches for network QoS have been distinguished in the past,

integrated and differentiated services. Both techniques serve orthogonal purposes: while the integrated services (*IntServ*) approach is to treat each traffic flow individually and consequently to provide a certain level of end-to-end QoS for each individual flow, in contrast the differentiated services (*DiffServ*) approach is to offer a few service classes with different QoS parameters and to multiplex the traffic within the service classes without distinguishing individual flows.

2.1.1 Integrated Services

Within the IntServ framework, a number of *service classes*[1] were specified to meet the QoS requirements of applications. The most important are the *guaranteed* service [SPG97] and the *controlled load* service [Wro97a]. The former was designed to support applications which cannot afford a single packet to arrive "late", e.g., a video streaming application with the requirement that each packet arrives before its playout time. The introduction of the controlled load service was based on the observation, that applications of this class behave sufficiently well on a lightly loaded network. Thus, the intention of the controlled load service is to simulate a lightly loaded network for these applications.

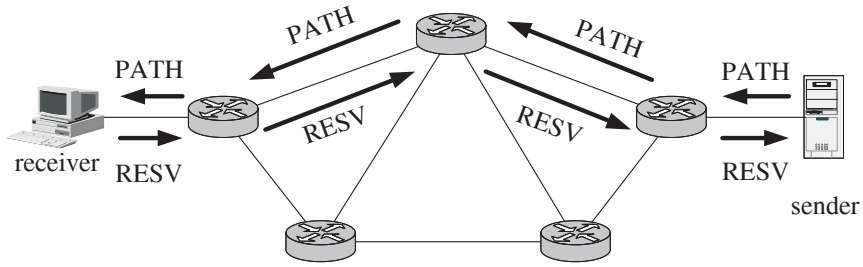


Figure 2.1: RSVP signaling between sender (*right*) and receiver (*left*). RESV messages follow the reverse route of the PATH messages and set up the reservation.

Often related to the IntServ approach is the usage of RSVP as reservation protocol [Wro97b, ZDE+93, BZB+97]. RSVP provides signaling functionality for individual flows using messages sent along the whole path from sender to the receiver and back. On each router, an RSVP daemon is responsible for processing those RSVP messages. For the basic functionality, two messages are important. The PATH message is sent from the sender (data source) to the receiver of the data stream (data sink) for which a certain QoS is required. The purpose of the PATH message is to determine the path to be taken by packets of the subsequent prioritized flow to the data sink. Each router on this path determines the reverse path for the corresponding packet, i.e., figures out which outgoing interface is used to send the reservation back to the source (see figure 2.1). When the PATH message arrives at the data sink, the receiver's RSVP daemon generates a RESV message with the actual QoS parameters which is sent back to the data source using the same path as the original PATH

[1] This does not mean, applications of a certain service class do not receive individual, i.e., fine-grained QoS. The notion of "service class" here has a different meaning than in the context of DiffServ.

message. Each router on the reverse path checks whether sufficient resources are available to satisfy the request. If so, the RESV message is forwarded using the link determined during when the corresponding PATH message was examined. In case, the resources do not suffice to provide the requested QoS, the router generates an error message which is returned to the data source.

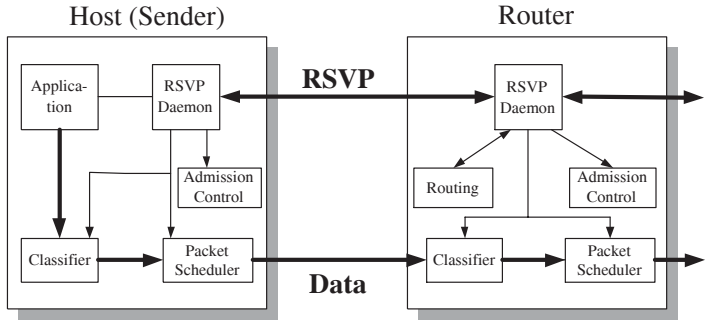


Figure 2.2: Architecture of RSVP enabled router and end-system. Daemons exchange RSVP signaling messages and determine the scheduling of the data packets [Bra97]. In addition to the specified components, each daemon needs a policy control module for authorization and authentication purposes.

When a reservation can be admitted, i.e., when the RESV message is received and successfully processed, each RSVP daemon on the path generates a session which keeps status information about the source and sink pair, i.e., IP address and port number, and the amount of resources to be reserved for the corresponding flow. In order to keep the session alive, RSVP follows a *soft-state* approach. This means, the session is not statically kept until explicitly removed by the entities involved, instead it is required to repeatedly - the suggested default is every 30 seconds [BZB+97] - sent RESV messages in order to keep the session alive. This is initiated by the RSVP daemons at the source and sink. In case, no such message is received by an RSVP daemon on the path, the corresponding session is removed. The main components of the RSVP architecture are depicted in figure 2.2. In addition, a policy control module is required for authorization and authentication purposes.

In order to accommodate for link failures and route updates in the network that require to reserve resources on an alternative path, the PATH messages are also sent periodically. In case of a failure, the refresh messages cannot be sent on the original path and hence lead to a removal of the reservation due to the timeout mechanism of RSVP. However, the PATH messages follow an alternative path and eventually establish a new reservation when sufficient resources are available on the alternative path.

One of the major drawbacks often related to IntServ and per-flow QoS is the lack of scalability due to the property of requiring status at each intermediate network node on the path between client and server. However, with efficient implementation and the usage of a lightweight protocol, in [PS00] it was shown that on an 700 MHz Pentium based PC, it was possible to handle around $10,000$ flow setups at a time or alternatively keep approximately $300,000$ IntServ sessions alive. Therefore, scalability is not that much of an issue today. Together with various extensions, RSVP has found a lot of different areas of application,

some of them also support advance reservations. In this context however, it will be shown in section 2.2 that the distributed approach of RSVP which does not rely on keeping central knowledge anywhere in the network has some other drawbacks that reduce its applicability to support advance reservations.

2.1.2 Differentiated Services

The DiffServ architectural model [NBBB98] itself does not provide end-to-end QoS, instead only guarantees for the performance of the service classes are given. For that purpose, a field in the IP header (*DiffServ Code Point*, DSCP) is used to mark IP packets as belonging to one of those service classes [BCD+98, Gro02]. Using this technology, it is possible to, e.g., provide a single service class for real-time traffic (Voice-over-IP, video streaming) and another service class for bulk transfer.

In order to define the behavior of routers in a DiffServ network, a set of *per-hop behaviors* (PHB) were specified which determine how a router treats packets (depending on its DSCP), i.e., traffic belonging to other than the best-effort class. Thus, DSCP specify a certain PHB. The *expedited forwarding* PHB [DCB+02] (EF) defines the requirements and properties for implementing low loss, low delay, and low jitter services, e.g., for real-time streaming. In contrast, the *assured forwarding* PHB (AF) [HBWW99] group defines a set of four services that provide different grades of assurance in the sense that packets in a service class are forwarded with high probability unless the used bandwidth does not exceed a certain level, previously notified or negotiated with the service provider. This can be used to define a bulk transfer service.

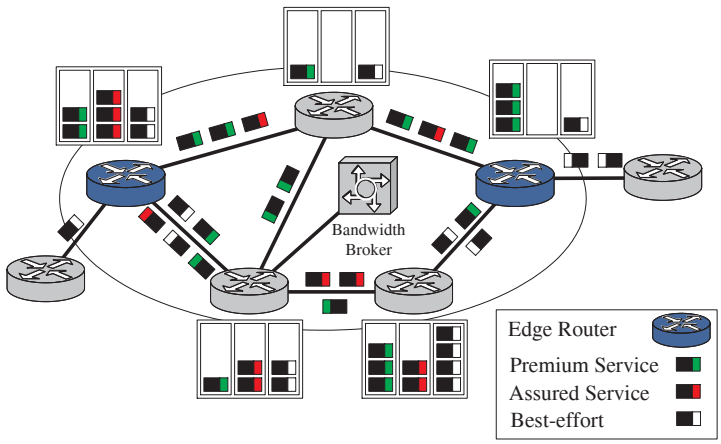


Figure 2.3: DiffServ network under the Two-Bit DiffServ model. Each router keeps three queues for the packet of different service classes. Edge routers perform the labeling of packets according to the rules set forth by the BB.

Since DiffServ itself does not provide end-to-end QoS guarantees, a mechanism is required in order to control access to the network and to manage the network's resources. For that purpose, the notion of *bandwidth brokers* (BB) was introduced which describes a centralized network management system. The concept of BB was described in [NJZ99], together with an architecture using a two-bit pattern of the DSCP to provide two service levels in addition to the

best-effort service. The purpose of this two-bit pattern is to mark traffic and to treat it differently, i.e., "better" than best-effort traffic. These two additional service classes defined in [NJZ99] were *assured service* and *premium service*. An example for a network under this model is depicted in figure 2.3.

The authors assume each *domain* - defined as a "region of trust" - within the Internet has its own bandwidth broker statically associated. This may include a hierarchy of bandwidth brokers supervised by a "root" BB at the top of the hierarchy. The purpose of the bandwidth broker is two-fold:

1. Management of each domains' higher priority traffic. This means identifying this traffic and configuring the routers within the domain to enable the higher priority service levels for marked packets. This requires also to control the edge routers which do the labeling of packets, i.e., adjusting the DSCP value.

2. Management of packets sent across domains. This requires inter-domain communication among different BB.

For that purpose, each bandwidth broker is equipped with a policy database which describes the traffic to be marked and the corresponding configuration of the routers.

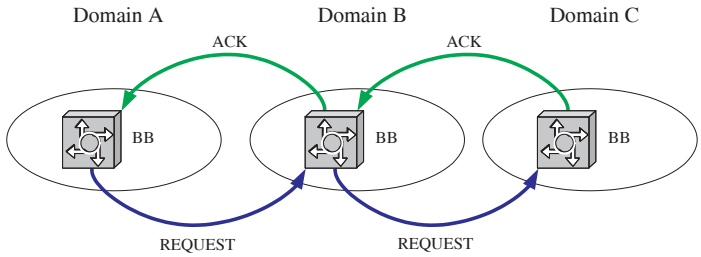


Figure 2.4: Domain structure and inter-domain communication of three BB. The communication follows a domain-per-domain concept. In this example, a request for a reservation from domain A to domain C is processed.

The scalability of the BB approach was addressed by assuming that each BB interacts with BBs in its neighboring domains (see figure 2.4). In general, two approaches are conceivable: hop-by-hop one-to-one reservations along the path from the source to the destination domain and one-to-many communication, i.e., direct contact of the source domain's BB with any other BB one the path to the destination domain. While the former is more flexible in the sense that each domain decides where to route its inter-domain traffic, the latter promises faster admission decisions.

2.1.3 MPLS

Evolved from a number of similar approaches aimed at simplifying routing in computer networks, the *multiprotocol label switching* (MPLS) architecture [DR00, RVC01] provides a number of opportunities to provide QoS support in networks. The importance of MPLS results from its wide availability and the flexibility to support many important aspects such as resilience mechanisms and traffic engineering.

MPLS was designed as a technology for the simplification of the routing process in a way that allows to use switching technology rather than routing based on complicated metrics. The reason was that routers require more complex hardware and software infrastructure than switches and therefore are more costly at the same level of performance.

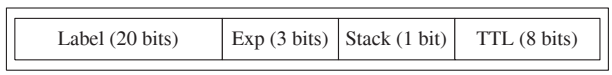


Figure 2.5: The content of the MPLS label stack.

The basic idea is to attach a *label* (see figure 2.5) to each packet in the network. The forwarding decision in each *label switching router* (LSR) for a packet is made solely on the basis of the packet's label. A label is basically a number encoded in a field a fixed length (20 bits) with no further internal structure. In particular, the label is not related to the source or destination address of the corresponding packet in any way. MPLS allows to build a *label stack* that contains more than a single label. Each entry of the label stack consists of four components:

1. the label
2. an experimental field (Exp)
3. the stack bit identifying the bottom of the stack
4. the TTL field, which serves the same purpose as in IP networks

This label stack allows to provide tunneling mechanisms within the MPLS domain.

Incoming Label	First Subentry	Second Subentry
Incoming Label	Outgoing Label Outgoing Interface Next Hop Address	Outgoing Label Outgoing Interface Next Hop Address

Figure 2.6: MPLS forwarding table

Labels are attached to any packet in an MPLS domain. The relation between label and forwarding decision (*label binding*) for the corresponding packet is established using the *forwarding table*. Forwarding tables define which outgoing interface the corresponding packet takes and which outgoing label the packet takes (see figure 2.6). This means, an LSR performs *label swapping*, i.e., an incoming label usually does not remain unchanged. This mechanism permits to have the same label but different meanings (bindings) at different locations within an MPLS domain. In addition to the information specified in figure 2.6, each entry can contain information about the treatment of packets, in particular which resources a packet may use, such as a certain output queue. A set of flows which are forwarded according to the same rules within the MPLS domain are called *forward equivalence class* (FEC). A path in the network for a FEC is

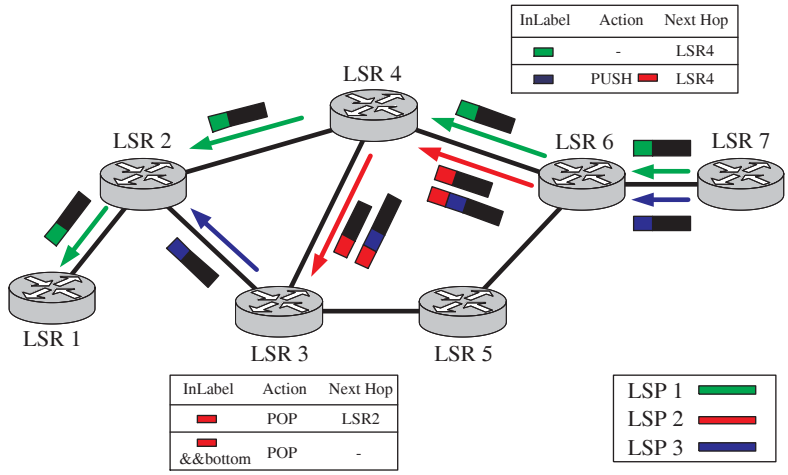


Figure 2.7: Example: LSPs and label stacking.

called *label switched path* (LSP). Each LSP carries packets belonging only to a single FEC.

In figure 2.7, an example of a network with three LSPs is outlined. Packets associated via their FECs to LSP 3 (*blue*) are tunneled through LSP 2 (*red*) using label stacking. The forwarding tables of LSR 3 and LSR 6 show the assignment of labels to packets and the particular action to be taken depending on the incoming label. For example, LSR 3 removes the topmost label of the label stack and in case this was the only label, the packet was destined for LSR 3, whereas when there is another blue label, the packet is forwarded to LSR 2. LSR 6 works similarly: packets with green labels are forwarded to LSR 4 without changes. Packets with blue label get an additional red label attached and are also sent to LSR 4. Packets which enter the network at LSR 6 get a red label attached and are forwarded accordingly. In this example, colors are used as labels. These labels are negotiated between each two LSRs and packets of the same FEC do not necessarily carry the same label on the complete path. Instead, label swapping may be used frequently.

In order to establish label bindings within an MPLS domain, signaling protocols exists, which are designed to exchange and negotiate binding information among the LSRs. In general, it is possible to piggyback the label binding information on top of existing routing protocols such as BGP [RR01], but can also be made using the *label distribution protocol* (LDP) which was especially designed together with MPLS [ADF+01]. LDP runs over TCP in order to establish a reliable communication channel between LSRs. LDP does not only provide means to distribute label bindings but also contains a discovery mechanism which enables LSRs to find each other and to establish communication among them.

Traffic Engineering

Among the functionalities of MPLS, especially *traffic engineering* is a powerful technique to optimize a network's utilization. Traffic engineering mechanisms

aim to avoid that some links are underutilized while others are congested. This in particular is a very important aspect in the context of advance reservations which - as will be shown in chapter 4 - tend to result in a degradation of the network performance. In such an environment which has the potential of negatively influencing the network performance, it is especially important to consider opportunities to efficiently manage the networks resources such that the impact of the performance degradation can be minimized.

One of the important requirements to support traffic engineering is to control and establish routes. For that purpose, MPLS provides an explicit routing mechanism. In general, the required signaling for defining an explicit route can be provided using two mechanisms: RSVP-TE [ABG+01] and CR-LDP [Jam02] which are extensions of RSVP respectively LDP with support for traffic engineering. Both protocols may be used to establish routes, i.e., enforce label bindings resulting in a certain state in the forwarding tables of LSRs, within an MPLS domain.

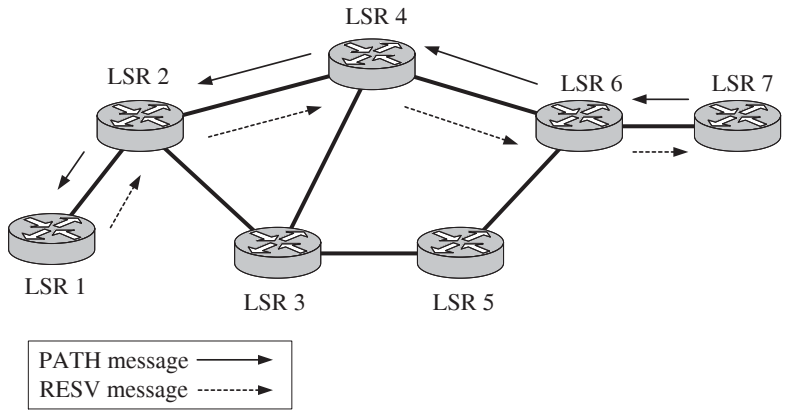


Figure 2.8: Explicit routing using RSVP-TE. The ERO object is used to establish the path <LSR1,LSR2,LSR4,LSR6,LSR7> for a particular FEC.

RSVP-TE is essentially plain RSVP augmented with a new object, the *Explicit Route Object* (ERO). The ERO is attached to the RSVP path message and contains the explicit route to be set up in the MPLS domain. The ERO object is examined by an LSR and the corresponding PATH message is sent along this path (see figure 2.8). The last LSR on this path then generates a RESV message sent on the reverse path. Thus, the common RSVP procedure for establishing QoS guarantees on the given path may also apply, i.e., the normal RSVP mechanism of defining and establishing reservations can be invoked. Establishing explicit routes in the network can be done using a management system, for example, a BB.

In order to establish forwarding state on an explicit route, CR-LDP uses a newly introduced object, called *Explicit Route* (ER). The ER object essentially has the same content as the ERO of RSVP-TE. Another similarity is that, in addition to the explicit route, CR-LDP messages can also contain information about QoS parameters on the explicit route. Thus, it is possible to establish not only the route but also to enforce QoS for the traffic with the corresponding FEC. A route is established in the same manner as with RSVP-TE, i.e., a label

request message is sent along the route to the destination and as response a label mapping message is returned which establishes the required states on the LSRs on the path.

As outlined in [DR00], plain IP routing is not sufficient to support traffic engineering in a suitable way. IP routing mechanisms are destination based routing techniques, that apply a least-cost routing depending on the destination of a packet. This property of IP routing has a considerable impact of the possible architecture for an advance reservation system. The reason is that plain IP routing, even QoS extensions for existing routing protocols such as QoS extensions for OSPF [AKW+99], cannot take the actual utilization on individual links into account for the end-to-end path selection. Since a considerable amount of control over the network routing is required to guarantee the required stability of routes together with its unsuitability for traffic engineering as described before, any mechanisms based on IP routing (such as RSVP based advance reservation systems) cannot achieve the best possible performance in a computer network.

It is possible to combine DiffServ with MPLS. In such a case, explicit control over flow aggregates can be achieved while maintaining the prioritization of DiffServ service classes.

2.2 Advance Reservations in Networks

Previous work on the field of advance reservations covered only some aspects of the overall problem space. Compared to immediate reservations which received most of the attention in the area of network QoS, only a few aspects have been examined:

Requirements Some research dealt with the foundations of advance reservations and central design decisions to be made in such environments. Such aspects have also been covered partly in other publications, dealing mainly with other topics.

Architectures Most of the previous work on the field of advance reservations dealt with architectures, such as extensions to existing reservation protocols.

Admission Control The admission control process has been dealt with mostly regarding the performance of the resulting process.

Performance A number of very different performance aspects have also been examined. This covers the areas of off-line optimizations, computational complexity, and overall network performance.

The results of these previous works laid out a foundation for implementing advance reservations, especially those dealing with fundamental requirements, which can be built upon in order to develop the more advanced services and mechanisms discussed in this thesis.

2.2.1 Requirements

As described before, previous work dealt with basic issues that must be considered when implementing advance reservation services in networks. Besides the

support from the network layer, which is covered in section 2.2.2, this means the difference between immediate and advance reservations, client/management system interaction, the request specification, and the different phases of the negotiation process.

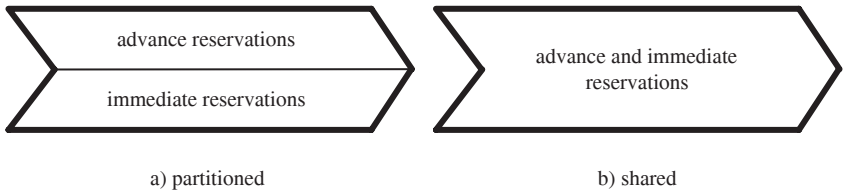


Figure 2.9: Coexistence of advance and immediate reservations in different (*left*) or the same partition (*right*).

In [WDSS95], a general approach to facilitate advance reservations in computer networks was described. Advance reservations for real-time transmissions such as video streaming applications were subject of a detailed examination with respect to different problems to be overcome in a practical implementation, such as coexistence of advance and immediate reservations using shared resource partitions. For that purpose, the advantages of distinct partitions for advance respectively immediate reservations (see figure 2.9) are discussed (see also [FGV95]). The main focus of [WDSS95] is on real-time connections as required for video streaming applications. The authors restrict their in-depth considerations to the network transmission but also mention the requirement to consider the computational resources on the computers involved in the transmissions. They propose a reservation architecture that integrates network and computer resource allocation.

In an extension of that work, a framework for advance reservations was developed in [WS97] and integrated into a general resource reservation architecture. The characteristics of advance reservations are outlined together with a systematic model of the interaction between clients and management system, especially concerning the different states of a reservation system and the corresponding timescale. Three different phases during the lifetime of an advance reservation are distinguished:

1. Negotiation phase between client and network management system
2. Intermediate phase
3. Usage phase of the allocated resources

The model allows for renegotiations during the usage phase, e.g., in order to extend or reduce the duration of a request.

In order to overcome the strict distinction between advance and immediate reservations - respectively the specification of the duration of a transmission at its reservation time even for immediate reservations - in [KBWS99] a framework which allows to provision of a ”general network service” supporting both immediate and advance reservations was proposed. The framework allows to specify a non-preemptable period of time within which a request is guaranteed

not to be interrupted. During this period of time, the given request is marked as prioritized. However, after this time has elapsed the request is not automatically interrupted. Instead, the request may continue without reduction of its QoS until further requests with prioritization, i.e., requests within their non-preemptable period, block the available bandwidth. The system is accompanied by a policy layer that controls the system and applies charges for user's requests. Using the ideas put forth in [KBWS99] allows to increase the flexibility of an advance reservation service and reduce the dependency of the admission control process on the specification of the duration of a transmission request. Especially allowing to change the duration of a given resource allocation further than originally specified is an important feature, which is also used in the field of cluster management [KR01]. Extending the originally specified duration requires invocation of the admission control procedure.

In [FGV95] some basic requirements for implementing advance reservations in computer networks were discussed, such as defining the stop time of an advance reservation request in order to reliably perform admission control. For supporting also immediate reservations, usually being defined without stop time, it was proposed to introduce two resource partitions for advance and immediate reservations in order to strictly divide resources available for both reservation types (see figure 2.9). The boundary between both partitions was proposed to be movable in order to avoid resource fragmentation as much as possible and to support adaption to changing request characteristics. The requirement for defining separate partitions for reservation types with defined end time and those that may run infinitely has also be stated by others [WS97, SBK98, SP97]. The developments described in this thesis were based on the strict distinction between requests with and without definition of the request duration. In this sense, the advance reservation partition accommodates for all requests including the requested duration, no matter whether made in advance or immediately.

In this thesis, the aspects considered in [WDSS95, WS97] are extended covering also performance optimization aspects and services that can be built on top of the basic advance reservation service. For this purpose, the timing aspects are taken into account in order to develop mechanisms that may be used to improve the performance of the network.

2.2.2 Architectures

Many early works related to advance reservations on the field of computer networks mainly concentrated on architectures and extensions of existing reservation technologies in order to support advance reservations. In general, two types of approaches can be distinguished: those that allow and provide explicit control over routing in the network and those that do not. The former are mainly based on a (domain) central management instance such as a bandwidth broker, the latter rely on RSVP or similar mechanisms, or passive agents [SP98b].

In [Rei95b, Rei95a], RSVP and the *Internet Stream Protocol* ST-2 [DB95] protocol were examined with respect to their suitability to implement advance reservations and some general criteria for network protocols that support reserving resources in advance in computer networks have been postulated. In particular, it was stated that *routing stability* is essential for a reliable advance reservation service and thus, routes must be fixed during the negotiation phase. While ST-2 supports this feature using a hard-state approach with fixed routes

after the QoS negotiation phase [Rei95a], the soft-state implementation used by RSVP is less suited due the requirement for periodical state updates and the resulting uncertainty about the actual routes during the usage phase. The stability of routes during the intermediate phase is a very important aspect that, in the implementations presented in this thesis, is dealt with using the MPLS explicit routing functionality which allows one to use a source routing approach initiated by the network management system.

The practical implementations described in [FGV95] focused on extending of the TENET network protocol suite [BFM+94, GHMN95]. TENET however, did not succeed in reaching broad acceptance and availability and hence, cannot be considered as a useful foundation for further developments.

An approach to enhance the RSVP functionality to support advance reservations (called *resource reservations in advance* ReRA) was presented in [SKB97]. The authors considered networks with partial ATM infrastructure which was assumed to be especially suited for supporting network QoS. The basic requirements for implementing advance reservations were discussed, such as the requirement for defining the request duration for reliable admission control. Furthermore, the signaling required for the advance reservations is discussed, especially in the ATM environment. The main focus however, was on the extension of RSVP for supporting RSVP.

The protocol extensions necessary to provide advance reservation support within RSVP were described in more detail in [SBK98]. The focus was on extensions of RSVP and the IntServ approach in order to support also the allocation of resources in advance. The proposed solution mainly concentrates on the protocol extension using an optional field in the RSVP signaling messages to specify the start and stop time of the request.

The exchange of PATH and RESV messages between the routers on the reserved path was extended to cover the whole intermediate phase (see figure 2.10). Modifications of the initial reservation can be made during this whole phase up to the begin of the usage phase. For this purpose, modified RESV messages are sent to the sender. The authors also proposed to extend the refresh intervals and lifetime of a particular soft-state. Thus, it is not only possible to reduce the amount of refresh messages sent during the intermediate phase but also to "survive" temporary router outages and hence the loss of refresh messages. However, the approach does not deal with the treatment of link failures during the transmission phase is not treated. The problem of defining the duration also for immediate reservations when both reservation types use a shared partition was solved using an approach with strict partitioning of advance and immediate reservations. The time model is based on a time interval update technique rather than using the slot-based model. The general problem of resource fragmentation in advance reservation environments is also mentioned and it is proposed to be overcome by defining and advertising periods of time with sufficient bandwidth for the given request to the client. Gathering information for this service is made by collecting status information from each router and providing this information to the client. Finally, an admission control scheme is presented. This scheme is based on the *equivalent capacity* admission control mechanism used in ATM networks.

An agent-based approach to facilitate advance reservations in computer networks was presented in [SP97]. The considerations deal with deploying an infrastructure of reservation agents within routing domains. In order to select routes

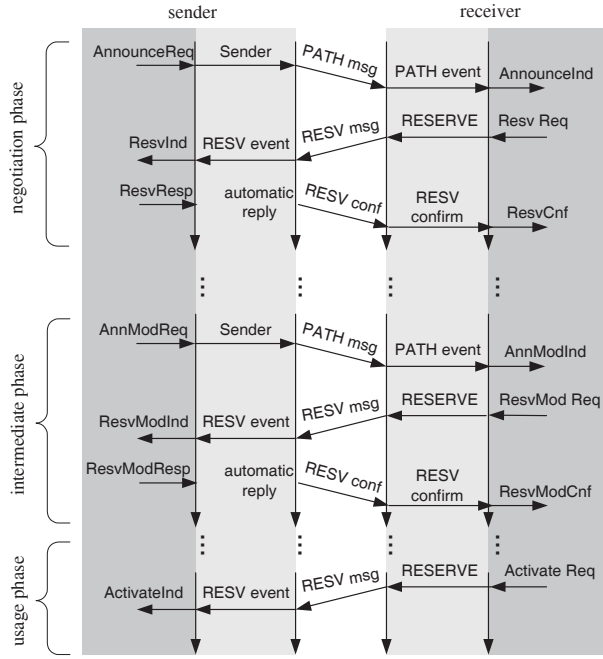


Figure 2.10: Extension of RSVP signaling for support of advance reservations as described in [SBK98].

for advance reservations, the authors favor explicit routing over simply following the available best-effort routes of the network. The latter approach has the significant disadvantage of being inflexible towards route changes. When allowing agents to explicitly set up routes for advance reservation flows, this drawback can be avoided and control over the network is established. The work was extended in [SP98b], with the agents being integrated into a network using OSPF [Moy98] as routing protocol. The agents in this context are configured to passively listen to OSPF messages, thus obtaining information about current topology, routes, and route changes in the network. In order to implement this approach, each router is configured to send OSPF messages to the reservation agent of the respective routing domain. Thus, no explicit routing is done by the agents. Instead, admission control is performed using only the information obtained by listening to OSPF messages sent through the network.

In contrast to these architectures which did not actively influence the network, e.g., in terms of explicit routing, in [San03] a bandwidth broker architecture based on DiffServ in combination with MPLS was proposed. As discussed before, using MPLS provides the opportunity to implement traffic engineering mechanisms. However, in [San03], traffic engineering was not discussed. Instead, the focus was on the provisioning of two higher quality services which are related to the considerations about the EF (premium) and AF (guaranteed rate) PHB definitions (see section 2.1.2). An implementational framework as laid out in [San03], which deals with the implementational aspects of reservations on the network layer, is suited to be used as the foundation for the developments described in this thesis which focus on higher level functionalities, such as traf-

fic engineering, dealing only with flows rather than packet scheduling aspects. However, the implementations presented in [San03] itself do not provide support for any further service than bulk transfer and real-time streaming.

2.2.3 Admission Control

Admission control in general is a broad topic that contains a large variety of different mechanisms and techniques for limiting access to resources of any kind. Furthermore, in advance reservation environments different underlying approaches and frameworks were considered which strongly influence the choice of the admission control strategy. For example, models that consider advance reservations similar to immediate reservations without the need to specify the duration of a given request, cannot rely on accurate knowledge about request durations and hence not provide hard QoS guarantees. As a result, existing research on the field of admission control for advance reservations covers different topics, ranging from probabilistic guarantees that base admission decisions on the preemption probability [WG98], to the implementation of accurate admission control based on complete knowledge about the future [WS97, FGV95] which on the other hand requires additional effort to save this status information [SNNP99, BH02].

In [DKPS95], an admission control scheme for advance reservations was presented. The authors consider proprietary extensions of OSPF network infrastructure to support the advance reservation service based on an agent infrastructure [SP97] comparable to the concept of bandwidth brokers as described in this thesis. In this context, also data structures for storing the status information of future link utilization have been examined [SNNP99] and two tree structures were proposed. However, in [BH02] the most suitable of these trees has been compared to arrays with the result that arrays have significant advantages in terms of admission speed and memory consumption, and furthermore are much easier to implement. Details of the examination and the comparison of both data structures can be found in section 7. In general, the usage of OSPF causes problems in terms of general availability in networks and the inability to explicitly define routes also restricts the benefits that can be achieved using the available status information in order to influence and improve the network performance. The performance aspect which is the main focus of this thesis has not been considered in those papers.

A different admission control scheme for advance reservations based on probabilistic guarantees is developed in [WG98]. In contrast to the other approaches from this field, the authors only consider statistical guarantees rather than performing deterministic admission control. In particular, it is assumed that the stop time is not conveyed within the requests. Although working quite well for the considered scenario, this approach has the major drawback that the distribution of the duration of transmissions must be known in advance. The authors also experienced the problem of performance degradation when reserving resources in advance. However, the reasons were not examined.

In [GSW99], an approach to facilitate the co-existence of advance and immediate reservation in a single partition is outlined. In such a case, one approach is to require immediate reservations also to specify their duration in order to allow reliable planning and admission control. The authors try to overcome this requirement to notify the duration of immediate, instantaneous requests, by al-

lowing requests to be interrupted. They present an admission control algorithm that uses the probability of being interrupted in order to admit or reject a given request.

2.2.4 Performance Issues

Performance is an important aspect in terms of both the performance of the management system that controls the advance reservation environment and also in terms of the achievable network performance, i.e., the amount of admitted requests and the amount of data carried by the network. Previous work touched this important topic only briefly. Three important aspects are described in the following: the first deals with the overall network performance, i.e., the amount of requests and bandwidth accommodated by the network. For that purpose, an on-line algorithm (ROUTE_OR_BLOCK) was presented in [AAP93] which provably achieves the optimal worst-case in terms of throughput of the overall network. The algorithm is used as reference for the evaluation of other approaches developed in section 5.1.1. The second aspect covers the field of computational complexity of routing algorithms in advance reservation environments, which shows the complexity of all problems discussed in this thesis are computationally feasible. Finally, the third performance aspect deals with off-line optimizations of the placement and routing of advance reservation requests in networks. The general framework for these off-line optimizations requires complete knowledge about all the requests in advance. This is considered to be unrealistic for the environments discussed in this thesis. A fourth approach is also mentioned here: *deferred reservations* which were introduced to immediate reservation schemes [NT01] in order to improve the utilization. Although generally interesting, advance reservation environments provide even more opportunities to improve the network performance.

The ROUTE_OR_BLOCK Algorithm

Although not explicitly mentioning the term *advance reservations*, the throughput competitive on-line admission control algorithm presented in [AAP93] also falls into this category. Using only on-line admission control mechanisms, due to the incomplete knowledge about future requests, it is impossible to reach an optimal worst-case result. In [AAP93], lower bounds for the worst-case competitiveness of an optimal on-line routing algorithm called ROUTE_OR_BLOCK were given. Although not explicitly targeting advance reservations, the framework presented in [AAP93] is general enough to cover also this case field of application.

The algorithm achieves the theoretically optimal performance in the worst possible case by avoiding to admit flows that block bandwidth on too many links. In particular, a revenue is associated with each request and flows are admitted depending on the revenue that can be achieved. In this scheme, flows that require only a few links from the source to their destination are preferred since higher overall revenue can be achieved when only those flows. In addition to those requests with small path length, the algorithm also prefers flows with short duration since this allows to maximize the amount of flows to be admitted over time. In general, the revenue function can be arbitrarily chosen and therefore it is possible to implement the admission control algorithm such that, for example,

the total amount of accepted flows or the bandwidth carried by admitted flows is optimized.

The algorithm is outlined in the following:

ROUTE_OR_BLOCK($G(V,E), s, t, t_{\text{start}}, t_{\text{stop}}, r, \rho$)

1 $\forall \tau \in [t_{\text{start}}, t_{\text{stop}}], e \in E$:
2 $\quad c_e(\tau, j) := u(e)(\mu^{\lambda_e(\tau,j)} - 1)$
3 `if` $\exists$ path P in $G(V,E)$ from s to t s.t.
4 $\quad \sum_{\tau \in [t_{\text{start}}, t_{\text{stop}}], e \in P} \frac{r(\tau)}{u(e)} c_e(\tau) \leq \rho$
5 `then` route the connection on P, and set:
6 $\quad \forall e \in P, t_{\text{start}} \leq \tau \leq t_{\text{stop}}$:
7 $\quad\quad \lambda_e(\tau, j+1) := \lambda_e(\tau, j) + \frac{r(\tau)}{u(e)}$
8 `else` block the connection

In the above algorithm, $G(V,E)$ denotes the network graph, s and t denote the source and destination node, t_{start} and t_{stop} denote the start and stop time of the transmission, r denotes the requested transmission rate, and it is assumed that j requests were already admitted.

The parameter ρ denotes the revenue associated with a particular request and can be arbitrarily chosen as described before. For example, if the amount of requests to be accommodated by the network is to be optimized, ρ must be set to 1 for each flow. $c_e(\tau, j)$ denotes the cost of the edge e when request j is admitted (l. 2), with $\mu := 2nTF - 1$. F is chosen such that

$$1 \leq \frac{1}{n} \cdot \frac{\rho}{r(\tau)(t_{\text{stop}} - t_{\text{start}})} \leq F,$$

where $u(e)$ denotes the capacity of the edge e. The "relative" load on a link e at time τ after admitting j requests is defined by $\lambda_e(\tau, j)$ which is updated with each admitted request (l. 7).

where $r(\tau)$ denotes the sum of the allocated bandwidth at time τ. The algorithm rejects a request, when the achieved revenue is less than the "cost" for routing the request. The cost of a request (l. 4) depends on its duration, the number of links used, and the load already present on the links.

Let n be the number of vertexes, i.e., $n = |V|$ and provided that the maximal duration T of all incoming requests is known in advance, it was shown in [AAP93] that the ROUTE_OR_BLOCK algorithm is $O(\log\ nT)$-competitive, i.e., the performance of an optimal off-line strategy always reaches at most $O(\log\ nT)$ times the performance of the ROUTE_OR_BLOCK algorithm. Furthermore, it was proved that this result is optimal, i.e., no other on-line routing algorithm can perform better.

In [AAP93], the performance bound is achieved under the assumption, that the minimal length of a path in the network is 1. In case the arrangement of edge and core routers in the network topology or the traffic pattern guarantees that the minimal length of a path is at least $k < n = |V|$, this lower bound can be reduced to $O(\log\ (n-k+1)T)$. The ROUTE_OR_BLOCK algorithm is used

as reference routing algorithm to compare the performance of the other routing algorithms described in section 5.1.

The algorithm was designed to be run in an environment where each request is admitted on-line, and flows cannot be interrupted once admitted. This is exactly the sort of environment as considered in this thesis.

Computational Complexity

The topic of computational complexity of routing problems related to network with advance reservation mechanisms was discussed in [GO00]. The authors present several routing problems and propose algorithms to solve those problems. Among the routing problems is finding the first available transmission period as also discussed later in this thesis (see section 3.2.1). Most of the problems discussed can be solved in polynomial time, except for some of those that search for the first suitable transmission interval and allow the transmission rate to vary during the transmission period. Such a problem was proved NP-hard and some heuristics were given. The performance of the advance reservation mechanisms was only addressed with respect to the computational complexity. For the approaches discussed later in this thesis, the results of [GO00] mean that not the whole variety of possible services can be implemented in a sensible way, since this meant to solve computationally infeasible problems. In this sense, the authors of [GO00] showed the limits of implementing advance reservations in a realistic system.

Off-line Optimization

Recently, the interest in performance aspects of advance reservations has lead to two publications that deal with off-line optimizations of advance reservation requests. An approximation algorithm for an off-line optimization in an advance reservation system was described by [LNO02]. The authors show how geometric algorithms and linear programming methods can be used to solve this problem when complete knowledge about the whole set of requests is available.

The second approach to optimize the performance in advance reservation environments [Erl02] is also based on complete knowledge about the whole set of advance reservation requests. The author studies both on-line and off-line admission control, i.e., the arrival of requests during the runtime of the reservations system and the arrival of requests at the same time using admission control based on approximation algorithms. The considerations lead to lower bounds of the performance of admission control algorithms for both scenarios which are proved for three types of topologies, star networks, trees, and trees of rings.

These off-line optimization approaches require knowledge about the complete set of requests in advance which is a questionable assumption in a realistic environment. Instead, the considerations in this thesis apply off-line optimization techniques only to the flows that are already known to the management system.

Deferred Reservations

Besides those approaches to improve the performance of advance reservations, another attempt to include the time into a QoS scheme uses *deferred reserva-*

tions. This approach is solely intended to improve the overall performance of a computer network by allowing an otherwise rejected request to be postponed until sufficient bandwidth becomes available on the links with insufficient bandwidth. This approach was outlined in [NT01]. However, although this approach can increase the overall network performance, it is not useful in the scenarios considered in this thesis. Deferred reservations do not allow to provide feedback about the actual time of a transmission to a client since these information are simply not available and is not possible to implement any additional service that could not be offered using immediate reservations. Due to the lack of feedback and hence the absence of a mechanism that provides an additional value for a client in terms of reliable planning, deferred reservations may be an approach to improve the performance of immediate reservation networks, however since this requires additional efforts to deploy the mechanism, advance reservations can be considered as a much more advanced technique with additional value for clients. In particular the field of grid computing, which needs reliability and planning for co-allocation and other purposes, cannot be satisfied using deferred reservations.

2.3 Applications

2.3.1 Overview

Advance reservations are applicable not only for the reasons that are intuitively clear, i.e., to provide increased admission probability when reserving sufficiently early or to enable guaranteed allocations (*co-allocations*) when several resources are required. The ability to reserve in advance, i.e., to get reliable information and guarantees for a time interval in the future, can be seen as a QoS guarantee by itself. This means, the uncertainty about the opportunity to obtain allocations in the future is removed and thus, a new service level with considerably enhanced meaning can be offered to clients.

Applications for advance reservations have been identified in the field of *multimedia* (video conferencing) and *distributed systems*, e.g., media servers [BL00] with large amounts of data to be transferred in a timely fashion among different locations (servers). The latter example also fits the area of *content distribution networks*, e.g., large server networks for web caching. Furthermore, it is conceivable to use advance reservations in order to manage *virtual private networks* (VPN) in a timely fashion, i.e., provide bandwidth guarantees for such VPNs that may vary over time but always require a certain level of QoS. For example, the bandwidth broker application for managing VPNs, as described in [KB00], can be extended to provide such an enhanced service and guarantees the QoS level not only for the current time but also any desired interval in the future.

In the following, a number of additional application environments are presented that benefit substantially from the ability to reserve resources in advance.

2.3.2 Mobile Computing

A special application of advance reservations in computer network is the field of QoS provisioning for mobile devices. Providing QoS for such devices requires to

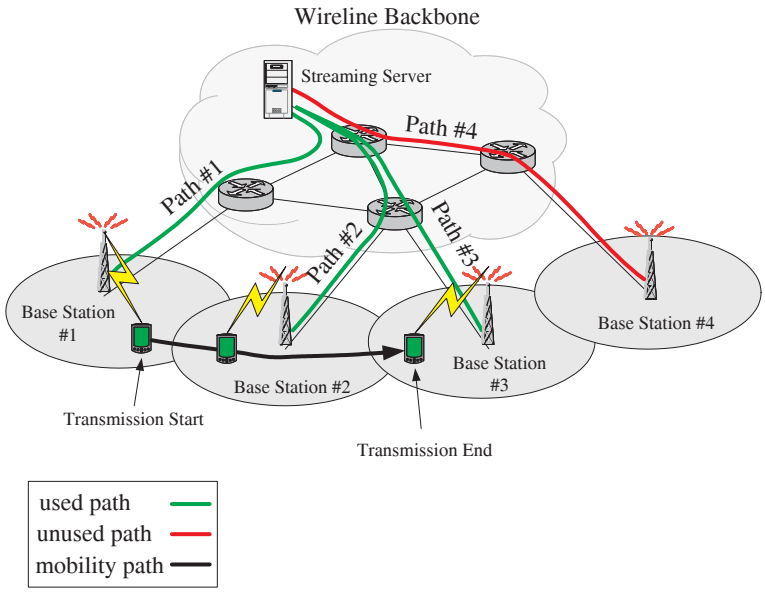


Figure 2.11: Example: mobile, wireless streaming application. A client moves from base station #1 to base station #3. Resources on the paths to any possible base station, i.e., #1 to #4, are reserved in advance.

deal with handovers. Considering a server sending a video stream to a mobile device, in a typical scenario the server is located somewhere in the wireline Internet and the stream crosses a number of links until it reaches the access point to which the mobile device is currently connected. In such a situation, the QoS required for streaming the video is established along the route from the server to the mobile client which involves a number of wired links. However, when the client moves around and eventually leaves the area of its initial access point, the handover that takes places requires not only to transmit the video across a new path from the server to the access point but also to establish the QoS guarantees for the video stream along this new path. This handover must be done transparently for the client, i.e., without noticeable reduction of the QoS. Such a situation is depicted in figure 2.11. It shows the problem of resource wastage in such a situation: although the fourth base station is not accessed, the resource on the corresponding path are reserved and hence not available for other applications.

Such an approach has been outlined in [CCM+00]. The authors proposed a framework for QoS in wireless networks called *ITSUMO*. Its foundation is a bandwidth broker called *QoS Global Server* (QGS). This QGS is responsible for negotiating QoS with wireless clients. Requests for QoS are only admitted when the resources are sufficient in *any* of the possible access points. When a reservation is successful, resources in all those access points are reserved. The backbone is considered to be equipped with DiffServ-aware infrastructure.

A similar approach was described in [TBA01] and allows a mobile client to set up a number of reservations along the paths from the server to the access points that will be visited by the client at a later stage in advance. Ideally, the set of

access points is known in advance. Otherwise predictions must be made which was not further discussed in [TBA01]. Reservations are made to any access point that may be visited in the future. In order not to waste too many resources, the approach distinguishes *active* and *passive* reservations. The resources on the paths that are currently unused may be used by other reservations. Once a path becomes active, i.e., right before a handover takes place, the state of the reservation is switched to active and is then exclusively available for the client that initiated the resource allocation. This has the drawback that the total amount of bandwidth available for reservations may be considerably decreased because the scheme does not even allow other requests to allocate the resources assigned to reservations in their passive state.

Another example for applying advance reservations in the field of wireless, mobile networks is the application of *M-RSVP* [MA02], an extension of RSVP to support wireless clients. The main idea behind this extension is to reserve resources for existing sessions to the surrounding access points in advance. In order to implement the reservation of resources for a wireless client which moves around, it is critical to implement handovers between the different access points and the wireless clients. Once an RSVP session is established between the client and a server in the wireline network, this session and the respective reservation will be lost once the client moves to another access point. The session therefore must be established in the wireline network between the server and the access point. The M-RSVP protocol allows to establish a resource reservation between not only the currently active access point but also the access point(s) most likely being accessed by the client in the near future before the handover actually takes places. In this sense, M-RSVP can also be seen as an advance reservation protocol for a special class of mobile applications.

In these environments, advance reservations impact the performance of the underlying system in the sense that resource are wasted when resource allocations are kept on several paths in parallel, i.e., from the server to more than a single access point. This is not only effective for the paths to access points that are actually accessed during the lifetime of the transmission but also for paths to access points that are not visited at all but had to be taken into consideration, e.g., due to a false prediction on the basis of a mobility profile [CCM+00]. It is possible that the predicted movement of a mobile client differs from its actual behavior. For those reasons, the applicability of advance reservations in this context is questionable and thus not explicitly in the focus of the considerations in the following sections.

2.3.3 Cluster and Grid Computing

One of the most important fields of applications for advance reservations is the area of cluster and grid computing [FK99b].

Cluster Computing

In the field of cluster computing, advance reservations are a mature concept to allocate compute resources.

Large scale computers such as clusters require management systems in order to control access to the nodes, monitor status, handle failures, and schedule jobs. An example of a management system for such cluster systems is CCS

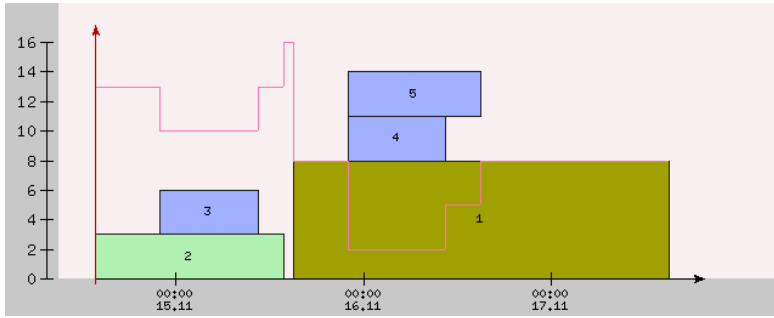


Figure 2.12: Advance reservations of compute nodes using the CCS cluster management system on a 16 node cluster.

[KR01] (see figure 2.12) which supports advance reservations of cluster nodes, thus enabling reliable planning of resource allocations. In order to reserve not only the compute resources in advance but also the network connections between several advance reservation cluster systems, it is necessary to have an advance reservation service available on the network layer, too.

Grid Environments

Although various other approaches exist to support grid computing, such as UNICORE [Uni], the Globus toolkit [GLO] gained the most importance and can be regarded as the de-facto standard in this area. This toolkit implements means for the allocation of various types of compute resources in the field of grid computing (see also [FK99a, FKL+99]). In [FKL+99], the implementation of the resource allocation component *GARA* of Globus is outlined. In order to keep track of future reservations, a time slot model is adopted [GO00, BH02], i.e., the timeline is divided into *slots* of fixed size. This model was also adopted for the implementations presented in this thesis (see section 3.4.2). A system component called *local resource allocation manager* (LRAM) is responsible for keeping track of the allocated and available resources in the future. The LRAM component can be omitted in case an actual reservation system for the given resource type is available which is capable of advance reservations. Resources in this context may be any type of compute resources, mainly processor and network, but may also include others, such as human resources, or other technical installations such as radio telescopes.

The implementations concerned with the network resources as described in [FKL+99] were based on RSVP without extensions for supporting advance reservations and hence, an additional LRAM component was required. The structure of GARA allows clients to use the respective functionality of any network reservation system with support for advance reservations. Hence, it is possible to use the bandwidth broker presented in this thesis also in a grid environment based on Globus.

An example for a grid application is depicted in figure 2.13, describing a tele-immersion application with the requirement to co-allocate very different resources. A large amount of data received via satellite (e.g., from a space telescope) is transmitted to distributed computer systems for processing. These computer systems generate data streams which are transmitted to a third loca-

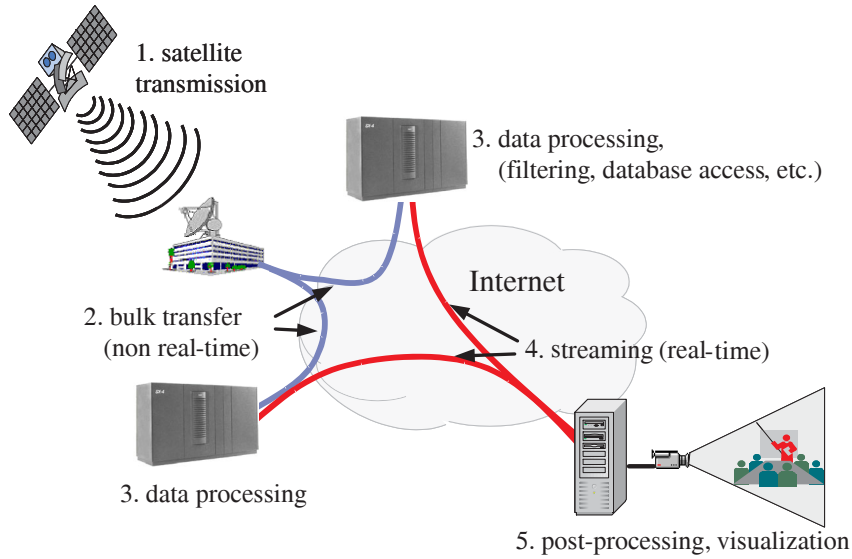


Figure 2.13: Example: grid computing application using several resources and network transmissions. Both bulk transfer of large amounts of data (step 2) and real-time streaming transmissions (step 4) are required.

tion where the data is multiplexed and visualized. The visualization controls the data processing and the real-time streaming. In this setting, the co-allocation of the different resources is essential. The scarce resource "space telescope" for example determines the earliest time for which the reservations of the processing units and the visualization infrastructure can be made. It must be assured, that the network transfer of the bulk data is finished before the processing starts. This processing may take some time before the actual streaming and finally visualization can start. These timing dependencies must be considered during the co-allocation which reasonably can only be made in advance.

As outlined in section 2.2.2, a bandwidth broker has been described in [San03] in order to support the particular requirements of different grid computing application scenarios. However, as the thesis only implements two services required by the grid environment, namely for real-time and bulk data transmission, the additional requirements of grid computing in terms of higher level services (see also [BHK+04]) are not addressed.

Future Directions

The current development in the area of Grid computing moves towards collaborative compute environments interconnected by dedicated network infrastructures, e.g., using a single fiber of an optical network solely for grid communication. Such infrastructures are called *LambdaGrids*, an example is the TransLight project as described in [DdLM+03]. In such environments, a bandwidth broker which supports advance reservations can be used without the necessity to support existing network infrastructure and components, e.g., based on RSVP or MPLS, but may be directly applied in the network. If the LambdaGrid provides static routes, i.e., point-to-point connections, the important issues such

as routing is no longer required. This simplifies the application of an advance reservation service and allows to concentrate on the actual purpose of advance reservations, i.e., providing enhanced services required to support future grid computing infrastructures as described in the following.

Within such infrastructures, the enhanced services to be provided will be based an SLA negotiations and deal with a different notion of network QoS than mainly used today. These services, which will be discussed in the following sections of this thesis, include fault tolerance and failure recovery, the capabilities to support guaranteed deadlines, and allow clients to reliably plan their compute jobs [BHK+04]. In this context, QoS requirements focus on the applications needs rather than the actual implementation of scheduling or routing algorithms. Application requirements are the support for reliable co-allocations of various resources with guaranteed start times and deadlines. The assurance of deadline guarantees in turn requires the management systems to provide means for failure recovery during runtime in order to find alternative resources if the primary resources fail. Network support in this case does not only need to deal with link failures but also with deadline guarantees for transmissions of bulk data that occur due to job migration (*run-time responsibility*, see [BHK+04]).

2.4 Summary

The technologies, architectures, and algorithms previously presented show that - among others - there exists a gap in previous work on advance reservations in the sense, that besides developing architectures upon which advance reservation services can be implemented, the possible impact of those service on the network performance have not been discussed in detail. Although sometimes mentioned in the previous work, the newly added *temporal* dimension raises a number of questions related to various performance aspects of the management of advance reservation networks. In particular, it is of interest how the introduction of the additional temporal information can be used to improve the network performance for clients and operators of a network. Performance in this context is meant in a broader sense than only throughput, utilization, or admission probability. In addition, such things as planning reliability, fault tolerance in case of link outages or failures and the performance of the management system, e.g., in terms of response times, are important to examine, in particular when SLAs are negotiated that define not only the behavior of a single component of a large system but the QoS provided by the system as a whole which includes any component involved in processing a compute job [BHK+04].

The previously mentioned QoS techniques and methods can be seen as the foundation for future implementations. In order to introduce a new network service such as advance reservations, it is generally important to avoid that a whole new infrastructure must be deployed in order to implement the service. Instead, it is favorable to rely on existing infrastructures which can be used without major adaptations.

When reviewing the QoS mechanisms available in current networks including MPLS, the drawbacks of especially the IntServ/RSVP approach become obvious. Traffic engineering based solely on this protocol is impossible since it relies on IP routing. The QoS requirements of future applications, e.g., in the field of Grid computing, can only be dealt with by providing an infrastructure that

supports more than the currently available QoS mechanisms.

In order to build a management infrastructure in such an environment, the concept of bandwidth brokers as defined in the context of DiffServ together with the flexibility of MPLS is a suitable foundation for advance reservations and the additional service on top of it, which will be described in the following sections. In particular, DiffServ may be used to provide partitioning of the available resources and to implement a service class for advance reservation traffic which is separated from immediate reservation traffic with unknown duration or best-effort traffic. Depending on the nature of the traffic, bulk transfer or real-time traffic, both the EF and AF PHB may be specified. Another example for such a strict partitioning approach is the design of LambdaGrids which uses a dedicated fiber of an optical network to interconnect the different entities of a grid system, for example, clusters and networks. Such partitioning has the advantage of allowing only access for reservations with specified duration and hence avoiding complications with other reservations where the duration is unknown. The combination of bandwidth brokers as admission control and management component within dedicated networks, together with MPLS which provides the functionality for traffic engineering, is the foundation for the implementations described in the following sections.

Chapter 3

Advance Reservation Framework

In order to provide an advance reservation service and the additional functionality introduced by the temporal dimension, a framework is required that manages the network resources and provides interfaces to clients in order to enable the service negotiation. Using the concept of bandwidth brokers as already discussed in chapter 2 is a suitable way to deal with such issues. As described in [San03], using MPLS together with DiffServ provides a suitable foundation for advance reservations which guarantees control over the network infrastructure as required for implementing a reliable advance reservation service.

In this chapter, the special properties of advance reservations in a computer network are presented. The general concept of bandwidth brokers and various aspects related to immediate reservations were introduced and widely discussed in previous work, e.g., [NJZ99]. Hence, the focus in the following sections is on the advance reservation environment and its special requirements that must be supported by a bandwidth broker. Furthermore, the additional services that can be offered to clients are discussed and how these services are implemented.

3.1 Application Environment

Due to various problems related to the different timing specifications in immediate and advance reservation requests, the approach described here is based on strict partitioning of advance and immediate reservations with known duration and those with unknown duration (see section 2.2.1). This procedure simplifies the implementation and can be extended to support also shared partitions if necessary. The restriction to resource partitioning does not affect the general approaches and techniques applicable in advance reservation environments which are the focus of this thesis. Most recent developments in the area of network support for e-science and grid computing focus on the implementation of LambdaGrids [DdLM+03, ABW04] where a dedicated fiber is available for inter-connecting globally distributed compute resources. Such a LambdaGrid can be understood as a network that provides a dedicated resource partition for the advance reservation service. The partitioning approach does not require a LambdaGrid, but can be implemented using common Internet technologies,

for instance, when using a particular DiffServ service class solely for advance reservations.

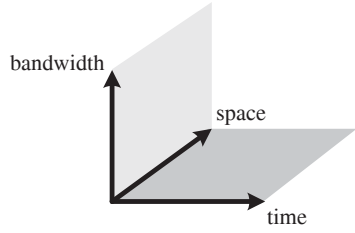


Figure 3.1: Dimensions of the advance reservation problem. The spatial (routing) and bandwidth dimension also exist in immediate reservation environments.

The space in which clients may specify their requests and the management system can influence the overall network performance is determined by the three dimensions *space*, *bandwidth*, and *time* (see figure 3.1). The spatial dimension in this context means the path selection process, i.e., space in this sense is the network topology, and the option to adjust the bandwidth is also a realistic opportunity in the advance reservation environment. The main property of the advance reservation service is to add the temporal dimension to the reservation mechanism. In contrast to advance reservations, in an environment that supports only immediate reservations the temporal dimension can only be included with very limited scope. For example, the approach to implement deferred reservations, i.e., requests which cannot be fulfilled when they occur are delayed, deals with this temporal aspect [NT01].

3.1.1 Reservation Model

The advance reservation service provides admission control for advance reservation requests. Such requests are processed and replied to in an on-line fashion such that the time between request and response as short as possible. This is essential in order to guarantee that in case a co-allocation of different resources in a grid computing environment is made, the admission decision can be provided as soon as possible to the grid application.

The general service model for advance reservation requests has been described in other previous publications, e.g., [WS97]. Two essential definitions, which will be used frequently throughout this thesis, are made in the following.

Definition 3.1 (Advance Reservation Request) *An* advance reservation request *is defined as* $r_a := (s, d, t_{start}, t_{stop}, b)$*, where* s *and* d *are the source and destination node,* b *denotes the requested transmission bandwidth, and* t_{start} *and* t_{stop} *denote the start and end time, respectively.*

Definition 3.2 (Reservation Time) *When an advance reservation request is issued to the management system at time* t_{resv}*, the difference* $t_{start} - t_{resv}$ *is called* reservation time.

An advance reservation request is sent to the bandwidth broker which processes the request and grants or denies access to the network during the requested time period.

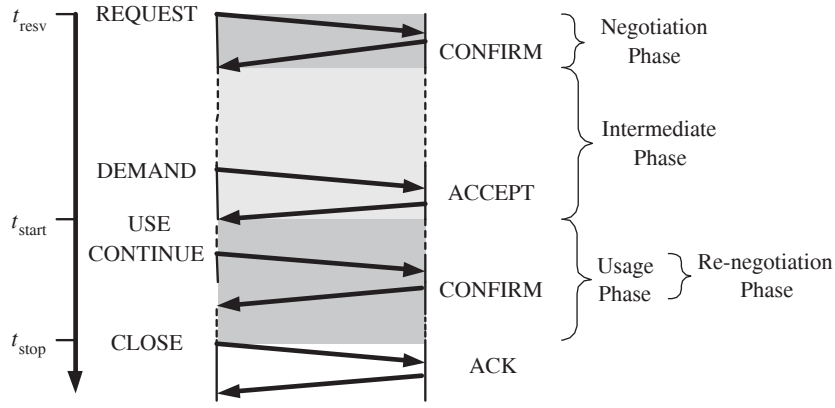


Figure 3.2: Advance reservation primitives and phases of the reservation [WS97].

This leads to the temporal sequence as depicted in figure 3.2. As in [WS97], the three phases between the initiation of the request at t_{resv} and the end of the usage of the resources at t_{stop} can be distinguished, using the reservation primitives shown in figure 3.2.

During the first phase (negotiation), the admittance to the requested resources is negotiated. This may include one or more rounds of REQUEST and CONFIRM (or REJECT) messages exchanged between client and bandwidth broker. The intermediate phase starts after the resources have been successfully reserved and ends with an explicit DEMAND primitive, which denotes that the client is alive and ready to start the usage phase. The DEMAND primitive may initiate a preparation of the resources for the usage phase. The intermediate phase is assumed to be considerably longer than the negotiation phase and perhaps even than the usage phase, and may last several hours, days or even weeks. In case the requested resources are not needed until the end of the negotiated usage phase, the handshaking (CLOSE/ACK) at the end of the usage phase can be used to signal to the bandwidth broker that the resources are no longer required and may be used otherwise. In general, when a reservation terminates as scheduled, it is not required to explicitly notify this event. For security reasons, the bandwidth broker must signal the end of the usage phase to the edge router in order to avoid having further packets emitted into the network by unauthorized sources.

The negotiation phase is not restricted to only a single round of messages as depicted here, i.e., negotiations with several rounds can be implemented. For example, as required for more sophisticated resource allocation protocols, such as auctions or malleable reservations as discussed in section 3.2.2, it is desirable to implement multiple rounds of negotiation in order to eventually reach an agreement on the extend to which the network resources can be used by the client. For example, it is possible that the BB responds to the initial request for a fixed transmission interval with a set of possible parameters that describe reservation alternatives in case the requested interval may not be available. In a second phase, the client may select one of the alternatives which is then acknowledged by the BB. Using such a mechanism is not only required to

implement auctions or malleable reservation but in case of an initial rejection may be required to negotiate service parameters that are subject of an SLA between client and management system.

3.1.2 Management Layers

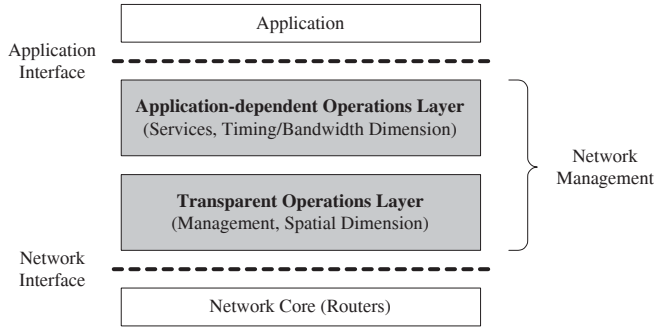


Figure 3.3: The two management layers of the bandwidth broker.

The bandwidth broker described in this thesis consists of two main layers which identify the different approaches to improve the performance. The *application-dependent operations layer* contains the functionality to support additional services, as will be described in section 3.2, such as the search for a suitable transmission interval. Below this, the *transparent operations layer* contains the mechanisms that are used to manage and optimize the network performance transparently for the applications. This contains the strategies that deal with the optimization in the spatial dimension, i.e., in particular the routing strategy. The two layers of the management system are depicted in figure 3.3. Both services and management mechanisms are described in the following sections.

3.2 Services

Advance reservations provide an intuitive model for increasing the admission probability and provisioning of reliable planning. On top of this advance reservation service, a number of additional services can be implemented that are new in the field of computer networks and improve the network performance perceived by clients. These services cannot be implemented in a common immediate reservation system but require the special temporal properties of the advance reservation environment.

In [San03], a bulk-transfer service and a real-time transmission service required in the grid computing environment were presented, thus concentrating on the way how a given requirement for a transmission is mapped onto the corresponding functionalities of the network. In this context, the term *service* refers to the way certain traffic characteristics can be assured using the router mechanisms available to prioritize network packets.

In contrast, the services outlined here each describe a functionality that uses in particular the additional temporal dimension in the advance reservation framework.

3.2.1 Search for Transmission Intervals

In a straightforward manner, a new service that can be implemented in the advance reservation environment is to determine the *first available transmission period* for a request of given duration and transmission rate. In the same way, the search for a suitable time interval for a transmission within a given *deadline* can be implemented.

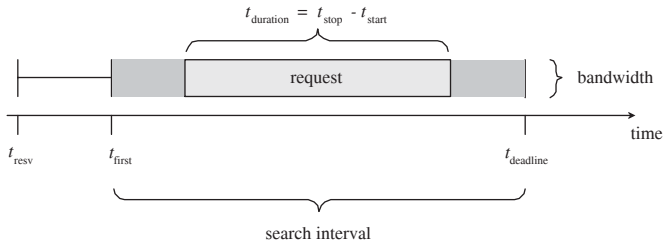


Figure 3.4: Parameters and temporal sequence for requesting the first available transmission interval starting at t_{first} respectively the search for an interval up to the deadline t_{deadline}.

Such a service is preferably implemented in the management system of a network. Although in general, it is possible to implement this functionality within a client by repeatedly issuing requests until finally a suitable transmission interval is found, the network overhead for submitting the requests and the computational overhead for processing frequently repeated requests is high and can be avoided. In an immediate reservation system such services cannot be implemented at all because information about the future is not available. In addition to the simple search for a transmission interval for a fixed rate transmission, e.g., required for streaming video, the transmission rate may vary during the transmission interval in order to most efficiently utilize the available network resources.

For this kind of service, in addition to the duration and bandwidth, the earliest possible start time t_{first} must be submitted within the REQUEST message and perhaps the deadline for the transmission. The actual parameters then must be included in the CONFIRM message sent by the bandwidth broker (see figure 3.2).

3.2.2 Malleable Reservations

When extending the service outlined before and also allow the transmission rate to vary within given boundaries, this leads to *malleable reservations*. This type of reservations has so far mainly been discussed in the field of job scheduling [LT94, Jan02, VD03]. The general idea of malleable reservations is that the duration of a job is a function of its resource consumption. This general framework allows to vary the resource consumption over time and also start and end times of the job, thus making resource management more flexible and improving the resource utilization.

In situations that require a fixed amount of data to be transmitted over a

network, the exact transmission rate is not of importance as long as the general capabilities of the sender and receiver, e.g., a maximum transmission rate, are not exceeded and optional timing constraints for the transmissions, e.g., a deadline, are met. As in the scenario outlined in section 3.2.1, in such a setting it is suitable to implement the search for an appropriate transmission interval and rate as a service within the bandwidth broker. A sample application for this type of service is transferring backup data collected during night time to a storage server until the next morning, or the transmission of input data required for a computation on a grid system to the computers involved in the computation before the computation starts. Another application is a content distribution network such a distributed media server system [BL00] where large amounts of media files must be distributed across the interconnecting network to the different servers.

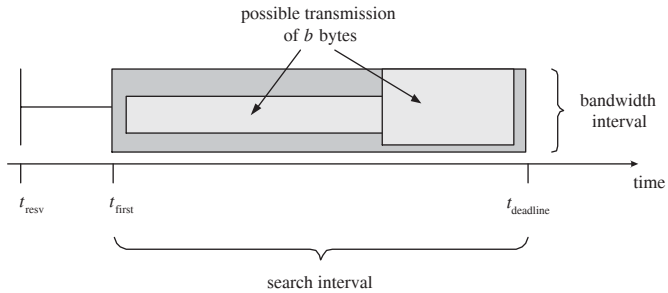


Figure 3.5: Parameters and temporal sequence for requesting a malleable reservation, e.g., for data transfers. Two possible shapes of the transmissions are given, both with d bytes being transmitted in total. The bandwidth constraints of sender/receiver ($b_{\min}$ and $b_{\max}$) must be specified within the request.

As described in section 3.2.1, the actual transmission rate may vary during the transmission period. In case the applications themselves are not capable of adjusting the transmission rate accordingly, traffic shaping can be implemented on the edge devices (routers) of the network by dropping packets in case the incoming packet rate is too high. Thus, the sender can always use the maximally possible transmission rate, and by dropping packets on the network devices, most appropriately edge routers, the desired rate can be met.

This concept is somewhat orthogonal to what has been described in [FRS00], where on-line QoS adaptations are proposed using sensor information from the network. In our case, the admission control process of the management system takes the sensor part (due to its knowledge about current and future network status) and is responsible for the adaption of the transmission rate and time. However, in contrast to [FRS00], in case of advance reservations these parameters are guaranteed by the bandwidth broker after the decision of the management system has been made. The parameters may be subject of an SLA, e.g., for the application in the larger context of a grid system [BHK+04].

In addition to the parameters identified in section 3.2.1, the capabilities of the sender and receiver in terms of maximal and minimal transmission rate must be specified for this service (see figure 3.5).

3.2.3 Feedback

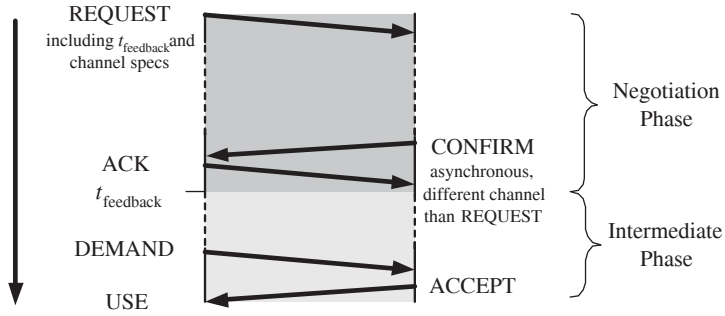


Figure 3.6: Feedback: communication between client and bandwidth broker

The possibly considerable amount of time between request submission and acknowledgment (CONFIRM), respectively the transmission start indicated by the DEMAND primitive can be used to implement a service that provides feedback even for rejected requests. Considering a given request, it might not be possible to admit the request to the network immediately after its submission, even in case only loose constraints for a transmission interval were specified. However, after a certain amount of time, but before the actual start time of the rejected transmission request, the status of the network may change - e.g., due to a reorganization of the admitted flows (see section 3.3.5) - and as a consequence it may be possible to admit requests that had to be rejected before. In those cases, the bandwidth broker can provide this as feedback to the clients. This can also mean to delay the response to a request until the latest possible point in time as specified by the client, thus increasing the time available for the admission decision.

In figure 3.6, the steps during the communication between client and server with enabled feedback are depicted. t_{feedback} is specified by the client within the request and denotes the latest point in time for the feedback message to arrive at the client. The response message contains information about the actual timing and bandwidth parameters as chosen by the admission control process. Upon reception of such a message, the client replies with an acknowledge message in order to notify the acceptance to the management system.

In order to implement such a functionality, appropriate asynchronous communication channels between clients and bandwidth broker must be established. This is difficult to engineer since it highly depends on the communication partners involved. For example, a human client may receive an email whereas a software client, e.g., an autonomous agent acting as a part of a grid toolkit, may be accepting asynchronous messages using a suitable network protocol. Moreover, the feedback channel and the feedback protocol (email, TCP, etc.) must be submitted within the request.

Table 3.1: Parameter set of advance reservation requests

Service Type	Parameters	Status
`Fixed`		optional
	`Start time`	mandatory
	`Stop time`	mandatory
	`Transmission rate`	mandatory
`First-fit / Deadline`		optional
	`Duration`	mandatory
	`Transmission rate`	mandatory
	`Time`	mandatory
`Malleable`		optional
	`Total amount of data`	mandatory
	`Earliest start time`	optional
	`Deadline`	optional
	`Min. transmission rate`	optional
	`Max. transmission rate`	optional
`Feedback`		optional
	`Deadline`	mandatory
	`Feedback channel specs.`	mandatory

3.2.4 Request Types

The services described before require additional parameters to be submitted with a request. These parameters are summarized in table 3.1. Exactly one parameter out of `fixed`, `first-fit/deadline`, and `malleable` must be specified, changing the advance reservation request (see definition 3.1) accordingly. The time parameter belonging to `first-fit/deadline` is interpreted as first possible start time (first-fit) or deadline, respectively.

The first three items in the table describe the different service types which can be submitted using a number of mandatory or optional parameters. The feedback specification can be used to allow off-line optimizations and requires to specify the deadline for the reception of the response message to the request as well as the specification of the feedback channel, e.g., email, RPC, etc.

3.3 Managing Advance Reservations

In general, in the advance reservation environment any performance optimization mechanism which is applicable in immediate reservation environments can also be used. This includes techniques such as call preemption [GG92] or parallel transmissions on multiple paths [AAA+02, LG01]. In addition to these mechanisms, advance reservations allow the management to use a number additional techniques in order to improve the performance and to admit more requests.

In the following, the basic ideas behind these techniques are presented. The actual implementation and performance will be discussed in chapter 5.

3.3.1 On-line versus Off-line Mechanisms

In contrast to immediate reservations, where usually only on-line admission control is performed, i.e., requests are admitted as soon as possible after the submission of the corresponding request, the advance reservation scheme allows to include both on-line and off-line mechanisms in the management system. In particular, it is possible to perform an optimization of the flow-to-path mapping off-line. This extends the considerations in [WS97], which proposed to use the intermediate phase for the preparation of the resources for the usage phase, e.g., set up routes. A management system should be capable of processing requests both on-line, i.e., a response is expected as soon as possible, and off-line, where "off-line" means, for example, that the response from the management system is delayed (see section 3.2.3).

The problem to be dealt with in any case is to find suitable optimization criteria for both on-line and off-line mechanisms. Using the on-line admission control, knowledge about the whole set of reservations that will ever enter the system is not available and admission control must be performed individually for each request. This is likely to lead to suboptimal admission decisions [GG92].

In a realistic scenario, requests frequently enter the system at any time and due to the dynamics in this environment, global knowledge is also unavailable to an off-line strategy. This means, it is infeasible to compute a globally optimal solution for the admission control problem, i.e., to determine which requests to admit and which to reject. In such an environment, the strategies proposed in [Erl02, LNO02] (see section 2.2.4) are not applicable.

3.3.2 Routing

In general, routing in advance reservation networks has the same purpose has in immediate scenarios: the task is to find a path with sufficient bandwidth in order to provide the requested transmission rate. For this purpose, any suitable algorithm may be chosen, for example, Dijkstra or Bellman-Ford. The only optimization opportunity arises when several such feasible paths exists. Then, the routing algorithm needs to select one of the feasible paths.

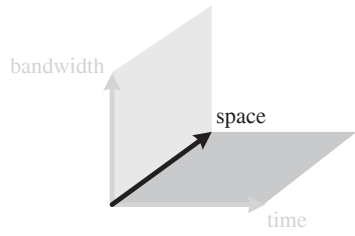


Figure 3.7: Routing in networks deals with the spatial dimension of the problem space.

In this sense, routing deals with the spatial dimension of the advance reservation problem as depicted in figure 3.7. A considerable amount of research has been dedicated to the field of routing and in particular QoS routing. The different approaches can be divided into source routing, hierarchical routing, and distributed routing strategies [CN98]. In the environment as presented here

which supports explicit routing based on MPLS, a source routing approach is appropriate. In general, source routing has some drawbacks, such as the difficulty to obtain a global state and inaccurate state information due to propagation delay and protocol overhead, in particular in case of link failures. However, in the environment considered here, i.e., a bandwidth broker with knowledge about the network topology, the network management itself keeps the global state (i.e., the information about link utilization) and thus does not rely on information from nodes. The general setting uses a specific partition of the network capacity only for advance reservations and immediate reservations with known duration. This partition is exclusively controlled by the bandwidth broker and hence, the drawbacks of source routing do not apply here. The routes can be set up by the bandwidth broker and in case of failures, a number of different fast rerouting schemes exist which do not require intervention of the bandwidth and can be applied also in the advance reservation environment.

Suitable routing strategies for advance reservations are scarce and not well-studied so far. There are mainly two interesting aspects in this context. The first is the computational complexity of routing problems under the advance reservation paradigm. This aspect was covered for a number of different scenarios in [GO00]. A number of routing problems related to advance reservations are computationally feasible, among the problem of finding a feasible path for a request with fixed parameters 3.1. Only certain problems, allowing the resource usage to vary during the transmission interval, are computationally infeasible, i.e., NP-complete. The considerations in this thesis concentrate on fixed bandwidth problems, since they are the most realistic ones. The second aspect of routing in advance reservation environments is the overall performance of the routing algorithms which has not attracted much attention in previous work.

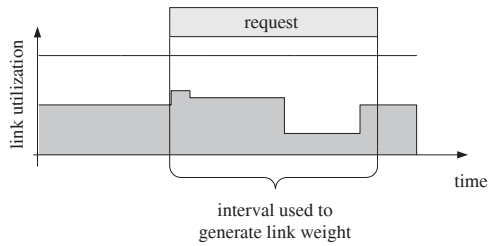


Figure 3.8: Routing: usage of link utilization to weight paths.

In order to develop routing algorithms that are successful in advance reservation environments, it is possible to adopt strategies applied in immediate reservation scenarios. Routing algorithms that use link utilization to weight paths have been shown to perform well in such immediate reservation environments [MS97a, MS97b, WN02]. In that context, link utilization refers only to a single point in time, i.e., the actual time. This must be extended in advance reservation scenarios in order to cover the whole duration of the request (see figure 3.8). An example for such an algorithm is the ROUTE_OR_BLOCK algorithm presented in section 2.2.4.

The performance of a particular routing algorithm in terms of achievable number of admitted flows and throughput depends on the nature of the requests, e.g., in case only requests between two end nodes of the networks are submitted,

it is useful to exploit any possible network path between those nodes independent of the hop count, whereas in case requests between many different pairs of end nodes are submitted it is preferable to only use the shortest paths in order not to waste bandwidth by allocating bandwidth on too many hops.

Concluding, it can be said that optimizing the performance in a network with advance reservations using the on-line routing strategies differs from the situation in an immediate reservation scenario, but the difference are only moderate. In particular, the additional complexity of the routing process is only a single factor which depends on the duration of a given request.

3.3.3 Flow Switching

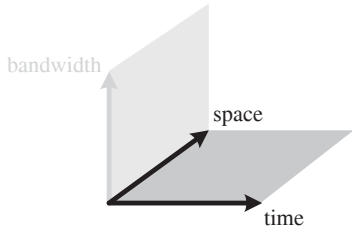


Figure 3.9: Flow switching: the usage of multiple paths during the lifetime of a flow affects both the spatial and temporal dimension.

In order to improve the performance of the network transparently for clients, the advance reservation environment allows to transmit each flow using multiple paths during its transmission interval, i.e., a flow can be switched to a different path during its lifetime (*flow switching*). It must be noted, that this does *not* mean the transmission of a single flow using multiple paths in parallel.

The *flow switching* approach affects both temporal and spatial dimension (see figure 3.9). The time is affected at the single path level, since different paths are used at different points in time.

In contrast to transmitting the packets of individual flows using multiple paths in parallel [LG01], this approach does not have the drawback of requiring intelligence within network devices that have to split flows, or large overhead due to an increased number of routes to be set up on the network. Moreover, the sequential usage of several paths during a data transmission does not require additional efforts such as reordering packets at the sink and can be implemented without impacting the sender or receiver application.

An example is depicted in figure 3.10. The search for the paths can be efficiently implemented by iteratively computing the path with the longest possible time interval with sufficient bandwidth. The path computation can be implemented, e.g., using Dijkstra's shortest path algorithm as foundation.

A possible drawback of this approach is that the packet delay may change due to the path switches and in case additional metrics besides bandwidth are important for a flow, this must be considered.

The flow switching approach using several paths can only be applied in the advance reservation environment by using the knowledge about future link utilization.

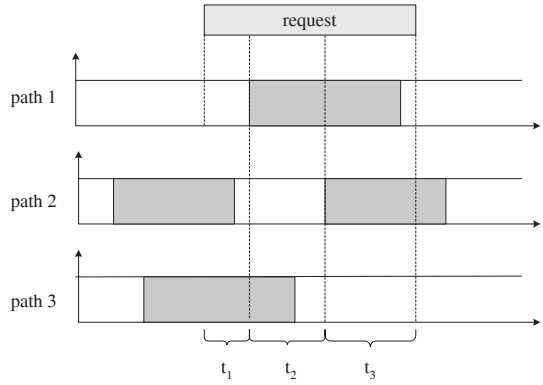


Figure 3.10: Example of a flow taking multiple paths. The flow is routed using path 1 during the interval t_1, path 2 during t_2, and path 3 during t_3.

3.3.4 Flow Scheduling

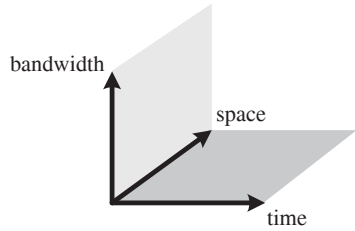


Figure 3.11: Flow scheduling affects all three dimensions of the problem space.

In the advance reservation environment, the time difference between transmission request and transmission start allows to handle requests much more flexible than in an immediate reservation environment. The malleable reservations service specified in section 3.2 where only boundaries for transmission in terms of duration and bandwidth are given, allows to schedule such flows according to the needs of the management system, i.e., that the network performance is optimized. Thus, all three dimensions of the optimization space can be exploited (see figure 3.11). In [Bur03a], the flow scheduling approach has been examined, showing that the network performance can be significantly increased.

Using the on-line admission control approach, it is not possible to optimize the network performance from a global view, i.e., each request is admitted individually and hence, any global optimization criterion such as the number of admitted flows or the utilization of the network resources cannot be used. Consequently, only algorithms can be applied that try to follow local optimization criteria. In this context, "local" refers to the individual admission decision, i.e., accepting or rejecting a request, where accepting is the optimization goal.

3.3.5 Off-line Optimization

As discussed in section 3.3.1, the intermediate phase can be used to perform off-line optimizations, i.e., it is possible to optimize the placement of flows onto

the network. Two variants are conceivable: the first requires to contact the client and provide feedback concerning the parameters of the transmission in an asynchronous manner, the second only performs optimizations which do not require to send feedback to the client. The latter deals with rerouting of admitted flows, e.g., by choosing multiple transmission paths over time. Thus, it is possible that the off-line optimization only affects the spatial dimension, but also all three dimensions (in case feedback is possible) may get affected.

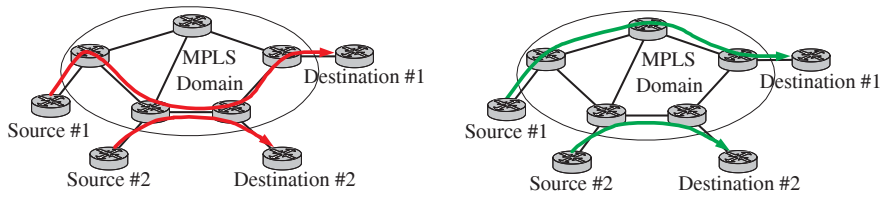


Figure 3.12: The route setup may result in blocking in the common link (*left*). Off-line optimization can result in rerouting (*right*) and more efficient resource usage.

One task in this context is to determine the set of flows that can be considered during the optimization. In general, any admitted flow can be included in that set. Furthermore, the advance reservation environment allows to take rejected flows into account using the feedback mechanism described in section 3.2.3. Together with the flow scheduling option (see section 3.3.4), this allows the management system to optimize not only the routes, which can be done transparently for the network end points, but also to adjust the start and stop times of future transmission according to the needs of the network management. In figure 3.12, an example for this procedure is shown. A commonly used link may be blocked when routes are set up as depicted on the left. This can be detected and changed using the off-line optimization.

Such an optimization can be rather successful even in case only the currently active flows - which represent only a subset of those flows that can be considered in the advance reservation environment - are considered for rerouting. This was shown in [BKL03], where the authors presented an optimization technique for rerouting flows in MPLS-based networks. The technique can easily be adapted to fit into the advance reservation environment.

In contrast, the optimization strategies for advance reservations proposed in [Erl02, LNO02] require knowledge about the whole set of requests. It is questionable, whether such assumption is valid in a real-world scenario where the status of the network constantly changes due to requests arriving frequently over a long period of time rather than altogether at a single point in time. Consequently, coordination between on-line admission control and off-line optimization is required.

3.3.6 Failure Recovery

An important task of a network management system is to deal with link failures. In case a failure occurs on any link of a path, the flows affected by the failure are ordered, e.g., by their request time, and in this order switched - if possible - to one of the other paths from the set which does not contain the broken link.

Moreover, not only the flows being active at the time the failure occurs are rerouted but also those that are expected to start within the downtime of the broken link can be rerouted in advance which reduces their preemption probability. The failure recovery strategy in an advance reservation network is especially important with respect to the performance of the network. The consequences of the additional temporal dimension for the failure recovery mechanisms are discussed in chapter 6.

3.4 Software Architecture

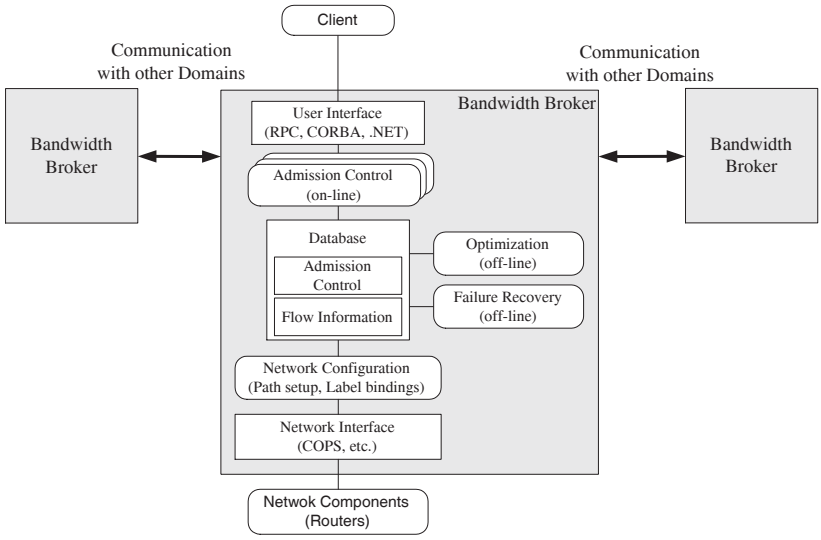


Figure 3.13: Software architecture of the bandwidth broker

In order to provide the services previously described it is required to set up a management infrastructure which provides the interface between network layer, i.e., MPLS aware routing infrastructure, and clients such as grid or mobile applications. Unlike the previous approaches that do not actively influence the network behavior, e.g., RSVP-based mechanisms [SKB97] or passive management agents [SP97], the usage of MPLS allows the management system to apply the services and management techniques previously described by defining and setting up routes on the network using the explicit routing functionality of MPLS. This is the purpose of the bandwidth broker. Its basic functionality is to perform admission control, to set up paths, and to initiate binding of flows to paths within a network domain. Performance from the view of the network operator usually means to maximize the profit of the network in terms of accepted requests and throughput, i.e., the amount of bytes carried by admitted flows. Based on the properties of the advance reservation environment and using the services previously defined, the bandwidth broker can implement a set of optimization procedures which improve the utilization and benefit of the network.

The bandwidth broker implements the services and optimization mechanisms

described in sections 3.2 and 3.3. The basic task of the bandwidth broker is to control admission to the network, i.e., to check whether sufficient resources are available to satisfy a given request, and to communicate with the network components. The software architecture of the bandwidth broker is depicted in figure 3.13. In addition to the denoted components, a policy module is required in order to perform authentication and authorization. However, as the focus of this thesis is on other aspects, the policy related questions are omitted here.

3.4.1 Interfaces

The bandwidth broker requires to expose three interface for the communication with the "outside world". The first is the user interface which provides functionality to submit requests for network resources and the second is the interface to the network which implements the functionalities required to establish the advance reservation service. The third interface is responsible for the communication between bandwidth brokers in multiple domain environments.

Application Interface

The application interface can be implemented using a variety of different methods such as RPC, CORBA, Java RMI, or .NET. These functional interfaces can be integrated into web services in order to provide user access. Hence, it is possible to have human clients as well as software agents. The actual type of client (human or software program) also has implications on the functionality of the bandwidth broker. For example, a software agent may allow several rounds of negotiation and establish return channels thus allowing a considerable amount of time between a request and the respective admission decision whereas human clients may want to receive the response (i.e., admission or rejection) without any delay.

Network Interface

The interface between the bandwidth broker and the network devices, i.e., in general these are edge routers, can be implemented using several currently available protocols such as COPS [Dur00, CSD+01], or SNMP [MCRW96, CMPS02].

If the COPS protocol is used in order to enforce certain policies, e.g., QoS or security, on the network, each network device connects to its *policy decision point* (PDP) upon boot-up. In the context of this thesis, the policy relates to the QoS guarantee for admitted flows and the PDP is the bandwidth broker. Upon connection, the broker then provides the network device with any information necessary to perform its role. For edge routers (PEPs) this comprises, e.g., information about set up of explicit routes and currently admitted flows.

In case a new flow is about to enter the network, the bandwidth broker uses the protocol to inform edge routers (*policy enforcement points, PEP)* about this event and - when required - initiates the definition of a new explicit route (see figure 3.14). Following that, the traffic flow is admitted to enter the MPLS domain at the specified start time. In order to accurately provide the requested resources to clients, it is necessary to start the signaling and path setup a certain time before the transmission commences. This scheme using explicit signaling from the PDP is called *control driven*.

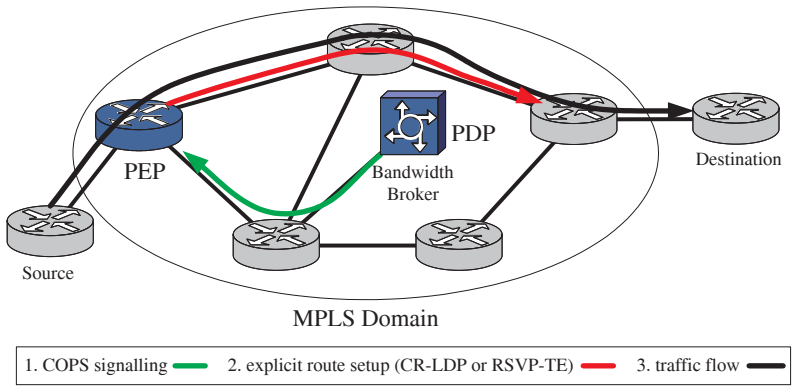


Figure 3.14: Interface BB-Network: signaling between bandwidth broker (PDP) and edge LSR which acts as PEP.

Using the reservation setup as outlined in figure 3.2, the signaling of the bandwidth broker is preceded by a DEMAND message from the client requesting a transmission. The subsequent steps can be classified as preparation of the requested resources for the usage phase [WS97]. Even when this DEMAND message is omitted in the protocol, the control driven approach allows the bandwidth broker to prepare the resources in a way that upon arrival of the first packet of an admitted flow, the respective path is set up on the network.

As an alternative to the previously described control driven scenario, it is conceivable to implement a *data driven* preparation of the requested resources. This means, upon reception of a new data flow, the PEP (edge router) will contact the bandwidth broker in order to acquire information about admissibility and perhaps routing for the new transmission. The bandwidth broker may then provide the requested information to the PEP, which in turn initiates the path setup as in the control driven setting. This approach has the disadvantage that a considerable amount of time may pass between the arrival of a new flow at the PEP and the end of the resource preparation. During this period, the traffic flow will not receive its guaranteed QoS. As this is usually not desirable, the control driven approach should be preferred over the data driven approach.

In conjunction with an underlying MPLS infrastructure, routes are established on the network using the explicit routing functionality. The setup of a new path using CR-LDP or RSVP-TE on the network only needs to be performed if the path does not exist yet. In any other case, it is sufficient to bind the new flow to the respective label at the ingress node. This means, whenever a route can be reused for a new flow, the bandwidth broker only notifies the arrival of this new flow to the corresponding PEP which is responsible for appropriate labeling of the flow's packets.

Inter-Broker Interface

In order to communicate with other domains and to allow resource reservations across domain borders, it is necessary to communicate among bandwidth brokers of different domains. The different architectures and approaches for managing inter-domain traffic in advance reservation scenarios will be described in chapter

8.

The actual interfaces for establishing communication channels between brokers in general does not necessarily differ from those used for the communication between client and bandwidth broker. However, an effort to develop a protocol was made by the Internet2 QBone Signaling Group [Tea01]. The *Simple Interdomain Bandwidth Broker Signaling* (SIBBS) protocol was developed but reached dissemination only in the limited QBone environment. Its design goal was to support the QBone premium service which can be observed as an instance of the DiffServ premium service as described in section 2.1.2 and is characterized by six parameters, i.e., start and end time, source and destination, MTU size, and peak rate. The SIBBS protocol in this context is a TCP-based request-response oriented protocol which can be used to request a certain type of service guarantee of the QBone premium service. SIBBS acts on a 1:1, domain-to-domain basis exchanging pairwise messages between two bandwidth brokers. This means, a request message is sent from the source to the destination domain which - upon successful reservation - in turn sends a response message back to the source domain (see also section 2.1.2). In case of resource shortages in any of the domains involved, the request message is immediately responded by a negative response message.

3.4.2 Admission Control

Requests from clients are processed by one or more admission control processes (parallelism can be implemented using the locking mechanism described in section 3.4.6) which uses the information stored in the database. This process implements the services and strategies discussed in the previous sections, such as routing or scheduling malleable requests. Once the admission decision is made, the database is updated and the response sent to the client. In case, no decision can be made in time, such a request is rejected.

Slotted Time

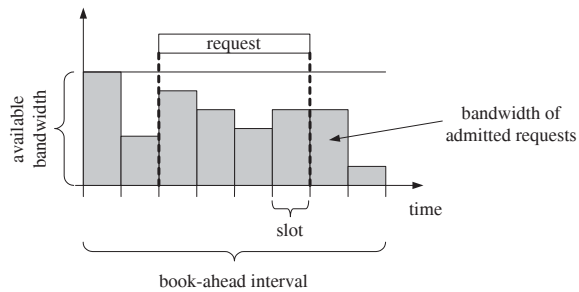


Figure 3.15: Storage of Requests using slotted time and fixed book-ahead interval.

In general, two opportunities exist to specify the timing parameters of an advance reservation request. The first to allow a client to choose the start and end time of a request without any restriction, i.e., it is possible to select values with arbitrary granularity. On the other hand, several approaches exists

which are based on *slotted time*. In this model, the timeline is divided into slots of fixed size as depicted in figure 3.15. Thus, reservations can be issued for a number of consecutive slots. The duration covered by a single slot can be chosen depending on the actual application environment, i.e., a slot may represent a minute, hour, etc. This slotted time scheme has been widely adopted [SNNP99, GO00, FKL+99, BH02] and therefore is also applied for the implementations made in this thesis. Furthermore, the slotted time model does not represent a restriction if slots are reasonably sized.

Book-Ahead Interval

Using the slotted time model, the resource usage for each link in the network must be stored by the bandwidth broker. This information is accessed during admission control and used to determine whether sufficient bandwidth is available to admit a request. Storing utilization information in this way provides much faster access to the relevant information during admission control in contrast to storing only flows with the related reservation data. The reason is, that the admission speed is decoupled from the amount of admitted flows because only a certain number of slots, which depends on the requested duration, needs to be checked for each request.

The time interval for which reservations can be issued is called *book-ahead interval*, which may be infinite or restricted to some large number in order to limit the amount of status information stored in the bandwidth broker (see figure 3.15). Both opportunities are conceivable, however using only a limited book-ahead interval restricts the memory consumption of the bandwidth broker.

3.4.3 Database

Besides storing information about link utilization for the book-ahead interval, it is required to keep the admitted flows in a separate database. This database stores the flow's source and destination, start and end time as well as the flow's path. Information like this is required to set up network paths and to reroute flows in case of a link failure. This database can be implemented using commodity software and hence is not subject of this thesis.

The database is organized as two individual parts. The admission control database keeps information about static parameters such as the network topology, the available bandwidth on each link, and the utilization of the links during the book-ahead interval. Furthermore, the link status (UP/DOWN) is stored for each link. The data structures used to store the link status are examined in chapter 7.

A second database is required to store information about admitted flows (source and destination node, start and stop time, and path). This database is queried periodically (once per slot) by the a process responsible for the configuration of the network, i.e., the flow-to-label binding and - if required - the explicit routing, i.e., the path setup. Furthermore this process collects information from the network about link failures and updates the respective data in the admission control database.

3.4.4 Off-line Optimization

The off-line optimization process is supposed to run in the background while requests arrive frequently at any time. Hence, the optimization process deals with optimizing the routing transparently for clients and optimizing the placements of those transmissions that requests that accepted feedback at a later time.

3.4.5 Failure Recovery

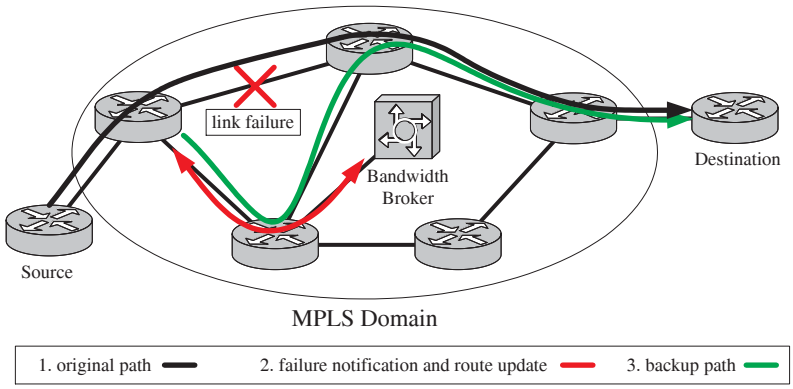


Figure 3.16: Sequence of events after a link failure.

In case a link of the network fails and is detected by the network components, this is notified to the bandwidth broker. The failure recovery module is then started and initiates the rerouting. For each affected flow, the failure recovery module determines an alternative path if possible and signals the new routes to the network (see figure 3.16).

The failure recovery module runs as an independent process. The priority of the respective process should be higher than that of the others, i.e., admission control and off-line optimization, in order to handle failures as fast as possible and to recover as many affected flows as possible. In fact, it is conceivable that the failure recovery completely blocks the bandwidth broker for further requests in order to assure that the maximal amount of affected flows can be switched to alternative paths in the shortest possible time. The details of the failure recovery will be described in chapter 6.

3.4.6 Locking Mechanisms

In order to run the independent processes in parallel, i.e., one or more on-line admission control, off-line optimization, and perhaps failure recovery processes, without completely blocking one of them, it is required to implement a locking mechanism that takes the special properties of the advance reservation management system into account. This means, the data structures of the management system must not be locked completely for either of the two processes rather than only the relevant parts. The three relevant parts in this sense correspond to the three dimensions of advance reservations (see figure 3.1):

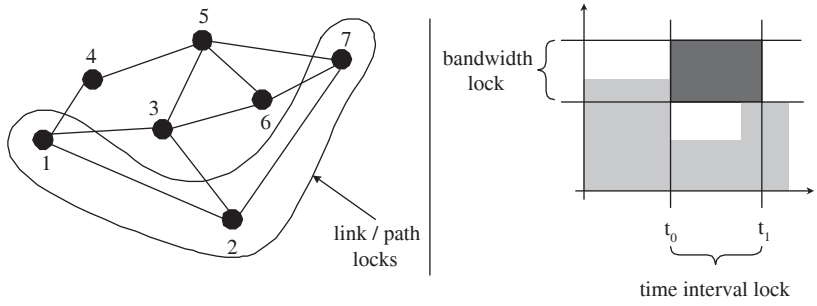


Figure 3.17: Example: different locking strategies for a reservation request

Links Only parts of the network require locks that are currently used, i.e., certain paths required for admission control for a particular request. This is especially useful when using the k path routing strategies with restricted path set for each pair of network end points as described in [Bur03b]: in this case only the links on the k precomputed paths are locked. Furthermore, it is possible to perform local off-line optimizations, i.e., only a subset of the network links is considered for that purpose.

Time It is not required to lock the complete book-ahead interval when a particular request processed by the on-line admission control only affects a short period. The off-line optimization may still work on the remaining portion of the book-ahead interval, i.e., it is possible to optimize flows starting earlier or later.

Bandwidth When applied to one or more links, this lock restricts access to the remaining amount of available bandwidth. This means, the currently considered bandwidth is marked as unusable for other requests.

The locking strategies previously described can be combined in order to restrict the impact of a single lock and thus, the granularity of the actual locking mechanism can be chosen. This means, the parts as required by the respective process can be locked using one or more of the locks previously described.

In figure 3.17, an example is given showing all three locking mechanisms. The figure shows the situation for a request from node 1 to node 7 with the bandwidth and time interval $[t_0, t_1]$ as shown on the left hand side in the dark gray box. On each link on the path, i.e., $(1, 2)$ and $(2, 7)$, locks for a certain amount of bandwidth and time interval are applied as depicted on the left hand side. The area denoted by the gray box shows the parts on the each links that are actually not accessible for other admission control or optimization processes. For example, a request on the same path for a time interval overlapping with $[t_0, t_1]$ will be blocked until the current request is either rejected or admitted. On the other hand, requesting bandwidth during a time interval before t_0 or after t_1 will not result in blocking the corresponding process.

Deadlock Avoidance

The locking mechanisms need to be implemented in a way that avoids deadlock situations. In order to avoid a scenario where several processes are in a deadlock situation, each process can be assigned a unique ID which determines the priority of the respective process. Thus, higher priority processes may be given preference in accessing the required locks. This ordering avoids deadlocks in a simple and straightforward manner.

Chapter 4

Performance Analysis

The performance of advance reservations is an important issues, as the benefit for a network operator depends not only on the implementational effort and cost of providing advance reservation mechanisms but also on the benefit that can be achieved by offering those mechanisms to clients, i.e., the amount of traffic that can be accommodated by the network.

4.1 Performance Metrics

Before discussing the performance issues, in this section the metrics used to measure the performance in a network are given and the methodology for the performance analysis is discussed. For a network operator, the profit of a network mainly depends on the price that can be charged for the transmissions in the network. For a client, it is important to have a high probability that its transmissions are accepted by the network management. The *request blocking ratio* (RBR), i.e., the probability that a request is rejected by the network, can be easily measured. In contrast, the profit for a network operator cannot be assessed that simple because it depends on many different parameters, such as the charging model. Instead of trying to model such a complex system, in this thesis the focus is on the amount of bandwidth that is transmitted by accepted flows, measured by the *bandwidth blocking ratio* (BBR). These two metrics are widely used as metrics for measuring the performance of computer networks [MS97b]. Formally, RBR and BBR are defined as follows:

Definition 4.1 (Request Blocking Ratio, Bandwidth Blocking Ratio) *The request blocking ratio (RBR) is defined as*

$$\text{request blocking ratio} := \frac{|\bar{R}|}{|R|}, \tag{4.1}$$

where $\bar{R}$ denotes the set of rejected requests and R denotes the entirety of issued requests. The bandwidth blocking ratio (BBR) is defined as

$$\text{bandwidth blocking ratio} := \frac{\sum_{f \in \bar{R}} b(f)}{\sum_{f \in R,} b(f)} \tag{4.2}$$

with $b(f)$ denoting the total number of bytes transmitted by f.

RBR and BBR define the different properties that are important in the context traffic flow in networks. A low RBR does not necessarily mean a high network utilization since flows carry very different amounts of data. Consequently, the BBR covers the amount of bytes transmitted by the flows in the network. Other possible metrics, e.g., related to the profit achieved by the network operator are assumed to be derived in some way from RBR and BBR and are therefore not discussed here.

4.2 Analysis

It has been discovered in previous works that advance reservations can lead to a different allocation and utilization of resources in contrast to immediate reservations. This effect is not a specific problem of computer networks, however did not attract much attention in the past research concerned with advance reservations in networks. In particular, in [WG98] this effect is mentioned without giving an explanation but stating that "booking ahead can lead to either an increase or a decrease in utilization depending on the traffic mix". While it is true that advance reservation can coincidentally also increase the network performance, for instance, in terms of utilization, it is much more likely that the performance degrades due to *resource fragmentation*, as will be shown in section 4.2.2. Furthermore, the fragmentation does not depend on the traffic mix but the results presented in the following sections lead to the conclusion that this is a phenomena that occurs frequently in such environments and is directly related to the fact that reservations are made with different reservation times. It will be shown that resource fragmentation is the cause for an increased blocking of particularly requests with low transmission rate.

4.2.1 Overview

In general, the performance of advance reservations in comparison to immediate reservations cannot be predicted accurately. In general, when all reservations are made in advance, it is either possible that the performance increases or decreases with respect to the metrics presented in definition 4.1. The reason is that allowing requests to be made in advance only changes the order in which the reservations are made. Under the assumption that the network load is sufficiently high and requests are being rejected - which is the only sensible scenario in which to study the performance impact of advance reservations compared to immediate reservations - it is possible that a single request of minimal bandwidth and minimal duration blocks the complete bandwidth for any other request.

This is not a unique property of advance reservations but may happen with both reservation types. An example for such a worst-case scenario is depicted in figure 4.1. A small request of minimal duration (1 slot) and minimal bandwidth allocation may block a large request independent of the reservation time of the small request. In both cases, the large request may also be made in advance, only the order in which both requests are made - small before large - is important for the performance. The only difference is the order in which both requests are made. Thus, advance reservations have the potential to either improve or degrade the performance compared to immediate reservations.

The question is, whether fragmentation is either a common phenomena or

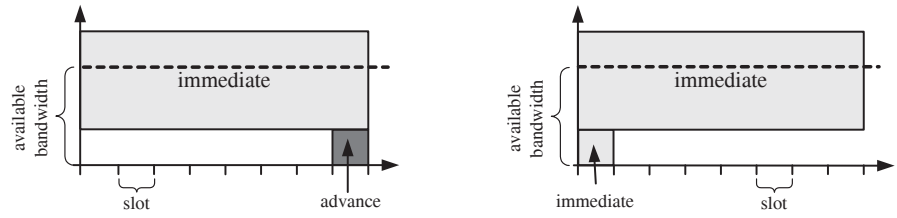


Figure 4.1: This example illustrates the worst-case behavior of advance (*left*) and immediate (*right*) reservations: a small request blocks a large request in both scenarios.

unlikely to happen, whereupon in the latter case, no additional measures are required by the network management in order to avoid fragmentation. Additionally, it must be determined how the fragmentation affects the network performance, i.e., in which way the two metrics outlined in definition 4.1 are affected when reserving in advance.

4.2.2 Resource Fragmentation

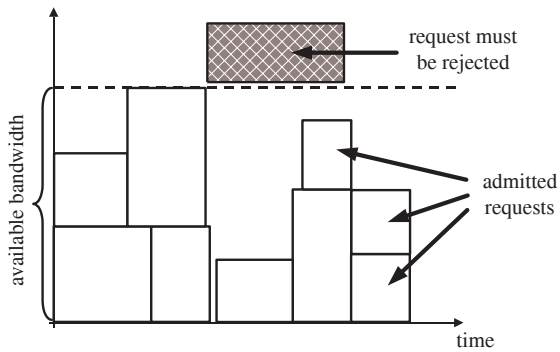


Figure 4.2: An example of resource fragmentation resulting from advance reservations. Future requests block the unused bandwidth.

As depicted in figure 4.2, the opportunity to allocate resources in advance may lead to a fragmentation of the available resources. The reason is that "gaps" appear when a certain amount of requests for future resource allocations has been admitted. Between two peaks of small distance bandwidth remains unused since on further requests are not sufficiently short to fit into the resulting gaps.

The resource fragmentation as outlined in figure 4.2 is the reason for the performance difference of advance and immediate reservations. Under the assumption, that in a given set of requests there are always sufficient requests to completely utilize the available bandwidth, it is possible to achieve 100% utilization in immediate reservation environments. In the same setting, allowing to reserve resources in advance may lead to fragmentation.

In order to assess the probability that fragmentation occurs, the following scenario is considered. Assume unit transmission rates for all requests, a network

consisting of only a single link, and unit bandwidth availability on that link, i.e., at most one request can be admitted per time unit. Furthermore, assume a set R of requests, all requests in R are ordered according to their reservation time (see definition 3.2), and there is more than a single request starting at any given time, i.e., an overload scenario[1].

When the requests in R are issued according to their given order, the result is the immediate reservation scenario. It is easy to see, that this ordering results in the optimal bandwidth utilization since new requests occur at any given time, i.e., there is no unused bandwidth. In contrast, when a single request r is issued t_r time units in advance, this results in fragmentation. Let $T_{\min}$ be the minimal duration computed over all requests $r \in R$. For a duration equal 1, the problem of fragmentation does not exist, therefore it is additionally required that $T_{\min} > 1$.

A request r that is made $t_r < T_{\min}$ time units in advance and can be admitted inevitably results in fragmentation of the bandwidth since $T_{\min} - t_r$ bandwidth units cannot be allocated by subsequent requests. The reason is that t_r for the request r differs from the other reservation times, i.e., subsequent requests exist for transmission bandwidth before t_r. This means, the probability p_{frag} that fragmentation exists in such a scenario depends on the probability that the reservation times of requests differ. It is clear, that the case where $t_r^i = t_r^j = t_r$ for each two requests $r_i, r_j, i \neq j$ does not lead to fragmentation in this sense, because this situation describes an immediate scenario with a "time shift" of t_r.

When $t_r > T_{\min}$, the request r leads to fragmentation in the case that the sum of the durations of requests admitted after r does not equal t_r. This event can considered to be extremely likely and thus, the probability for the event that fragmentation occurs is also very high, i.e., in general close to or equal 1.

The previous observations can be generalized to the case where a link accommodates more than unit bandwidth. In this case, w.l.o.g. the first request r reserved $t_r < T_{\min}$ time units in advance occurs in a situation where r can be admitted such that the remaining bandwidth is too few to admit additional requests. As in the simple case discussed before, r then blocks at least t_r bandwidth units. Similar considerations lead to the conclusion, that fragmentation also occurs in the case that $t_r > T_{\min}$. In addition to the condition, that the sum of the request durations does not equal t_r, in this case it is also required that the sum of the requested transmission rates does not equal the blocked bandwidth which is a stronger, thus less likely, requirement.

In order to determine the impact of advance reservation on the performance in a network with multiple links, it is useful to examine how resource are allocated in a network when - as in this thesis - bandwidth is used as metric. In this case, bandwidth can be seen as *bottleneck metric* [WC96] (sometimes also called "concave" metric [XN99]) in contrast to, for example, delay which is an additive metric. The nature of the bottleneck metric implies, that fragmentation may occur also in a multiple link scenario.

Concluding, the examination of the very simple scenario shows, that fragmentation occurs very likely in case advance reservations are allowed, i.e., almost anytime reservation and start times of two different requests differ. The actual amount of fragmentation depends on the distributions of the reservation times, start times, transmission rates, and transmission times of requests.

[1] Otherwise, reservations are not required at all (see chapter 1)

4.2.3 Worst-Case Bounds

Since advance reservations almost inevitably lead to a degradation of the network performance, it is also of interest to determine to which extend the network performance is affected.

Under certain assumptions about the distribution of the requests, it is possible to determine bounds for the performance of advance reservations in the worst case. In this context, the worst case means that the maximum amount of requests are rejected and the maximum amount of transmission bandwidth remains unused. The assumptions are that

1. at each time slot a sufficient number of requests is available in order to completely utilize the available bandwidth
2. the minimal duration T_{min} for a request is known in advance

The first assumption also implies, that when issuing all the requests in an immediate reservation environment the total amount of bandwidth can be utilized by admitting only the requests with minimal duration. This assumption describes a scenario with a competition for the available resources which is also the only scenario where reservations make sense at all (see chapter 1). The second assumption is sensible, since it is unrealistic to assume that a reservation - which requires additional administrative effort for the requesting client - is made for a request with a duration that is hardly noticeable, in particular a request with a duration of only 1 slot is not realistic. Hence, a lower bound for the minimal transmission duration can be at least estimated.

The Single Link Case

In this case, the worst-case performance with respect to request blocking ratio and bandwidth blocking ratio can be determined depending on the reservation time t_r of an advance reservation, i.e., the resource fragmentation resulting from a single advance reservation will be assessed in the following.

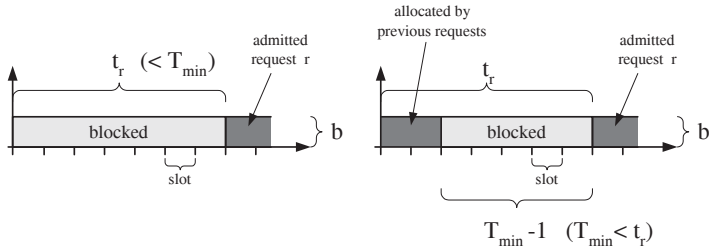


Figure 4.3: Worst-case bandwidth blocking: bt_r ($t_r \geq T_{\min}$) respectively $b(T_{\min} - 1)$ ($t_r < T_{\min}$) units of bandwidth are blocked (light gray rectangles).

Throughout the following, the notion outlined in figure 4.3 is used. Consider an admitted request r of given bandwidth b with a reservation time of t_r - i.e., it is reserved t_r slots in advance - and furthermore, let the shortest possible duration of a request again be $T_{\min}$. Provided that at each time slot t there is at least one request of duration $T_{\min}$ starting at t, in the worst case the request r blocks at most either

$$\begin{aligned} bt_r \quad &, \quad t_r < T_{\min} \text{ or} \\ b(T_{\min} - 1) \quad &, \quad t_r \geq T_{\min}. \end{aligned}$$

bandwidth units that cannot be used by other allocations, provided that for each time slot a sufficient amount of requests with duration $T_{\min}$ are issued.

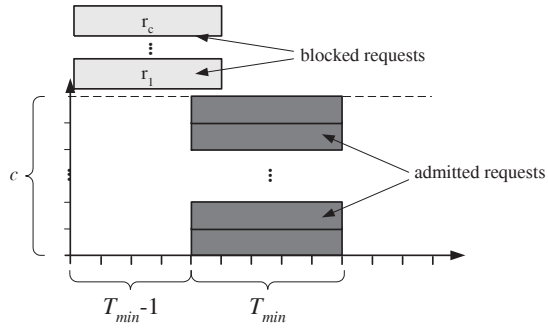


Figure 4.4: Worst-case request blocking: on a link with capacity c the resources allocated by the admitted requests block c requests. The unused time interval must be less or equal $T_{min} - 1$.

The number of request blocked as a consequence of the fragmentation can be determined as illustrated by the example shown in figure 4.4. For each request made in advance, two requests may be admitted, i.e., 2 instead of only 1 request could have been admitted.

Using these considerations, the upper bound for the requests blocked as a result of allowing advance reservations and the respective bandwidth can be determined as follows. W.l.o.g., let $t_r < T_{\min}$. Given a book-ahead interval T, link capacity c, and unit bandwidth requests, this means that for each request admitted, the amount of requests blocked within the interval T in the worst case can be determined as suggested in figure 4.4, which means

$$\mathrm{R}_{\mathrm{frag}} \leq c\frac{T}{2(T_{min} - 1)}. \tag{4.3}$$

Consequently, the upper bound of the bandwidth blocked as a consequence of the fragmentation ($\mathrm{B}_{\mathrm{frag}}$) is given by

$$\mathrm{B}_{\mathrm{frag}} \leq (T_{min} - 1)c\frac{T}{2(T_{min} - 1)} = c\frac{T}{2}. \tag{4.4}$$

Although these worst-case bounds imply the performance impact on both RBR and BBR is equally high, the simulations that will be presented in section 4.3.2 show that the impact on the RBR is actually very high while the BBR is far less affected. The reason for this behavior will be illustrated in section 4.2.4.

The Multiple Link Case

As described before, bandwidth is a bottleneck metric. In this sense, the worst-case fragmentation for a complete network is determined by the worst-case resource fragmentation at bottleneck links, i.e., links that are shared by many

flows with different source and destination nodes. Obviously, for each bottleneck link the above equations hold as well. The performance can neither get worse nor improve. Thus, the worst-case fragmentation in the network is determined by the worst-case fragmentation on the bottleneck links. Consequently, the previously shown equations also hold for a complete network.

4.2.4 Impact of Fragmentation

The worst case describes the influence of fragmentation onto the performance of the whole link or network and thus, is of interest for the operator of a network. Besides, it is also of interest to examine the impact of the fragmentation on other parameters, such as the average path length and individual requests. This examination will give a more detailed impression of the possible impact of an advance reservation service on the performance of a network.

The Single Link Case

When considering only a single link, the fragmentation leads to a reduction of the available bandwidth on that link. This is the same effect as in other application environments, such as node reservations in a single cluster computer. It will be shown that resource fragmentation mainly affects requests with high admission probability, this means requests which require only a low transmission rate.

Let $R = \{r_1, \ldots, r_n\}$ denote a set of n requests, and let $B = \{b_1, \ldots, b_k\}$ be the possible transmission rates associated with requests $r \in R$. W.l.o.g., assume that $b_i < b_j$ for $i < j$ and that the transmission rates are uniformly distributed among the requests $r \in R$. In an immediate reservation environment, the probability for a given request $r \in R$ to be admitted at time t_0 depends on its requested bandwidth b and the available capacity at time t_0.

Let $\mathbb{P}(B)$ denote the power set of B, let $\hat{B}_c \subseteq \mathbb{P}(B)$ be the sets of transmission rates with $\sum_{b_j \in B', B' \in \hat{B}_c} b_j \leq c$ and $b_1 + \sum_{b_j \in B', B' \in \hat{B}_c} b_j > c$. Informally, this means that $\hat{B}_c$ contains each set of requests that fills the capacity c such that no further request can be admitted. Furthermore, let $\hat{B}_c(b) = \{B' \in \hat{B}_c | b \in B'\}$ denote the sets that contain $b \in B$ at least once, and let $f_b : \hat{B}_c(b) \rightarrow \mathbb{N}$ denote the overall number of occurrences of b in $B' \in \hat{B}_c(b)$, i.e., $f_b(B') := |\{m \in B' | m = b\}|$. The transmission rates are uniformly distributed among the requests and hence, the mean value of $f_{b_i}, i \in \{1, \ldots, n\}$ depending on the available capacity c can be recursively determined using the following recursive formula:

$$(1) \quad E(f_{b_n}, c) = \left\lfloor \frac{c}{\sum_{1 \leq l \leq n} b_l} \right\rfloor$$

$$(2) \quad E(f_{b_i}, c) = E(f_{b_{i+1}}) + \left\lfloor \frac{c - \sum_{i < l \leq n} E(f_{b_l}) b_l}{\sum_{1 \leq l \leq i} b_l} \right\rfloor \quad , \text{with } i \in \{1, \ldots, n-1\}.$$

In the second equation, the first term of the right hand side denotes the mean value for the next larger transmission rate b_{i+1} and the second term denotes the mean value for f_{b_i} for the remaining capacity when subtracting the respective number of transmission rates $> b_i$.

Using this definition, the rejection probability $p_{\text{rej}}(b, c)$ of a request with transmission rate $b \in B$ on a link with capacity c can be determined as

$$p_{\text{rej}}(b, c) = 1 - \frac{1}{n} E(f_b, c).$$

W.l.o.g., assume b_i, b_{i+1} with $b_i < b_{i+1}$. This means $E(f_{b_i}) > E(f_{b_{i+1}})$, and therefore

$$p_{\text{rej}}(b_i, c) = 1 - \frac{1}{n} E(f_{b_i}, c) < 1 - \frac{1}{n} E(f_{b_{i+1}}, c) = p_{\text{rej}}(b_{i+1}, c). \tag{4.5}$$

Assume, the capacity c was reduced to c'. For simplicity, we define

$$e := \left\lfloor \frac{c - \sum_{i<l\leq n} E(f_{b_l}) b_l}{\sum_{1\leq l\leq i} b_l} \right\rfloor \text{ and } e' := \left\lfloor \frac{c' - \sum_{i<l\leq n} E(f_{b_l}) b_l}{\sum_{1\leq l\leq i} b_l} \right\rfloor,$$

denoting the second terms of $E(f_{b_i}, c)$ and $E(f_{b_i}, c')$, respectively. For $b_i, b_{i+1} \in B$ this means

$$\begin{aligned} E(f_{b_i}, c) &= E(f_{b_{i+1}}, c) - e \\ E(f_{b_i}, c') &= E(f_{b_{i+1}}, c') - e', \end{aligned}$$

and since $c > c'$ it can be concluded, that

$$e \geq e'.$$

With equation 4.5, the following inequality can be derived:

$$\begin{aligned} p_{\text{rej}}(b_i, c') - p_{\text{rej}}(b_i, c) &= 1 - \frac{1}{n} E(f_{b_i}, c') - \big(1 - \frac{1}{n} E(f_{b_i}, c)\big) \\ &= \frac{1}{n} \big(E(f_{b_{i+1}}, c) + e - (E(f_{b_{i+1}}, c') - e')\big) \\ &= \frac{1}{n} \big(E(f_{b_{i+1}}, c) - E(f_{b_{i+1}}, c') + \underbrace{e - e'}_{\geq 0}\big) \\ &\geq \frac{1}{n} \big(E(f_{b_{i+1}}, c) - E(f_{b_{i+1}}, c')\big) \\ &= p_{\text{rej}}(b_{i+1}, c') - p_{\text{rej}}(b_{i+1}, c). \end{aligned} \tag{4.6}$$

This inequality states, that the difference $p_{\text{rej}}(b, c') - p_{\text{rej}}(b, c)$ increases with the difference $|c - b|$. Informally, this means that blocking a certain amount of bandwidth capacity in general more likely affects those requests with low transmission rate. In contrast, large requests suffer only little from the capacity reduction since the admission probability of these requests is generally much lower. Thus, fragmentation - which is essentially a capacity reduction - more likely affects requests with low transmission rate.

Using similar calculations, this result can be generalized in a way, that requests which are more likely admitted also suffer more likely from the capacity reduction resulting from fragmentation.

The Multiple Link Case

In a network, the bandwidth reduction due to fragmentation on the individual links increases the path length for individual requests. The reason is that the common routing algorithms are solely based on variants of a shortest path algorithm [MS97b, WN02]. Using the shortest paths minimizes the number of hops and thus, generally avoids using too many hops which potentially blocks bandwidth for other requests.

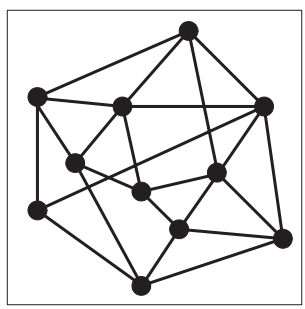
When the available bandwidth on the shortest possible paths exhausts resulting from fragmentation, paths with higher hop count must be used. Due the reduced link capacity, this happens earlier than in immediate reservation environments. The result is that the average path length computed over all admitted requests increases. Therefore, in a network with multiple source/ destination pairs the reduction of the performance results not only from the fragmentation on individual links but as a consequence also increases the path length used to route admitted requests.

Although in this work only bandwidth is considered, the larger average path lengths also show that the advance reservation service impacts the ability to guarantee delay constraints, which are heavily depending on the hop count of the routes.

4.3 Experimental Results

The equations shown previously mean that the RBR and BBR may increase considerably under the given conditions when all requests from a given set are defined in advance. Although this figure gives a hint on the amount of the possible performance degradation, this can only be a rough estimate of the actual impact of advance reservations on both performance metrics. Simulations were made in different network environments, showing how the performance is affected by allowing a varying percentage of requests to be advance reservations.

4.3.1 Simulation Environment



cost239

Figure 4.5: The network topology used for the subsequent simulations.

The simulations shown in this thesis were made in different environments using a total of seven network topologies. These topologies represent flavors of

networks with different properties, ranging from a simple single link network with only two nodes to a large backbone topology with 60 nodes and 110 links. Some of these topologies represent actual backbone networks, some are artificial topologies, e.g., a 6x6 two-dimensional grid, and some of them were generated using the BRITE network topology generator [MLMB01]. For the sake of simplicity and clarity of the document's organization, only the results for one of them, the *cost329* topology which represents an actual backbone [Bac], is given in here thesis (see figure 4.5). Since the most important performance aspects of advance reservations are similar, independent of the actual topology, selected performance results using the other topologies are given in the appendix.

In order to study the impact of advance reservations, the performance of advance reservations is compared to the performance achieved when allowing only immediate reservations. Since immediate reservations are just a special case of advance reservations without having a reservation time (see definition 4.1), the comparison was made as follows: a set of requests was generated with each request defined by its start and stop time, source and destination node, and the bandwidth requirement. In order to simulate advance reservations, each request was randomly assigned a reservation time.

It is yet unknown how realistic parameters for advance reservation requests look like. The environments considered here, e.g., grid computing, imply that mostly high bandwidth connections will be requested. Similarly, the requested duration (sometimes referred to as "call holding time") can be considered as longer than assumed in usual studies of current Internet traffic [Bol94]. These studies monitored todays Internet traffic consisting mainly of many short term transmissions such as web traffic. However, in scenarios such as grid computing with large amounts of bulk transfer requests or long-lasting video conferencing sessions, the average duration will be much longer. Therefore, the requested transmission duration was uniformly distributed in the interval $[100, 1000]$ and the transmission rates were uniformly distributed among 56 kb/s, 64 kb/s, 256 kb/s, 512 kb/s, 1024 kb/s, 2048 kb/s, 4096 kb/s, and 8192 kb/s. The request inter-arrival times of requests were assumed to follow the Poisson distribution as in [MS97b]. In [PF95], user initiated connection requests were found to be well-modeled using Poisson inter-arrival times. The reservation time of advance reservation requests was exponentially distributed with a mean of 200, unless explicitly stated otherwise.

4.3.2 Network Performance

The first performance evaluation was made using a mixture of immediate reservations with known duration and advance reservations. The route selection was made using an algorithm that determines the shortest feasible path for each request.

In figure 4.6, the RBR and BBR under three different load conditions are shown. The different load conditions were generated by modifying the mean inter-arrival time. In general, the performance decreases for both metrics in particular the RBR is affected while the BBR is only reduced by a few percent. It can be observed that both metrics decrease even with a low percentage of advance reservations.

The RBR for each of the two reservation types is depicted in figure 4.7. The admittance of advance reservation requests impacts especially the performance

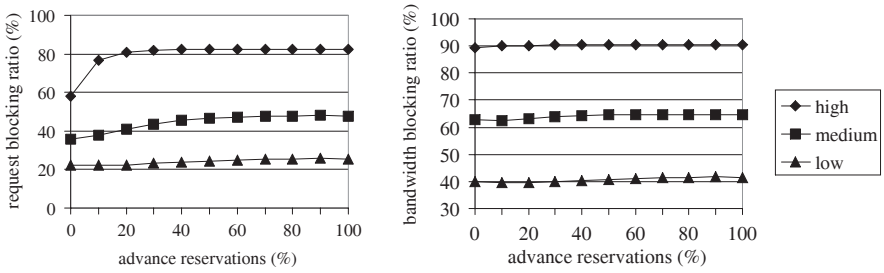


Figure 4.6: RBR (*left*) and BBR (*right*) for different percentages of advance reservations under three different load conditions.

of immediate reservations. This means, whenever advance and immediate reservations[2] share a single partition as is assumed here, immediate reservations suffer from constantly higher blocking rates, independent of the percentage of requests made in advance. Thus, it is not sensible to place immediate reservation requests in an advance reservation environment, as those requests are more likely being rejected.

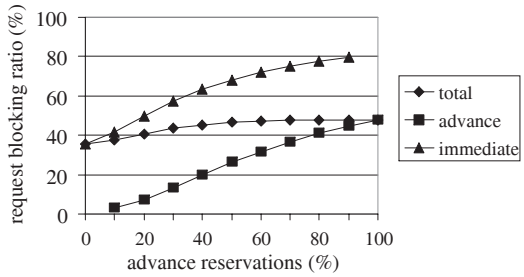


Figure 4.7: RBR for different percentages of advance reservations for both reservation types (medium load).

As discussed in section 4.2, the fragmentation is influenced by the variance of the reservation times. Thus, with higher variance the RBR is expected to increase. For example, when the reservation times are uniformly distributed in a given time interval, the variance of the reservation times increases with increasing interval size.

In figure 4.8, the relation of reservation time and RBR is depicted under three different load conditions. The curves are almost identical to those shown in figure 4.6. It can be clearly observed that, depending on the actual load, increasing the mean reservation time leads to a significant rise of the RBR until saturation is reached. Likewise, the BBR remains almost unaffected, similarly to the results depicted in figure 4.6.

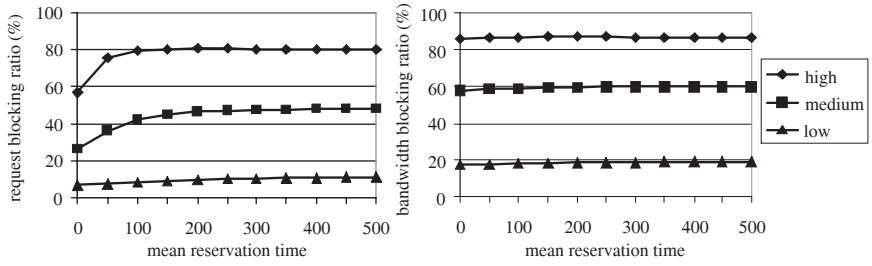


Figure 4.8: Dependency of the RBR (*left*) and BBR (*right*) on the reservation time.

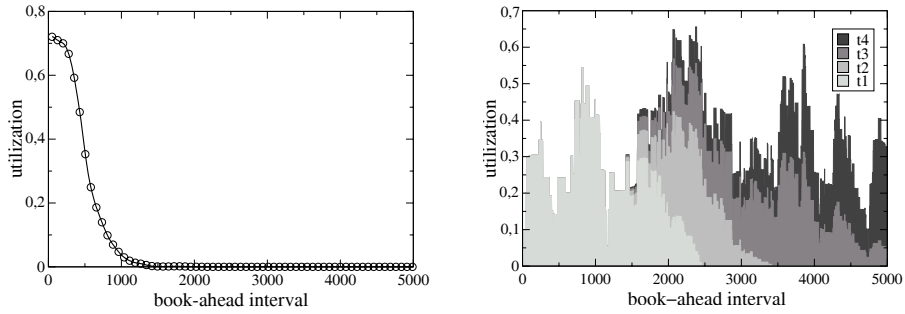


Figure 4.9: Book-ahead interval status on average (*left*) and status of a single link at four different points in time (*right*).

4.3.3 Link Utilization

In figure 4.9, the average book-ahead interval status in terms of utilization computed over all links and the exact status of a single link at four different times $t_1 < t_2 < t_3 < t_4$ is depicted. While the average shows virtually no difference to the status of links in immediate reservation environments, the utilization diagram of a single link illustrates the fragmentation resulting from advance requests. As more requests are admitted, at certain times the link utilization shows "peaks" which increase over time. When the gap between such peaks is too short for other reservations to fit in, bandwidth is blocked at this place.

4.3.4 Impact on Individual Flows

Blocking Probability

The most important property of advance reservations for network users is outlined in figure 4.10. With increasing reservation time, the blocking probability decreases and is reduced to zero at a certain point in time. Reservation requests for bandwidth beyond this point in time can be admitted with 100% admission probability. This property is independent of the distribution of request durations and reservation times (see also [WG98]). The point in time where the blocking probability is below a certain level, e.g., predefined threshold, was

[2]Immediate reservations with known duration.

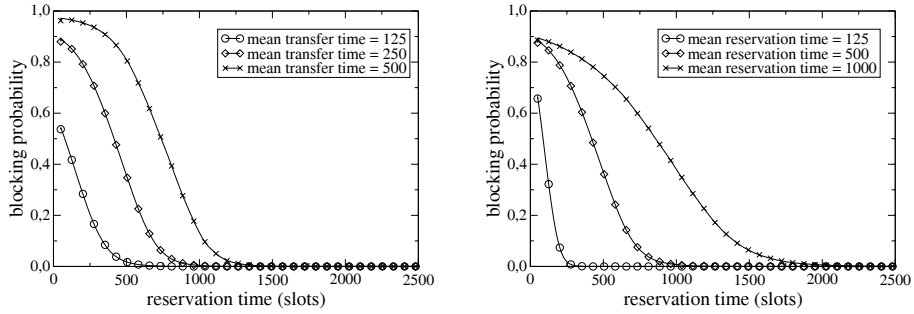


Figure 4.10: The blocking probability as a function of the mean transfer time (*left*) and the mean reservation time (*right*).

called *critical time* in previous examinations [WG98].

The blocking probability is influenced by the mean transfer time since this directly impacts the load on the network, i.e., the amount of traffic increases with higher transfer time. However, it is important to note that also the mean reservation time has an impact in this context. The higher the mean reservation time, the higher the blocking probability for flows, i.e., the critical time depends on the mean reservation time.

Typically, the drastically reduced rejection probability for advance reservations with sufficiently large reservation time is the most important reason for clients to employ advance reservations. In particular, when several different resources need to be reserved in a strictly coordinated fashion, i.e., timing constraints apply to the order and duration in which the resources are used, only the opportunity to reserve those resources well in advance provides the high degree of certainty required for the co-allocation. For example, in grid computing environments such co-allocations are necessary to guarantee to proper operational sequence of the whole system.

Impact of Fragmentation

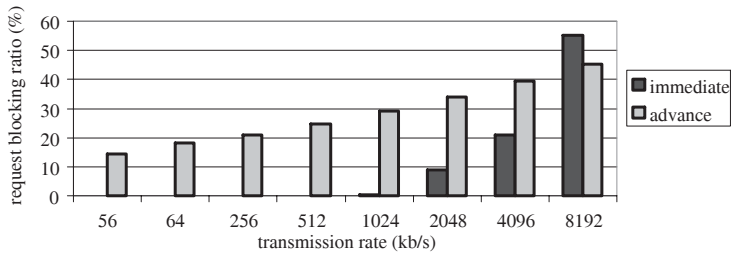


Figure 4.11: The blocking probability of advance and immediate reservations depending on the requested bandwidth under medium load.

The simulations described in section 4.3.2 showed, that using advance reservations mainly increases the RBR, although also the BBR is affected. This can be explained using the results of the analysis. Compared to the immediate reservation environment, requests with low transmission rate are more like being

rejected in advance reservation environments. Since these requests carry only a low percentage of the overall amount of traffic, the BBR is less affected by the performance degradation than the RBR. This analytical result is confirmed by the simulative results. In figure 4.11, the rejection probability depending on the requested transmission rate is depicted for advance and immediate requests. It can be observed that the immediate reservation scenario leads to a preference of those requests with low bandwidth requirement while the usage of advance reservations leads to higher fairness in the sense, that the rejection probability of requests with high bandwidth is decreased while those with lower bandwidth requirement are rejected more often.

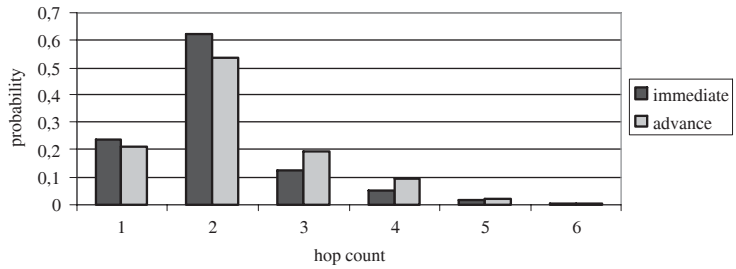


Figure 4.12: The distribution of path length of admitted requests.

Similarly, the distribution of the hop counts of admitted requests shows, that advance reservations significantly reduce the number of requests mapped onto short paths as outlined in figure 4.12. This results in a lower average path length in the immediate scenario, for the depicted situation this was 2.15 for immediate and 2.36 for advance reservations.

4.4 Conclusion

The important result of the discussion in this chapter is, that the main reason for the performance degradation in advance reservation environments is a fragmentation of the available resources. Furthermore, such a fragmentation is a very likely case and consequently, in order to improve the network performance measure must be taken that are able to avoid fragmentation. In particular, the flexibility permitting to define advanced services as described in section 3.2 can be used to achieve this goal. These issues will be discussed in the following chapter.

Chapter 5

Performance Optimizations

The performance of a networks degrades when advance reservations are allowed, as visible mainly by an increased RBR (see chapter 4). The advance reservation environment however provides mechanisms to overcome this problem. Some of these mechanisms are similar to those used in immediate reservation networks, i.e., these strategies only need to be adapted to fit the advance reservation environment. This is applied, for example, in the case of the routing algorithms. Other mechanisms are unique in the advance reservation environment and have their origin in the availability of additional information about future allocations. Using all those mechanisms, the performance achieved in immediate reservation environments can be reached and, depending on the actual setting, in some cases even outperformed. Thus, advance reservations are not only useful for customers and users of a network but can also help operators to improve their performance and revenue.

5.1 Routing

Routing represents the foremost measure to influence the performance of a network in best-effort or immediate reservation scenarios. Both on-line and off-line strategies have been proposed for that purpose. In the advance reservation environment, the routing strategy only serves as the foundation for further, more sophisticated techniques based on the availability of the temporal information in the advance reservation system.

The routing algorithm chosen for the measurements in the previous chapter was a simple shortest path strategy which does not use information about the future network load for more than the simple check for sufficient bandwidth. This algorithm is described in more detail in this section. In addition, a number of alternative routing algorithms are presented that make use of the knowledge about link utilization during the requested transmission in order to select paths. These routing strategies are used during admission control as described in section 3.4.2.

The theoretical worst-case performance bounds for an on-line advance reservation algorithm were shown in [AAP93], presenting the ROUTE_OR_BLOCK algorithm. In particular, it was proven that compared with the optimal off-line strategy, any on-line routing algorithm in the worst-case can only achieve

at most an $O(\log nT)$ share of the profit generated by the off-line strategy. In contrast to this approach, the on-line strategies presented in this section not necessarily reach the same competitiveness. However, as will be shown the ROUTE_OR_BLOCK algorithm cannot reach the performance of the other strategies which is a result of the conservative admission policy aimed on preferring requests that can be satisfied using only a few hops.

For the purpose of performing on-line admission control, two general approaches are conceivable that can be used in the given network management model, i.e., bandwidth brokers using MPLS explicit routing in order to enforce a particular routing strategy within a single domain. Firstly, it is possible to compute a path individually on-demand for each incoming request (*on-demand computation*), for example, using the ROUTE_OR_BLOCK algorithm. The second strategy is to use a set of precomputed paths for each pair of network end points and route a request on one of these paths with sufficient bandwidth (*precomputation*). The second approach is interesting in the MPLS environment. Although it is possible to bind each new flow to an individually computed path, which must be set up before the flow commences, it can be useful to predefine a set of paths using the corresponding MPLS mechanisms and set up these paths at start time of the system. Once a new flow starts, it remains to bind the flow to an existing path. This reduces the administrative overhead and the time required to prepare the network for a new flow. Both strategies may be combined, i.e., in case none of the precomputed paths has sufficient bandwidth to fulfill the request, a new path may be computed individually.

In this section, the basic approach to compute a feasible path for an advance reservation request is outlined and three variants of this algorithm are given. These variants use actual link utilization in order to weight paths and are based on existing strategies in the field of immediate reservation QoS routing. It will be shown, that these strategies improve the overall network performance in terms of RBR and BBR (see section 4.1).

5.1.1 On-Demand Path Computation

The on-demand path computation for each incoming request is based on an adaptation of Dijkstra's shortest path algorithm (DSP) to fit the advance reservation environment. However, it is also possible to use the Bellman-Ford algorithm for this purpose.

The computation of the link weight must include a check of the available bandwidth on each link during the requested transmission interval. The resulting algorithm *ASP* (*A*dvance reservation *S*hortest *P*ath), based on DSP as described in [Jun99], is outlined in the following:

```
ASP (G(V,E), s, d, t_start, t_stop, b)
1   foreach (v ∈ V) do
2      w[v] := ∞
3      φ[v] := NIL
4   done
5   T := V
6   w[s] := 0
7   while (T ≠ 0) do
8      find u ∈ T, such that w[u] is minimal
9      T := T\{u}
10     foreach (v ∈ T ∩ Adj[u]) do
11       if ((w[u] + weight(u, v, t_start, t_stop) < w[v]) and
12          (c(u, v, t_start, t_stop) > b))
13          then
14            w[v] := w[u] + weight(u, v, t_start, t_stop)
15            φ[v] := u
16       fi
17     done
18  done
```

In the algorithm, the array w contains the current node weights and ϕ is the array storing the actual path, i.e., predecessors of nodes. $c(u, v, t_{\text{start}}, t_{\text{stop}})$ (l. 12) denotes the available capacity on link (u, v) during the period from t_{start} and t_{stop}, i.e., the condition in line 12 assures, that sufficient bandwidth is available on the computed path. In contrast to immediate reservation scenarios, advance reservation architectures as discussed here allow to use information about the link load during the whole transmission period and to include this into the weight for each link. This is achieved using the function $\texttt{weight}(u, v, t_{\text{start}}, t_{\text{stop}})$ (l. 11).

Implementation Variants

Three variants are proposed here, using different implementations of the function `weight`. In the following, let $l(u, v, t)$ denote the allocated bandwidth at slot t on link (u, v), and let $c(u, v)$ denote the total bandwidth capacity of link (u, v).

- **Shortest-Path**: uses the hop count as the only path metric, i.e., the `weight` equals 1 for each link with sufficient bandwidth, and ∞ otherwise. In this case, the ASP algorithm computes the shortest available path. This variant was used to compute the performance results shown in section 4.3.2.
- **Widest/Shortest**: computes the widest among the paths with the shortest hop count. The algorithms includes the maximal available bandwidth - computed over all slots within the requested transmission interval - as link weight. This means, the weight is computed as

$$\texttt{weight}(u, v, t_{\text{start}}, t_{\text{stop}}) := \frac{1}{|V| c(u, v)} \max_{t_{\text{start}} \le t \le t_{\text{stop}}} \{l(u, v, t)\} + 1. \quad (5.1)$$

- **MinLoad/Shortest**: uses the average load during the requested transmission period as link weight. The path with the minimal average load is chosen by this strategy. The average load on (u, v) during the interval $[t_{\text{start}}, t_{\text{stop}}]$ is computed as

$$l_{\text{avg}}(u, v, t_{\text{start}}, t_{\text{stop}}) := \frac{1}{t_{\text{stop}} - t_{\text{start}}} \sum_{t_{\text{start}} \leq t \leq t_{\text{stop}}} l(u, v, t).$$

  The weight for each link is then determined as

$$\texttt{weight}(u, v, t_{\text{start}}, t_{\text{stop}}) = \frac{1}{|V| c(u, v)} l_{\text{avg}}(u, v, t_{\text{start}}, t_{\text{stop}}) + 1. \quad (5.2)$$

For both **Widest/Shortest** and **MinLoad/Shortest**, the additional factor $1/|V|$ assures that always one of the paths with the lowest hop count is selected, i.e., that the addend 1 dominates the weight of each link. Thus, the link utilization is only used as an additional metric in case several paths with the same hop count exist.

The correctness of the algorithms is shown in the following proposition.

Proposition 5.1 (Correctness of the algorithms) *The routing algorithms **Shortest-Path**, **Widest/Shortest**, and **MinLoad/Shortest** work correctly. This means, that **Shortest-Path** computes the shortest available path, and **Widest/Shortest** and **MinLoad/Shortest** select the widest path respectively the path with the minimal load among the set of shortest available paths with equal hop count.*

Proof. 1. **Shortest-Path** works correctly since the link weight is always nonnegative.

2. **Widest/Shortest** works correctly, if it always selects the shortest available path and in case several such shortest paths are available, it selects always the widest among them. The second property results directly from the definition of the link weights. In order to prove the first property, let p_{opt} denote the shortest available path with $|p_{\text{opt}}|$ being its hop count, and let $\texttt{weight}(p_{\text{opt}})$ denote the weight of p_{opt} according to eq. 5.1. Furthermore let $\omega_{\max} := \max_{(u,v) \in p_{\text{opt}}} \{\texttt{weight}(u, v, t_{\text{start}}, t_{\text{stop}})\}$.

   Assume there exists a path p with higher hop count, i.e., $|p| > |p_{\text{opt}}|$ but lower weight (w.l.o.g. $\texttt{weight}(p) = |p|$, i.e., $l(u, v, t) = 0, \forall t$), i.e.,

$$\begin{aligned} \texttt{weight}(p) &< \texttt{weight}(p_{\text{opt}}) \\ \Rightarrow |p| &< |p_{\text{opt}}| \omega_{\max} + |p_{\text{opt}}|. \end{aligned}$$

   Since $|p_{\text{opt}}| \omega_{\max} < 1$, this leads to

$$\begin{aligned} |p| &< |p_{\text{opt}}| + 1 \\ \Rightarrow |p| &\leq |p_{\text{opt}}|. \end{aligned}$$

This contradicts the assumption, that $|p| > |p_{\text{opt}}|$, and therefore such a path cannot be computed using **Widest/Shortest**, i.e., the algorithm works correctly. The correctness of **MinLoad/Shortest** can be shown using the same method. □

The complexity of the above algorithms is $O(t_d|E|\log|V|)$ with $t_d = t_{\text{stop}} - t_{\text{start}}$ being the duration of a given request. This can be directly derived from the complexity of the original algorithm by Dijkstra, when T is implemented as a heap, and the fact, that the computation of the link weight in any case can be performed in $O(t_d)$ time when using arrays (see also chapter 7).

The algorithms are derived from the QoS routing algorithms described and compared in [MS97a, MS97b] where they were implemented and compared in immediate reservation environments. In such a scenario, the computation of the link weights can take the actual utilization on the links into account, however this is only possible for the current time rather than the whole transmission duration.

With respect to the resource fragmentation, the **Widest/Shortest** approach, which uses the peak bandwidth utilization on each link, can avoid peaks as long as possible by distributing load onto several alternative paths with the same hop count. Besides avoiding that a single path fills up too soon which is its main achievement in immediate reservation environments, this simple method is very effective in avoiding peaks during the requested transmission period. The **MinLoad/Shortest** approach yields slightly worse performance since the peaks only contribute partially to the average load on a given link. This will be more detailed described in section 5.4.1.

Path Caching

When the algorithms described in this section are applied in a bandwidth broker, i.e., an on-demand path computation approach is used, the time required to compute a feasible path can be reduced by caching computed paths. Each path computed during the on-line admission control process is stored in a database which is consulted when a new request enters the system. The stored path set can then be checked for sufficient bandwidth and when possible a path can be selected using the metrics previously described in order to determine, for example, the **Widest/Shortest** path from the set.

When using the hop count as primary metric, this caching approach does not necessarily result in paths consistent with the metric. Although the paths stored in the database are actually shortest paths, it may be possible that more paths with the same hop count exist than stored in the cache. This obstructs, for example, the **Widest/Shortest** approach. Thus, paths from the cache of a given hop count l may only be used, when at least one path with hop count $l+1$ is also stored in the cache. In this case, paths with hop count $\leq l$ are called *valid* paths and can be used. Among those valid paths, one is selected according to the chosen metric, i.e., **Widest/Shortest**, **MinLoad/Shortest** etc. In any other case, i.e., if no path with sufficient bandwidth is available from the set of valid paths, the on-demand path computation must be involved in order to try and find a suitable path.

5.1.2 Other Routing Approaches

The algorithms presented in this section only represent a selection of implementation alternatives considered to be reasonable in the advance reservation environment. Although other implementation alternatives are conceivable, studies of QoS routing algorithms in immediate reservation environments [MSZ96, MS97a, MS97b, WN02] have shown that using other than shortest path algorithms generally results in performance degradation. The reason is that in scenarios with high load, which are the only scenarios where QoS mechanisms can be reasonably applied, it must be assured that not too many links are blocked by traffic between two network end-points. This is a general problem of QoS routing and hence, selecting only shortest paths is a successful and also easy to compute metric for on-line QoS routing strategies.

The previously shown techniques can be combined with others, for example, which restrict the path length either dynamically depending on the load situation or statically to some predefined maximum hop count. An example for the latter approach is the *dynamic-alternative* routing algorithm described in [MS97a, MS97b] which is basically a **Widest/Shortest** algorithm, allowing only paths with a maximal hop count of $|p_{\text{opt}}| + 1$ where $|p_{\text{opt}}|$ denotes the hop count of the shortest distance path. Likewise, many other techniques were developed in order to restrict path lengths, such as multiplying a constant factor to each link weight with increasing path length [WN02]. Since immediate reservations are only a special case of advance reservations, these techniques can also be applied in the advance reservation environment.

5.1.3 Path Precomputation

The precomputation of a set of alternative paths for each pair of network end-points is another opportunity to perform admission control. The general idea is to compute a fixed set of alternative paths for each pair of network end-points. This can be used to predefined a set of paths in the MPLS environments and as long as these path sets are used, each new flow only needs to be bound to one of those paths. On the other hand, the disadvantage is that feasible paths exists which are not contained in the precomputed path set. Therefore, it is possible that requests are rejected although a path with sufficient bandwidth exists. Furthermore, the advantage that a large number of alternative paths is available and therefore the path computation can be avoided, can also be achieved using the path caching approach which also results in a larger number of available paths after an initial start-up phase. Hence, both precomputation and path caching very likely lead to similar results in terms of processing time. As will be shown in chapter 7, routing only requires about 8% of the processing time of a single request and hence, the impact of the precomputation on the processing time can be considered as rather limited.

Implementation Alternatives

Several opportunities exist to compute the path set. In [LG01], an algorithm was proposed to compute a set of *maximally disjoint* paths for load-balancing and fault tolerance purposes, based on Bellman-Ford for computing shortest paths. The algorithm is repeatedly executed in a sequence of steps. In each

step, a new path is added to the existing path set. Furthermore, each iteration modifies the existing link weights in a way that more frequently used links are assigned higher weights. The way this link weight is computed determines the "disjointedness" of the resulting paths. Using high link weights in this case leads to paths which have fewer links in common and vice-versa.

The drawback of the maximally disjoint path computation is the relatively small set of alternative paths computed by the algorithm which leads to a sub-optimal network utilization and high request rejection rates. Therefore, in [BDF03] a set of k paths was used which allows to determine the parameter k depending on the network topology and also on the actual load situation in the network. The k shortest path computation was made using the algorithm described in [Epp98]. These k paths can be also weighted according to the **Widest/Shortest** and **MinLoad/Shortest** strategies as described before.

This approach has the interesting advantage that the computed path set can be set up on the MPLS network at startup and only requires a binding process of label to flow when a transmission starts. Instead, the individual path computation in the worst case requires to set up a new path each time a new transmission starts. As described in section 2.1.3, not only the signaling between bandwidth broker and edge node is required but also the label binding process using CR-LDP or RSVP-TE. In addition to the reduced signaling overhead, the precomputation approach has the advantage of requiring less computation time during admission control since only a check of the links has to be performed instead of also requiring to compute a complete path. This makes this approach interesting for the rerouting process in case of link failures (see chapter 6).

The drawback of the path precomputation of a limited set of paths is that possibly not the whole variety of feasible paths can be exploited. This is particularly true for the maximally disjoint path computation. In contrast, a study of both approaches [Bur03b] showed, that the number of requests rejected although sufficient bandwidth was available, is lower using the k path approach. This holds even with relatively low values of k. In contrast, the maximally disjoint approach does not provide a sufficiently large path set and results in much worse rejection rates. Unless it can be guaranteed, that the precomputation strategy determines any possible path, this approach has the drawback of possibly rejecting many flows although sufficient bandwidth is available.

The strategies **Widest/Shortest** and **MinLoad/Shortest** can also be applied when using a precomputation based routing strategy [Bur03b]. In general, the precomputation approach only reduces the admission speed and simplifies the route setup but does not allow to apply any other mechanisms that are more suited in the advance reservation environment than the on-demand computation. Hence, the performance evaluation only discussed the on-demand path computation. As shown in [Bur03b], the precomputation yields similar results.

Other Approaches

The precomputation reduces the computational effort for the admission control process, however still requires to potentially check the whole set of paths. Instead, it is conceivable to precompute not only the possible path set but also paths which can be used to carry a certain amount of bandwidth as proposed in [OS00]. However, this requires to compute feasible paths for any possible bandwidth requirement not only for a single point in time as in [OS00], but

includes the complete temporal dimension of the advance reservation problem, i.e., the whole book-ahead period. Consequently, this results in a much higher computation time, in particular in the advance reservation environment, and therefore, this approach was not considered here.

5.1.4 Flow Switching

Based on the previously described on-line routing strategies, which only consider the transmission of a flow on a single path during the flow's transmission interval, in advance reservation environments it is possible to also use multiple paths during their lifetime as discussed in section 3.3.3 and switch flows at predefined times onto different paths. While this *flow switching* is generally possible also in immediate reservation environments, the advance reservation perspective allows to plan these transmissions in advance, i.e., during admission control. It is important to note, that the transmission of a flow using more than a single path does *not* mean a parallel transmission but denotes the usage of more than one path throughout the lifetime of a flow, but only one path at a time.

Flow switching can be implemented transparently for the sender and receiver of a given flow by switching paths during run-time. However, in case other than bandwidth constraints must be obeyed, e.g., delay, flow switching may be infeasible unless it can be guaranteed that the delay bounds are satisfied by any of the multiple paths. When knowledge about node delay is available to the bandwidth broker this can be included in the admission decision. The routing algorithms presented in the previous sections are sufficiently flexible to allow the implementation of also such constraints. The only restriction in this case is, that QoS routing with multiple constraints can become NP-complete [WC96] and hence, computationally infeasible.

Routing Granularity

The granularity of the routing decision defines the length of the time interval a particular flows remains on a single path. This granularity also determines the computational, protocol, and administrative overhead of the flow switching approach. In general, two solutions can be conceived:

1. **Fixed granularity**: division of the transmission duration into smaller parts of fixed duration r. Each part is routed individually.

2. **Variable granularity**: determination of the granularity depending on the actual traffic situation during the transmission period. This strategy is activated only when the bandwidth is not sufficient to carry the request on a single path and can be implemented such that the longest feasible transmission interval is determined. If the available resources do not suffice to transmit the requested flow completely on a single path, i.e., the interval is shorter than the requested duration, the approach is to search for a new

path for the remaining duration, which in turn also may be unavailable for the whole remaining duration.

Fixed Granularity

Using this approach, the overhead for switching flows to new paths and for the admission control procedure computation required is always proportional to t_d/r, where t_d denotes the duration of a request and r is the granularity parameter denoting the length of the transmission interval for which a flow is routed on a path. However, the path computation can be made using one of the algorithms described in section 5.1, and since the total amount of slots to be checked remains $t_d = t_{\text{stop}} - t_{\text{start}}$, the complexity of the admission control algorithm does not differ from the single path case and is given by

$$O\Big(t_d\big(|E|\log|V|\big)\Big).$$

The computational complexity does not reflect the overhead required for the multiple invocations of the routing algorithm and the protocol overhead for switching flows to different paths. The implementation of such an approach however requires constant overhead which is known in advance, i.e., it can be statically determined whether the overhead is affordable for a given granularity r.

Variable Granularity

The overhead of the variable granularity approach depends on the dynamic load situation of the network resources during the requested transmission interval. This means, in case sufficient bandwidth is available, a given request will be transmitted on a single path. Otherwise, the length of an interval for which a flow remains on a path may be anything between 1 slot in the worst case and t_d $(= t_{\text{stop}} - t_{\text{start}})$ slots. Although the actual length of these intervals cannot be anticipated in advance, it is possible to restrict their length, i.e., to reject requests which would require to switch a single flow within too short time intervals.

Informally, the idea of the admission control algorithm is to determine the longest available transmission interval on each link for a given request of bandwidth b. In a series of computations, for each of the computed intervals a path is searched in the corresponding network consisting only of links which provide sufficient bandwidth. In each step, new links are added until eventually a path is found or all links are included without finding a feasible path. The following algorithm can be used to determine a minimal number of paths required for transmitting a given request $r = (s, d, t_{\text{start}}, t_{\text{stop}}, b)$.

```
FlowSwitching(G(V,E), r)
1    t := t_start
2    success := false
3    while ((success=false) ∧ (t < t_stop)) do
4       L := ∅
5       foreach (e ∈ E) do
6         l := length of the interval on e with capacity ≥ b, starting at t
7         insert l in L
8       od
9       while (L ≠ ∅) do
10        find maximal l ∈ L
11        L := L\{l}
12        E' := {e ∈ E| longest feasible interval on e is at least l}
13        p := DSP(G(E', V), s, d, t, t + l, b)
14        if (p =valid) then
15          if (t + l = t_stop) then success := true
16            else t := t + l
17          fi
18        fi
19      od
20   od
```

With $t_d := t_{\text{stop}} - t_{\text{start}}$, the outer `while` loop (l. 3) requires at most t_d iterations, the `for` loop (l. 5) requires exactly $|E|$ iterations, $O(t_d)$ steps required for scanning the links, and $O(\log |E|)$ steps to update the sorted list L (l. 7). The inner `while` loop (l. 9) requires at most $|L| \leq |E|$ iterations, each with complexity $O(|E| \log |V|)$ which is determined by the complexity of the DSP algorithm. The maximum search in line 10 can be done in $O(1)$ when L is sorted. Thus, the overall complexity is given by

$$O\Big(t_d|E|\big(\underbrace{|t_d| + \log|E|}_{\text{for loop (l. 5)}} + \underbrace{|E|\log|V|}_{\text{while loop (l. 9)}}\big)\Big).$$

Hence, the worst-case complexity of the variable granularity flow switching algorithm is higher than of the fixed granularity algorithm. However, as the results of the simulation will show, the actual number of path switches for each transmission is negligible and can be afforded. Since the above algorithm switches flows only in case of insufficient bandwidth, for most admitted requests the actual processing time is tolerable and the average admission speed is better than for the fixed granularity approach with small values of r.

A threshold parameter, defining the *critical interval length* which denotes the minimal length a flows must stay on a single path, can be integrated into the admission control algorithm, such that requests are rejected when the parameter l (l. 6) drops below the threshold.

5.2 Malleable Reservations

In chapter 3, the additional services that can be defined in an advance reservation environment have been specified, and among them the concept of malleable reservations in the environment of computer networks was outlined. The service in general determines suitable parameters for a transmission of given size, i.e., the amount of bytes to be transmitted. Based on that, it is possible to restrict the parameters of a given request, for example, by defining a deadline or requesting the earliest possible transmission start.

Malleable reservations may be used to achieve two different goals. Firstly, the fragmentation can be avoided as long as possible by placing malleable requests such that peaks do not occur. Secondly, in a situation where peaks have already occurred, malleable requests may be placed such that the gap between two peaks is filled as good as possible.

5.2.1 Properties

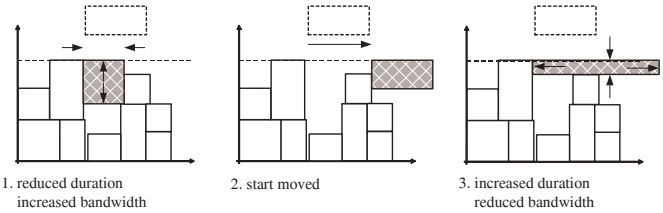


Figure 5.1: Examples how malleable reservations can permit the bandwidth broker to admit a request that would otherwise be rejected.

In figure 5.1, three examples are given which show how a rejected request (denoted by the dotted rectangle, see also figure 4.2) can be admitted when being defined as malleable. As previously described, the actual start and stop time and the bandwidth are not fixed but can be chosen by the network management within a certain range. This range must be defined by the client who issues a request. Alike job scheduling, the duration of malleable reservations can be defined as a function of the bandwidth it is allotted to. Once admitted, the bandwidth broker guarantees start and stop times, and transmission rate.

In general, enabling the mechanisms to cope also with variable transmission rates can be expected to achieve the highest performance gain and can be implemented, for example, by combining the flow switching mechanisms with those for malleable reservations. Moreover, allowing transmissions to be interrupted provides the highest amount of flexibility in terms of ”filling gaps”, i.e., utilizing resources likely to remain unused since the surrounding gaps are too short. However, in this thesis only requests are considered where the transmission rate does not vary throughout the transmission. This is an important difference between the approach discussed here and those in the field of job scheduling. The main reason is, that the admission control problem for searching the first feasible interval [GO00] in case of varying transmission rate can become NP-complete, and the question of providing accurate feedback about the variable transmission rate to the applications or users involved in the transmission is difficult to answer.

Furthermore, in the most extreme case, the transmission rate may also be zero during the transmission. Such an interruption of the transmission may not be desired since it may interfere with the transmission protocols. For example, interrupting an FTP session can mean that the transmission has to be started at the beginning. Instead, the considerations presented here are restricted to providing the parameters start and stop time together with the fixed transmission rate to the client in the response message to the request. This allows to seamlessly integrate malleable reservations into existing network management environments and simplifies admission control and notification to clients. It is also necessary for transmissions that rely on a fixed rate such as video streams.

An advance reservation request was defined in 3.1. In order to distinguish such requests from malleable requests, they are called *fixed requests*. In an advance reservation environment, the additional parameters to be submitted with a request defined as malleable are the minimal and maximal transmission rate, and the first start time and deadline, respectively.

Definition 5.1 (Malleable Request) *A malleable request is defined as* $r_m = (s, d, t_{\min}, t_{\max}, d_{\min}, d_{\max}, \bar{b})$*, where* $t_{\min}$ *and* $t_{\max}$ *denote the earliest start respectively the latest stop time,* $d_{\min}$ *and* $d_{\max}$ *denote the minimal and maximal duration, and* $\bar{b}$ *denotes the total amount of bytes to be transmitted. In addition, a maximal and minimal transmission rate may be specified, where the maximal transmission rate may directly depend on the capabilities of the sender and receiver.*

Duration and transmission rate are directly related when a fixed amount of bytes must be transmitted and therefore, instead of defining $d_{\min}$ and $d_{\max}$, it is also possible to provide the minimal and maximal transmission rate $b_{\min}$ and $b_{\max}$, respectively. This does not impact the admission control algorithm.

Instead of requesting a fixed amount of bytes to be transmitted, a malleable request can also be used to search the first available transmission interval or a suitable transmission interval up to a predefined deadline for a request with fixed duration and transmission rate. In such a case, the parameters used in definition 5.1 must be changed accordingly. Again, the general admission control algorithm can remain unchanged.

5.2.2 Admission Control

Admission control for malleable reservations requires to find a path with sufficient bandwidth such that the transmission can be made with parameters within the requested boundaries. As described in section 3.3.4, the search space for admission control of malleable requests covers all three dimensions of the general problem space. The admission control algorithm for this reservation type generally works as follows: a linear scan is performed over the available time interval $[t_{\min}, t_{\max}]$ where the duration of the request is adjusted within the given boundaries $d_{\min}$ and $d_{\max}$. Once a feasible path is found, the algorithm stops and returns the parameters of the request, i.e., start time, stop time, and transmission rate. Using this approach, the first dimension searched is the temporal dimension. Although in general the order in which the three dimensions are searched can be arbitrarily chosen, using the temporal dimension allows for an easy implementation of the interval search (see section 3.2.1).

The previously described approach selects the first feasible path. An implementation alternative is to choose the "best" of the whole set of feasible paths. In this case, it remains to define what "best" path in this context means. For that purpose, in the following an approach is presented that selects the path with the fewest hops and most remaining bandwidth, similar to the **Widest/Shortest** routing algorithm.

Before discussing the details of the different opportunities to select suitable transmission parameters, the general admission control framework is outlined. For a malleable request r_m the basic admission control algorithm can be implemented as follows:

```
Mall(G(V,E), r_m)
1   foreach (t_d ∈ [d_min, d_max]) do
2     b := b̂ / t_d
3     t_start := t_min
4     ∀e ∈ E : t*(e) := t_start, t*_d(e) := 0
5     while (t_start ≤ t_max − t_d) do
6       foreach (e ∈ E, with t_start ≥ t*(e) + t*_d(e), t*(e) ≠ −1) do
7         T := {t | t ≥ t_start ∧ ∀t' ∈ [t, t + t_d] : c(e, t') ≥ b}
8         if (T ≠ ∅) then
9           t*(e) := min{t ∈ T}
10          t*_d(e) := max{t | ∀t' ∈ [t*(e), t*(e) + t] : c(e, t') ≥ b}
11        else
12          t*(e) := −1
13      done
14      E' := {e | t*(e) ≠ −1 ∧ t*(e) ≤ t_start ∧ t*(e) + t*_d(e) ≥ t_start + t_d}
15      p := DSP(G(V, E'), s, d)
16      if (p =valid) then
17        break; // success:  path p found
18      else
19        t_start := max{t_start + 1, min{t*(e) | e ∈ E \ E', t*(e) ≠ −1}}
20      fi
21    done
22  done
```

The algorithm takes as input the network graph $G(V, E)$ and the request r_m. A scan across the interval $[t_{\min}, t_{\max} - t_d]$ is performed for $t_d \in [d_{\min}, d_{\max}]$. For any position t_{start} within this interval, the DSP algorithm is used to determine whether a path from u to v with sufficient bandwidth exists until such path is found. The index $t^*(e)$ denotes the start time for the first interval with sufficient bandwidth, where $c(e, t)$ denotes the available bandwidth capacity on link e at time t, and a length of at least t_d, with $t^*_d(e)$ (line 10) denoting the length of this interval. The computations in line 9 and 10 are implemented such that the whole interval $[t_{\min}, t_{\max}]$ is only scanned once for each duration t_d and each link e. The usage of these indices guarantees that the DSP algorithm is only performed on the network consisting of links from the set E' (line 14), which contains the links with sufficient bandwidth during the interval $[t_{\text{start}}, t_{\text{start}} + t_d]$.

In the algorithm, DSP denotes Dijkstra's shortest path algorithm, which determines the shortest path from s to d in the network given by $G(V, E)$. DSP can be used instead of ASP because the bandwidth availability was checked in the lines before. If the path search fails, the new index t_{start} is chosen from the indices $t^*(e)$ of those nodes e which were not included in the set E'. It is possible to use the also the other algorithms than DSP as show in section 5.1.1 in order to select other than the shortest paths.

Admission control for malleable reservations can be performed in polynomial time using the algorithm previously outlined. It takes at most $d_{\max} - d_{\min}$ cycles of the outermost loop (line 1), $t_{\max} - d_{\min} - t_{\min}$ cycles of the loop in line 5, and for any t_{start} there is a call to DSP with complexity $O(|E| \log |V|)$. The computation of the indices $t^*(e)$ and $t_d^*(e), e \in E$, guarantees that for each link the interval $[t_{\min}, t_{\max}]$ is only scanned once. This means there are at most $t_{\max} - d_{\min} - t_{\min}$ calls of DSP. Hence, the overall complexity of the algorithm is given by:

$$O\big((d_{\max} - d_{\min})(t_{\max} - d_{\min} - t_{\min})|E| \log |V|\big)$$

The complexity of admission control algorithm for malleable reservations is considerably higher than the basic routing algorithm, in particular the duration factor $t_d := t_{\text{stop}} - t_{\text{start}}$ is replaced by the length of the search interval and an additional factor $d_{\max} - d_{\min}$ is introduced which reflects the range of possible durations. Since in a realistic setting the search interval may not be much longer than the duration parameter in the fixed reservation scenario, the factor $d_{\max} - d_{\min}$ dominates the additional complexity of the malleable admission control process. For example, a request with a duration parameter $d_{\min} = 100$ and $d_{\max} = 1000$ may require a computation time which is 900 times higher than needed for processing a similar request with fixed parameters. This represents a substantially higher admission time which has be taken into account when implementing such a service, in particular processing a single request may require several seconds. In chapter 7, this aspect will be examined more thoroughly.

First-Fit

The previously outlined algorithm implements a first-fit approach, selecting the first available position and bandwidth. For malleable reservations, different variants of the algorithm are conceivable, depending on the way t_d (line 1) and t_{start} (line 3) are initialized. Four variants of the algorithm are proposed in the following:

- **Max/Start**: starts the scan with duration $t_d = d_{\max}$ at slot $t_{\text{start}} = t_{\min}$.
- **Max/End**: starts the scan with $t_d = d_{\max}$ at slot $t_{\text{start}} = t_{\max}$.
- **Min/Start**: starts the scan with $t_d = d_{\min}$ at slot $t_{\text{start}} = t_{\min}$.
- **Min/End**: starts the scan with $t_d = d_{\min}$ at slot $t_{\text{start}} = t_{\max}$.

The **Min/End** and **Max/End** variants require the scan to start from the end of the search interval. For this purpose, the computation of $t^*(e)$ and $t_d^*(e)$ (line 9,10) must be changed accordingly.

These strategies can also be used to schedule transmissions with fixed deadline or the earliest suitable interval for a transmission. In those cases, the duration t_d remains fixed. In particular, the **Min/Start** and **Max/Start** versions of the algorithm implement the search for the first available transmission interval with fixed transmission rate.

Best-Fit

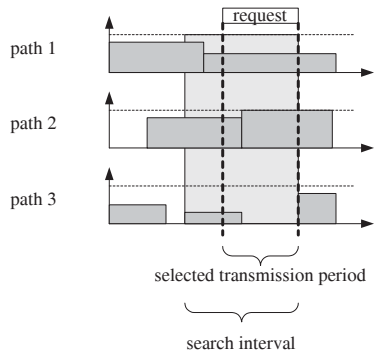


Figure 5.2: **Widest-Interval** strategy: setting with three alternative paths. The request is allotted to path 3 at the end of the search interval since this provides the lowest average utilization during the whole search interval.

Choosing the first available transmission interval and rate may not always result in a good performance. Thus, in addition to the first-fit strategies, another conceivable approach is to select a path according to a metric as used for the routing algorithms in section 5.1.1. This means, the best path and transmission rate according to some metric is selected. For example, the **Widest/Shortest** strategy performs rather successful and its strategy can be adapted for the malleable environment. Based on this observation, an additional scheduling strategy called **Widest-Interval** is proposed. Among any feasible parameter setting, this strategy chooses the path, transmission interval, and transmission rate which assures that the remaining available bandwidth on the path during the transmission interval is maximized (see figure 5.2). The **Widest-Interval** strategy can be implemented by modifying two lines of the above algorithm as follows:

$\texttt{Mall}_{\texttt{Widest}}(G(V,E), r_m)$
...
3 $t_{\text{start}} := t_{\min}; \quad c^* := \infty$
...
16 `if` $\big((p = \texttt{valid}) \wedge (c(p) > c^*)\big)$ `then`
17 $P := p; \quad c^* := c(p)$ `// save path`

where P denotes the currently best path, i.e., the path with the maximal amount of unassigned bandwidth, the variable c^* denotes the amount of unas-

signed bandwidth on P, and $c(p)$ denotes the available bandwidth on a path p.

Its worst-case complexity is the same as for the other variants, however in any case the whole interval size is checked which increases the run-time for accepted requests. The nature of the **Widest-Interval** strategy implies that its run-time can be considerably higher compared to the other, first-fit strategies.

The **Widest-Interval** approach may be restricted such that the scan is terminated after a predefined sequence of scans. For example, the scan may be terminated after the temporal dimension was scanned once using the lowest allowed transmission rate and a feasible path was found with this transmission rate. The rationale behind this approach is that using the lowest possible transmission rate very likely results in a high remaining bandwidth on the respective links.

Path Length Restriction

In order to restrict the path length generated by the previously outlined algorithm, the decision to accept a path can depend on the actual length of the path. The following extension to the previous algorithm outlines this idea.

```
16      if ((p = valid) ∧ (c(p) > c*) ∧ (|p| ≤ l*)) then
17          P := p;   c* := c(p) // save path
```

The hop count of a path p is denoted by $|p|$ and can be restricted using the third inequality in line 16. The parameter l^* determines the longest allowed path and, e.g., may be chosen by the operator. For example, when $\hat{l}$ denotes the hop count of the shortest path, l^* may be set to $\hat{l} + k$, with k depending on the network topology. Thus, a strategy like the *dynamic-alternative* (see section 5.1.2) can be implemented.

Interval Search

Depending on the actual application, instead of defining the total amount of bytes $\bar{b}$, it is possible to define other properties for the respective transmission from which the required parameters are then computed by the network management system. For example, for a video streaming application, a maximal bandwidth $b_{\max}$ for the video stream and a fixed duration d might be given. The required parameters then can be computed as $d = d_{\min} = d_{\max}$ and $\bar{b} = b_{\max} \cdot d_{\max}$. The task of the admission control algorithm then remains to search for a suitable transmission interval.

5.3 Off-line Optimization

The mechanisms described before were all applied during the on-line admission control procedure. Since the knowledge of the admission control at the time of the admission control is limited, i.e., only information about the current and past requests is available, this can lead to routing decisions that are suboptimal. This

means, routes block the available bandwidth that is required of other requests (see also figure 3.12). In those cases, if the opportunity exists to revise the on-line decisions, the performance of the network may be increased, i.e., more requests can be admitted.

As described in section 3.3.5, when implementing such off-line optimization mechanisms in immediate reservation environments, the difficulty is that no information is available on the stability of current traffic demand. The off-line optimization takes a certain amount of time and, when it is finished and routes are updated accordingly, rapid changes in the traffic situation in the network diminish the benefit of rerouting. The reason is that the time required to compute the off-line optimization may well exceed the period of stability.

Hence, those optimization mechanisms can only be applied when a certain stability of the traffic can be guaranteed. In contrast, the properties of advance reservation systems provide a better environment to apply such techniques. The flows present within the whole book-ahead interval can be included in the off-line optimization. Thus, the time available in order to optimize the network routes is much higher and therefore the application environment is much better suited than the one of immediate reservation networks. In the following, the solution to a multi-commodity flow problem is used to determine unbalanced network situations and to remap flows onto the topology. The optimization strategy was taken from [BKL03], where the approach was to optimize the route layout in an immediate reservation or best-effort network environment. However, the advance reservation environment is much better suited to support the off-line optimization, since the route changes can be made only in the internal database without the need to influence routes of active flows, i.e., interaction with the network devices. As a consequence, additional protocol overhead can be completely avoided. Furthermore, the available time for the off-line optimization, which is a critical factor, is much longer due to the length of the book-ahead interval.

5.3.1 Optimization Framework

The Maximum Concurrent Flow Problem

In order to optimize the routing on the network, algorithms solving multi-commodity flow problem are used. In particular, we deal with the *maximum concurrent flow* (MCF) problem [SM90] which is defined as follows: with given demand for bandwidth between pairs of network end-points, the question is to compute routes for the given demand such that a maximum quantity of the demand can be accommodated by the network.

Formally, this means for a given a network $G(V, E)$ with V being the set of vertexes and E the set of edges, link capacities $c : E \rightarrow \mathbb{R}^+$, and a set of *commodities* $(s_i, d_i) \in V \times V$ defined by the source and destination nodes for commodity i, a demand $d(i) > 0$ is defined for each commodity i. The MCF problem is to find the largest λ such that at least a λ portion of each demand $d(i)$ can be routed on the network. The problem can be defined as a solution for the following linear program: when P_i denotes the set of paths between s_i and t_i, $P = \cup_i P_i$, and $x(p)$ denotes the amount of flow routed on path p for each $p \in P$, the LP formulation of the problem is:

$$\begin{array}{lcll} \text{maximize} & \lambda & & \\ \sum_{p:e\in p} x(p) & \leq & c(e) & \forall e \in E \\ \sum_{p\in P_i} x(p) & \geq & \lambda d(i) & \forall i \\ x(p) & \geq & 0 & \forall p \end{array}$$

While the integral version of the MCF problem is NP-complete [SM90], it is possible to compute an ϵ-approximation for the fractional maximum multi-commodity flow problem in $O(\epsilon^{-2}|E|^2 \log^{O(1)} |E|)$ time with ϵ being the error factor and E being the set of edges (links) in the network, as was shown by [Kar02].

The basic idea of solving the MCF problem requires to specify the commodities and to find a way to implement the solution of the flow problem in the network management system. In our case, we model commodities as the pairs of end nodes $(s_i, t_i) \in V \times V$ with demand $d_i > 0$, where d_i denotes the total amount of bandwidth allocated by flows between s_i and t_i. The values of d_i are obtained using the status information available to the network management system. Given these input parameters, the optimization process implements the MCF algorithm described in [Kar02]. The solution to the MCF problem provides not only the optimal value of λ but also an assignment of bandwidth to routes such that the optimal value of λ is achieved. If the computed assignment differs from the current assignment on the network, routes are updated accordingly. In case this affects only inactive flows only the internal database is updated, otherwise also the routes currently set up on the network are changed. However, this can be adapted to only affect flows that are not active, thus avoiding any reconfiguration of the network.

Using the MCF based optimization guarantees that at any point in time, the currently best mapping of flows to paths is available at the time the optimization is performed. When changes in the traffic pattern occur, the optimization process reacts rapidly and thus keeps the network as balanced as possible with the available information. Since traffic changes cannot be anticipated by the off-line optimization (see also section 2.2.4), this strategy may also lead to suboptimal results with respect to the global perspective.

The actual computing time for solving a single MCF problem and performing the route updates is rather high, in particular the simulations for this thesis required an actual time of approximately 0.7 seconds for a single run of the off-line optimization involving an average of $1,000$ flows. This value is of course implementation dependent and cannot be seen as representative for other implementations. However, this holds only for a single MCF computation. In the advance reservation environment, this computation must be made for the entire book-ahead interval. Using the slot based model, this yields at most a factor of the length of the book-ahead interval. Hence, a single optimization of the entire book-ahead interval takes a large amount of time and therefore the impact of the off-line optimization may be limited, depending on the actual parameters of the environment, e.g., the slot granularity.

Optimization Granularity

The granularity of the optimization process must be determined regarding the intervals of consecutive slots for which an optimization is performed, similar to the flow switching approach. The previously described optimization technique

can only be applied to time intervals with the same network load. However, this means the demand associated with any pair of network end points must not change during this particular interval. Since the intervals with the same demand are rather short as implied by the utilization profile of individual links (see figure 4.9), this property is relatively difficult to achieve. Thus, it may be possible to compute intervals with only similar demand, and compute the solution of the MCF problem using the average demand during an interval. It is also conceivable to compute the optimization only for individual slots which would require much higher computational effort.

For each of those intervals, a solution for the MCF problem is computed. If necessary, the routes stored in the network management database are updated according to this solution. The slot-based allocation allows to perform the optimization for each individual slot which is of course the finest granularity for the flows.

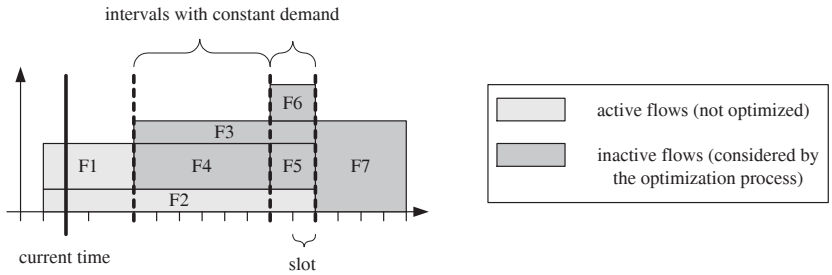


Figure 5.3: Optimization scheme: an MCF computation is performed for each interval with constant demand. Flows can take different routes during different intervals.

In figure 5.3, the general mechanism is depicted showing an example with flows that are included in the computation within a certain interval. The diagram only shows the situation on a single link, as outlined before the demand for any pair of end points must not change during the optimization interval.

The active flows do not contribute to the demand, thus their bandwidth allocations must be subtracted from the available bandwidth, and only inactive flows are rerouted. A different routing can be computed for each time interval, which means flows take more than a single route. This requires the network to change routes over time. In this particular case, flow F3 may be assigned two different paths during the two time intervals marked. The second interval in this example must not be longer even if F7 had the same bandwidth allocation as the cumulative bandwidth allocated by F3, F5, and F6.

Alike the flow switching approach described in section 5.1.4, the advance reservation environment allows to send flows along different routes during their lifetime and to compute these routes before the flows have actually started. This is unique in the advance reservation environment and comprises a considerable advantage in contrast to immediate reservation environments where this opportunity cannot be used for the optimization. This is the reason why the rerouting approach in [BKL03] was only feasible for periods with the same demand matrix which cannot be anticipated in the immediate reservation or best-effort environment. In contrast, the advance reservation environments provides much better

prerequisites to perform an off-line optimization. Similar to the flow switching approach, the off-line optimization also enables the bandwidth broker to restrict the number of different paths that a flow can take.

5.3.2 MCF Implementation

The optimization is implemented as a background process which runs permanently, scanning the whole book-ahead interval and performing optimizations if necessary. As described before, the optimization is performed for time intervals with the same demand such that flows do not start or end in the middle of the interval. This allows to compute the MCF in parallel for different intervals since the individual computations are independent of each other. In order to avoid interference with the on-line admission control process, the locking mechanisms described in section 3.4.6 can be used.

Each MCF computation yields an ϵ-optimal solution for the traffic flows within the respective time interval. The actual parameter λ is not of particular interest in this context. Instead, the assignment of demand to routes is more important. This is implicitly given by the solution for the MCF problem, i.e., for each pair of network endpoints $(s_i, t_i) \in V \times V$ with demand $d_i > 0$ the MCF algorithm provides a set of paths P_i^+ with overload and a set of paths P_i^- with underload, i.e., for each $p \in P^+$ the load must be reduced until the level computed by the MCF algorithm is reached and accordingly for each $p \in P^-$ the load must be increased.

After the MCF computation has finished, route changes are immediately updated in the network management database according to the following algorithm.

```
1  foreach (s_i, t_i) ∈ V × V with d_i > 0 do
2     foreach p ∈ P_i^+ do
3        order flows on p in descending bandwidth order
4        while x_current(p) > x_target(p) do
5           remove flow f with max. c(f)
                such that c(f) < x_current(p) − x_target(p)
6     foreach p ∈ P_i^− do
7        order unassigned flows in ascending bandwidth order
8        while x_current(p) < x_target(p) do
9           add unassigned flow f with max. c(f)
                such that c(f) < x_target(p) − x_current(p)
```

where $x_{\text{current}}(p)$ denotes the amount of bandwidth currently carried on path p and $x_{\text{target}}(p)$ denotes the target load on p computed by the MCF algorithm. $c(f)$ denotes the bandwidth carried by flow f.

It is important to note, that the optimization computes an optimal solution for the MCF problem, but in order to exactly meet the computed amount of traffic sent along each route, it may be necessary to route some flows along more than a single path in parallel. Computing an optimal solution without allowing flows to be transmitted on multiple paths in parallel can be reduced to the integral multi-commodity flow problem and hence is NP-hard. Thus, the above

algorithm computes an approximation using the greedy heuristic which removes flows in descending bandwidth order from the overloaded paths, starting with the flow with the maximal bandwidth. Flows are assigned to the underloaded paths in the same way.

Alike the solution for immediate and best-effort environments outlined in [BKL03], in order to reduce the amount of flow rerouting, it is possible to define a threshold parameter δ such that the actual reassignment of flows to paths is only started when the benefit obtained by rerouting is at least δ, i.e., the condition $\lambda \geq \delta$ holds. In this case, it is necessary to solve the MCF problem twice: once with the currently available unused bandwidth and once with the whole network capacity as unused. When the parameter λ shows a sufficiently large difference when comparing both cases, it is beneficial to reroute the flows according to the second solution.

5.3.3 Issues of the Implementation

A number of different parameters and implementation details exist that can influence the quality of the optimization and the complexity of the optimization process.

The main idea of the optimization is to include only flows that are not yet active. As a consequence, updates are only required in the internal database of the network management and an actual reconfiguration of the network is avoided. However, the framework also enables including active flows in the optimization since their duration is known to the network management system. In such case, the MCF computation only includes those active flows with remaining lifetime above a certain threshold t. This threshold is introduced to ensure that only flows with a sufficiently high remaining transmission time are possibly rerouted. This threshold can be an estimate based on the amount of flows that have to be optimized and the size of the network. t is a parameter that can be dynamically chosen by the network operator.

Restricting the number of different routes of a flow as discussed before may reduce the complexity of the MCF computation but on the other hand lead to a reduced efficiency of the optimization process. Flows that have already taken n different routes will no longer be included in the computation and hence the total computing time is reduced. In this case, restricting the amount of alternative paths each flow may is introduced in order to restrict the processing time of the off-line optimization.

The processing time can be further reduced using the following observation. When the result λ of the MCF computation for a particular time interval yields less or equal $1 + \epsilon$ for the second consecutive time, this means optimizing the respective time interval does not result in any further benefit. Hence, such a time interval no longer needs to be processed in subsequent optimization steps.

When the off-line optimization is implemented as a background process, this process only needs to be run when significant changes of the traffic situation during certain periods within the book-ahead interval occur. Thus, the process can be paused when incoming requests do not imply changes of the demand or actually no requests are issued at all.

5.4 Evaluation

In this section, the mechanisms presented in this chapter were examined using the same setting as described in section 4.3.1, i.e., using the cost239 topology. Several load situations were examined, which were modeled by adjusting the request inter-arrival times.

5.4.1 Routing

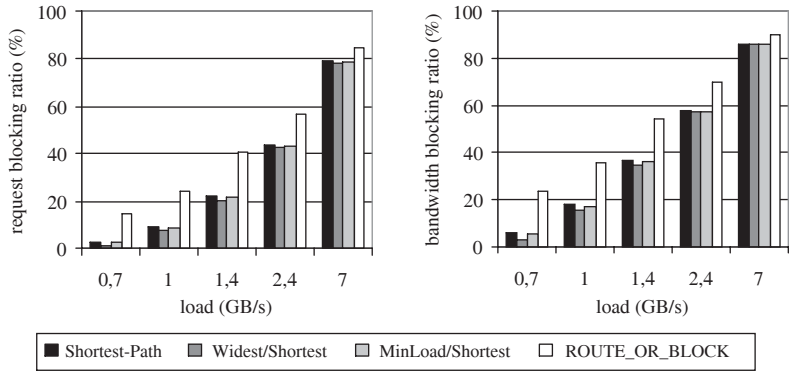


Figure 5.4: RBR and BBR of the different routing algorithms under variable load conditions.

In figure 5.4, the performance of the ASP routing algorithm variants is depicted for different load conditions. Evidently, the ROUTE_OR_BLOCK algorithm cannot compete with the other strategies in the examined environment whereas the other three variants achieve similar performance. This holds for both performance metrics.

The more detailed illustration depicted in figure 5.5 shows that the other algorithms result in similar performance under any of the load conditions, although the **Widest/Shortest** variant achieves better RBR and BBR compared to the simple **Shortest-Path** routing algorithm and the **MinLoad/Shortest** strategy. These results show, that the routing strategy itself is important, and the straightforward extension of known immediate reservation algorithms such as **Widest/Shortest** to advance reservations shows no surprises. Although theoretically optimal in the on-line routing scenario, the ROUTE_OR_BLOCK algorithm cannot compete with the other algorithms.

5.4.2 Flow Switching

The flow switching strategy can increase the network performance. Both types of flow switching approaches described in section 5.1.4 are conceivable and were examined in this context.

Fixed Granularity

The impact of the switching granularity on RBR and BBR is depicted in figure 5.6. The longer the routing intervals are the higher is the BBR. With respect

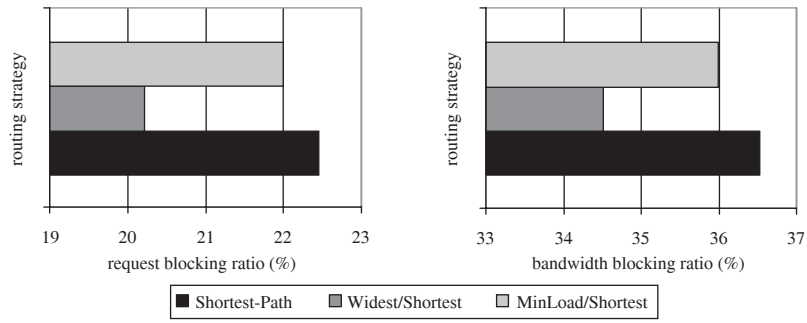


Figure 5.5: RBR and BBR of the on-demand routing algorithms under medium load (1.4 GB/s). **Widest/Shortest** achieves the lowest RBR and BBR.

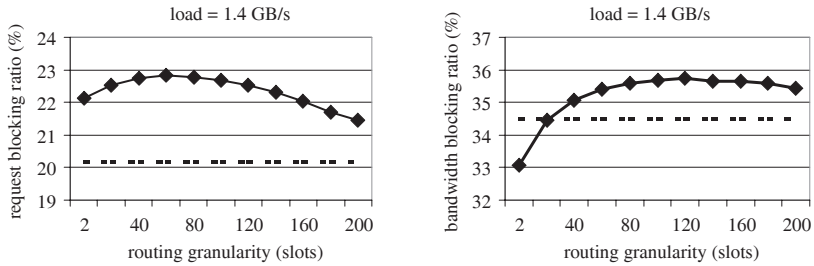


Figure 5.6: RBR and BBR as a function of the routing granularity with medium network load. The dotted line denotes the RBR and BBR using the `Widest/Shortest` algorithm.

to the RBR, the performance does not improve, instead a degradation can be observed. In terms of BBR a slight gain can be achieved when using a granularity of 2 slots. However, in general the fixed granularity approach does not improve the overall performance.

Variable Granularity

Using variable granularity, the RBR and BBR can be improved more significantly. As can be observed in figure 5.7, especially with low load this strategy leads to a performance improvement which even outperforms the approach using fixed granularity. In contrast to the fixed granularity approach which may result in worse performance, the variable granularity approach has the major advantage of achieving considerable performance gains (around 10%) under any load condition.

The overhead of the flow switching approach under the different load conditions is determined by the number of different paths each flow takes. This is depicted in figure 5.8. The diagrams shows, that the average ranges between 1.21 and at most 2.45 different paths. This is a significant advantage compared to the approach using fixed sizes. This approach requires an average of up to 100 different paths when using a granularity of 2 slots with an average duration of 200 slots. The fixed granularity approach with granularity equal to the average number of path switches required for the variable approach is inferior to

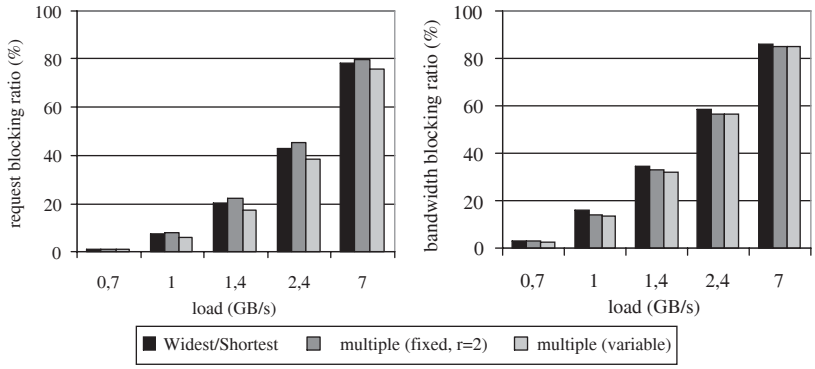


Figure 5.7: Variable granularity of flow switching. In order to show the improvement, the **Widest/Shortest** algorithms for immediate and advance reservations are compared with the best fixed (2 slots) and variable granularity flow switching strategy.

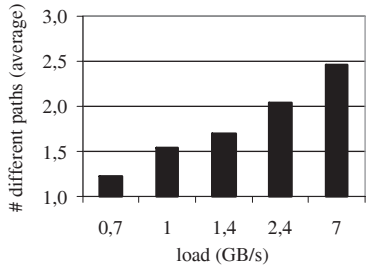


Figure 5.8: The average number of different paths for flows routed on multiple paths using the variable granularity approach.

the variable granularity approach. Concluding, the variable granularity reaches by far superior performance while requiring less frequent route changes. This makes this approach preferable in the given environment.

5.4.3 Malleable Reservations

In order to generate the request sets for malleable reservations, the same parameters as described in section 4.3.1 were chosen and a varying portion of those requests were defined as malleable, while always 100% of the reservations were made in advance. Malleable requests were allowed to differ from the original start, stop times, and bandwidth, i.e., the network management was allowed to change these parameters. In order to exemplify the possible performance of networks with malleable reservations, the following setting was chosen. Some restrictions were applied to malleable reservations: The duration was allowed to differ at most 50% from the originally defined duration. The earliest start time $t_{\min}$ and the latest stop time $t_{\max}$ were at most 50% of the original duration earlier and later, respectively. This means, a request with a given duration of 10 slots was allowed to commence 5 slots earlier than originally specified. These parameters can influence the performance in the sense that less flexibility results in lower and more flexibility results in higher performance.

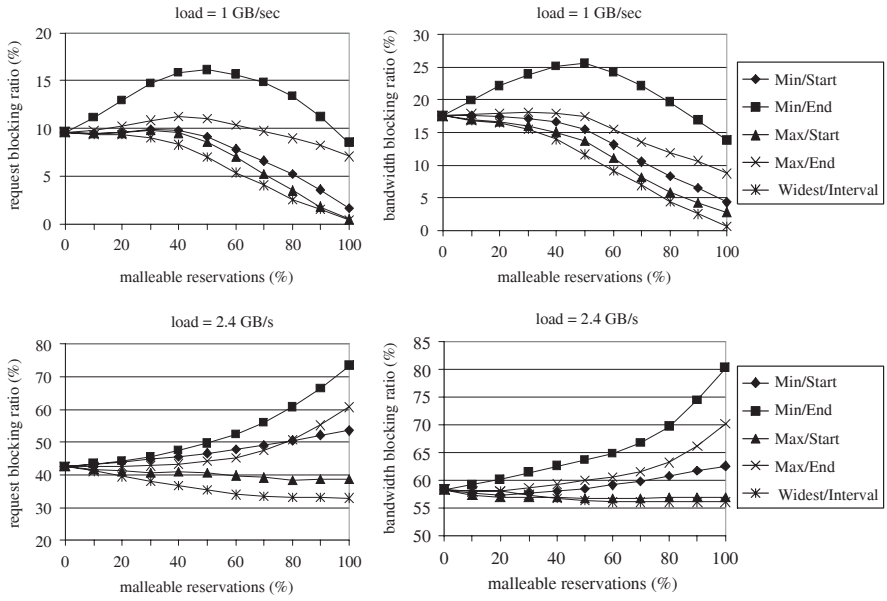


Figure 5.9: RBR and BBR for varying percentage of malleable reservations, using low (1 GB/s, *above*) and high load (2.4 GB/s, *below*).

In figure 5.9, the RBR and BBR are depicted under low and high load conditions, i.e., a load of 1 GB/s and 2.4 GB/s, respectively. It can be observed, that with a relatively low percentage of malleable reservations, the RBR increases for some of the strategies until a point is reached (approximately 40% malleable reservations) where the increment is stopped and the RBR drops significantly to at most below 1%. The BBR is affected in a similar way. Besides this common behavior, the results of the strategies differ largely. While the **Min/End** strategy nearly doubles the RBR with 50% malleable reservations, the **Widest-Interval** strategy achieves an improvement with rising amount of malleable reservations.

The performance in the high load scenario shows a different picture. Only two strategies achieve a performance gain, although this is only moderate. However, the performance is increased with respect to both metrics.

In general, the best performance is achieved using the **Widest-Interval** strategy, although this is achieved at the expense of a much higher computation time. With 100% malleable reservations, the RBR and BBR can be reduced to almost zero percent in the scenario with low network load. This is a significant reduction which outperforms the result of the immediate reservation scenario by far.

The interpretation of the results shown in figure 5.9 requires to recall the results of the analysis in chapter 4. It was shown, that the reason for the performance degradation of advance reservations is fragmentation which impacts the RBR and BBR even with a few percent of the reservations made in advance. Consequently, those strategies which place malleable requests at the end of the search interval increase the fragmentation, while the others - including **Widest-Interval** - achieve a significant performance improvement with suffi-

ciently high amount of malleable requests. In most cases, the **Widest-Interval** approach outperforms the other strategies which shows that a best-fit strategy is worthwhile being implemented although it requires higher response times.

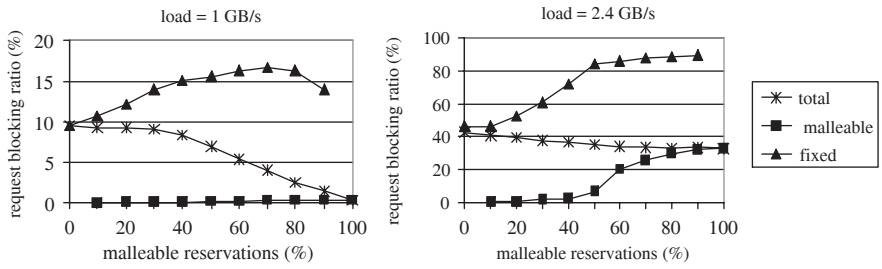


Figure 5.10: The RBR of malleable and fixed reservations using the **Widest-Interval** strategy.

Using a suitable scheduling strategy, malleable reservations are an opportunity to significantly improve the performance of a network. For an individual request, it is also beneficial to be specify as malleable as depicted in figure 5.10. It can be observed, that the RBR of malleable requests remains very low for any possible percentage of malleable requests. In contrast, the RBR of fixed requests increases rapidly, reaching a maximum at approximately 80% malleable requests. With higher percentage, also the RBR of fixed requests decreases.

5.4.4 Off-line Optimization

In order to obtain results for the off-line optimization, the optimization process was run in the background during the whole usual simulation using the **Widest/Shortest** routing algorithm. The granularity for the off-line optimization was 1 slot.

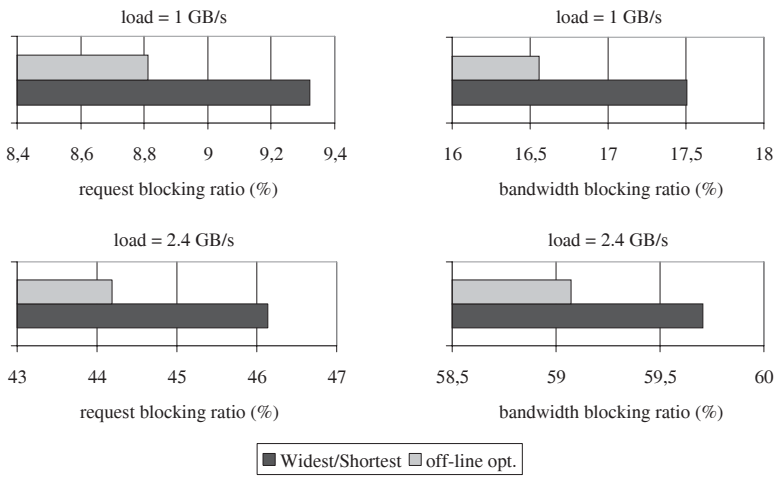


Figure 5.11: The results of the off-line optimization together with **Widest/Shortest** compared to only the **Widest/Shortest** algorithm.

The result of the off-line optimization is depicted in figure 5.11 for two load situations compared to the results of only the **Widest/Shortest** algorithm. It can be clearly observed, that the performance gain is only small with respect to RBR and BBR. In particular, the RBR is reduced whereas the BBR, especially for the higher load scenario, is less affected.

Although the actual computation time depends heavily on the actual implementation and load situation, the simulations made for this thesis showed that the time required for the off-line optimization is considerable. Since locking of certain time intervals enables parallel processing of off-line optimization and on-line admission control, it is possible to handle also incoming requests in parallel. However, the performance of the bandwidth broker is still degraded since the off-line optimization process is operating throughout the whole run-time of the management system. In the bandwidth broker implementation used for the simulations described in this thesis, the off-line optimization required approximately 0.7 seconds for solving the multi-commodity problem for a single slot on a Pentium IV, 1 GHz PC with an average of approximately $1,000$ flows to be handled. With a book-ahead interval of several thousand slots, this means a considerable amount of time must be spent for the off-line computation.

5.4.5 Summary

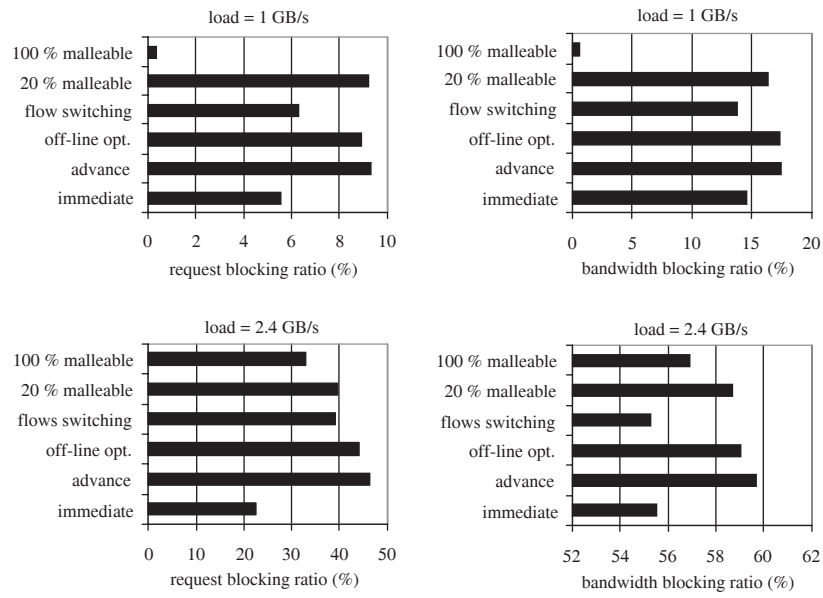


Figure 5.12: Comparison of the different optimization techniques. RBR and BBR are depicted for different load situations.

The performance gain that can be achieved using the different techniques described in the previous sections is depicted in figure 5.12. The diagram shows the performance of the **Widest/Shortest** routing algorithm for immediate and advance reservations compared to the variable granularity flow switching approach and 20% respectively 100% malleable reservations using the **Widest-Interval**

strategy. Furthermore, the result of the off-line optimization is depicted. It can be clearly observed that the performance of immediate reservations can be reached and even improved, in particular with respect to BBR. It is apparent, that especially the flow switching approach achieves a considerable performance advantage. The off-line optimization in general achieves only a low reduction of RBR and BBR. However, the when the costs of this type of optimization can be afforded, in particular the blocking caused by the locks, its application is in particular suitable for the advance reservation environment since stability of the traffic at least during a single slot can be guaranteed and the length of the book-ahead interval allows also for longer processing times of the optimization algorithm than possible in immediate reservation environments.

These results show the considerable performance improvement that can be achieved when using the opportunities of the temporal dimension available in the advance reservation environment. Besides the increased admission probability and additional services that can be implemented for clients in such an environment, the performance that can be achieved by network operators in the environment shows, that it is worth to offer such services. The figures show also, that the actual pricing policy of network operators should be related to the amount of traffic carried by admitted requests rather than to the amount of flows admitted in the network. When the BBR is the dominating performance metric, the performance of advance reservation networks is comparable to immediate environments or even better, depending on the properties of the requests.

Chapter 6

Fault Tolerance

In any QoS environment, the tolerance against the failure of components is a vital part of the system and considerably affects the performance of the network. The term *failure* in this context applies solely to *link failures*. The reason is that the failure of any other component, e.g., the bandwidth broker, does not require strategies specific to advance reservations but can be implemented as in an immediate reservation environment with similar setup. Furthermore, since router failures in essence result in a number of link failures, the same strategies for failure recovery can be applied.

Since failures in computer networks are a common phenomena [ICM+02], it is necessary to develop mechanisms that deal with failures also in the advance reservation environment where resources can be reserved well before they are used. Thus, also in this context the management system needs to use the available knowledge about flows which are not yet active but already admitted and might get affected by a failure.

In this chapter, strategies are described which tackle this problem. These strategies were integrated into the failure recovery module of the bandwidth broker (see section 3.4.5). As for the overall performance (see chapter 4), the focus of the developments shown in the following is on the question how to use the additional knowledge about the future to improve the systems' performance. In particular, two issues are of interest. The first is the impact of failures on the overall network performance, and the second is the impact on individual flows. It can be shown, that the general method to find alternative routes also for flows that are not yet started but are likely to be affected by a link failure reduces the termination probability for flows when reserved sufficiently early. Alike to reserving in advance, this approach is called *rerouting in advance*. In comparison to immediate reservation environments, the proposed techniques lead to superior fault tolerance properties of the underlying network since a larger amount of the affected flows can be rerouted.

6.1 Failure Recovery Mechanisms

6.1.1 Overview

Fault tolerance mechanisms for immediate reservations have been widely studied, in particular in conjunction with MPLS. The general framework for MPLS

based failure recovery was outlined in [SH03] and distinguishes two basic approaches: *protection switching* and *rerouting*. The former requires establishing *recovery paths* before a failure occurs and in case of a failure only requires switching the affected traffic to such a recovery path. In case the affected flows were given a QoS guarantee it is required to assure sufficient capacity on the recovery paths is available. Otherwise, the flows will experience a performance degradation or even termination. Alternatively, it is possible to preempt other than QoS flows on the recovery paths. In contrast, the rerouting approach does not establish any recovery paths in advance and hence, does not require resource reservations for higher quality traffic before a failure occurs. The option to establish recovery paths is an important feature of MPLS since it allows one to switch LSPs as soon as the failure becomes known at the respective LSRs. Since this process does not require convergence of routing protocols (in order to deal with the failure and the resulting topology), switching can be made within extremely short periods of time.

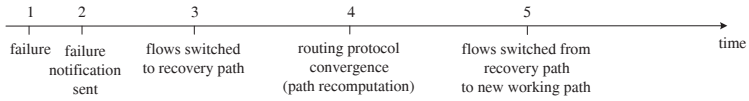


Figure 6.1: Event sequence of the recovery mechanisms used by MPLS [SH03].

Both protection switching and rerouting may be used in combination, i.e., traffic is firstly switched to the recovery paths while new working paths are determined by the routing protocols or a central instance like a bandwidth broker. Once the new working paths are computed, traffic is switched from the recovery paths to the new working paths. The temporal sequence of the whole failure recovery mechanism is outlined in figure 6.1.

In [AK02], an approach to provide different *resilience classes* was introduced. The considerations are focused on providing sufficient bandwidth for all flows in better than best-effort resilience classes using a protection switching approach. The protection alternatives *path protection* and *link protection* were examined. In addition, the impact of link protection respectively path protection mechanisms in the given scenario is examined. The result is the path protection approaches provide a higher resource efficiency, i.e., less bandwidth must be provided for the resilience classes. The reason is that path protection aims to find a global solution for the rerouting problem, thus allowing the management system to use alternative paths with equal or similar hop count. In contrast, link protection reroutes flows locally around the fault and hence, the local alternative increases the overall path length significantly.

In order to determine suitable recovery paths, in [LG01] a path computation algorithm is presented that provides a set of *maximally disjoint* paths for each pair of network end points. This properties assures that a number of alternative paths is available for the initial admission control algorithm which aims to provide load balancing functionality such that a single link failure does not affect too many flows. Furthermore, the path set is used as recovery paths in case of a failure. Flows are switched to the first available of the alternative paths if sufficient capacity is available, and are preempted otherwise.

The approaches described previously deal with routing issues in case of link failures, i.e., how to find an alternative path and which alternative path to

choose. In case, insufficient bandwidth is available at any alternative path, other mechanisms were developed. For example, it is possible to reduce the rate of selected flows in order to allow others to survive a link failure. Besides a rate reduction it is also possible to use *preemption*. Its aim is to preempt one or more low priority flows in order to allow other flows to survive a link failure. Using the preemptive approach, a priority is assigned to each flow and flows of higher priority may preempt those of lower priority. For example, in [dSAU02], an NP-complete optimization problem was formulated which describes the problem of minimal rerouting LSPs in case of link failures in an MPLS-aware network. The optimization problem then is to reduce the amount of LSPs that are rerouted and - based on a priority assignment for LSPs according to the DiffServ model - allows one to reduce the capacity of LSPs in order to reroute higher priority LSPs. Both the preemption and rate reduction of flows (LSPs) are opportunities to limit the impact of link failures on the network performance.

Previous work in the field of failure recovery in advance reservation scenarios dealt mainly with architectures that provide fault tolerance. The assumptions made in [SBK98] are based on the network mechanisms to find alternative routes of RSVP signaling messages. However, as stated in [Rei95a] this is a problem since it leads to an unpredictable behavior and does not provide a sufficient amount of fault tolerance for advance reservations.

In [BN02], a distributed approach for managing advance reservations has been proposed. The focus on the survivability of the management infrastructure. The actual considerations deal with measures that handle with link failures when they occur.

6.1.2 Properties of Advance Reservations

In an advance reservation environment, in accordance with [WS97] three periods of time where failures occur can be distinguished:

1. during the reservation and negotiation phase
2. during the intermediate phase, i.e., when resources are allocated but not yet used
3. during the usage phase

As outlined in [WS97], dealing with failures during the first and the third phase of a flow does not significantly differ from the case in immediate environments. For example, in case of MPLS networks failure recovery mechanisms as described in [LG01] may be used when the failure occurs during the usage phase. Failures occurring during the reservation and negotiation phase, e.g., failures of the bandwidth broker, are not considered here.

Thus, due to the nature of advance reservations, link failures in such environments do not only affect currently active flows but also those that have already been admitted but are not yet active (see figure 6.2). These flows are affected during their intermediate phase. Since a QoS guarantee has already been given, interruptions of these flows must be avoided and hence, advance reservations require a different approach for dealing with link failures.

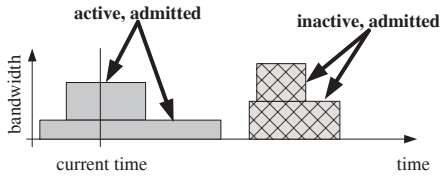


Figure 6.2: Flows in advance reservation environments

6.2 Failure Recovery

As described before, the approach discussed here is the rerouting as denoted in [SH03]. Evidently, it is possible to apply a combination with the recovery switching techniques provided by the MPLS network infrastructure.

The fault tolerance concept for advance reservations outlined in the following consists of two mechanisms: *pre-failure* and *post-failure* schemes. The pre-failure strategies are the initial routing strategies which may be designed in a way to perform load-balancing [LG01], i.e., trying to avoid that a single link failure affects too many flows in situations with low network utilization. Furthermore, when a set of precomputed paths, determined by the pre-failure strategy, is available when a link fails, this path set can be checked in order to determine backup paths. Since this does not require to compute new paths, the whole rerouting process can be accelerated. This is especially important because advance reservations require to check the whole duration of each flow in order to perform admission control. In contrast, the post-failure strategies which are applied after a failure occurred, determine the affected flows and how these flows are treated during the subsequent recovery steps, e.g., the order in which flows are rerouted.

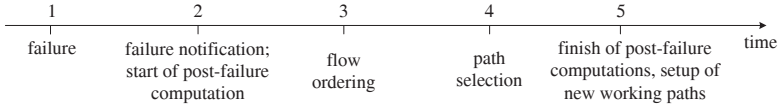


Figure 6.3: Temporal sequence of events in case of a link failure in the advance reservation scenario.

The failure recovery procedure works as indicated by the temporal sequence outlined in figure 6.3. In case a link failure occurs and is notified to the bandwidth broker, the flows that have to be rerouted are determined. The affected flows are then ordered according to the post-failure strategies which will be presented in section 6.3.3. Using the path set associated with each pair of nodes, for each flow a backup path with sufficient bandwidth from the set of precomputed paths is determined. If such a path is found, the flow is rerouted. If not, an individual backup path is computed if one exists, otherwise the flow must be terminated.

The approach described in the following sections aims to use the information about the future network status and reservations as available in the advance reservation scenario in order to improve the probability for a given flow to survive a link failure. Its key feature is that in case of a failure those flows that not yet active but likely to suffer from a service degradation or interruption due to the

failure are *rerouted in advance*. The failure recovery mechanism is implemented within the bandwidth broker as described in section 3.4.

The emphasis in this thesis is on the general approach to provide fault tolerance mechanisms in advance reservation environments. A variety of different opportunities exist to further increase the performance of the network, such as rate reduction or preemption of low priority flows [dSAU02], or transmissions across multiple paths in parallel [LG01]. Support for these methods can be easily added to the bandwidth broker framework, however is not considered here.

According to [SH03], the scheme described before can be classified as a *rerouting* technique, i.e., no recovery paths are allocated before a failure occurs. Instead, each alternative path is determined on-demand at the time the failure is notified to the bandwidth broker. This can be combined with protection switching, i.e., in the MPLS scenario the respective mechanisms for switching the active flows to alternative recovery paths may be used to temporarily reroute the affected flows [AK02, LG01]. In the advance reservation scenario, these recovery paths would be computed and maintained by the bandwidth broker and, since timing is important in the advance reservation context, these recovery paths may change over time.

6.2.1 Rerouting in Advance

The cancellation of flows already admitted a long time ago (see figure 6.2) should be avoided which leads to the question which flows are actually affected by a link failure. For that purpose, the notion of *expected downtime* is introduced which defines the assumed downtime period of a given link. The expected downtime must be computed for each link failure, for example by using statistical data about the duration of former failures, as measured by [ICM+02]. Any flow which is active within this period is taken into account for rerouting which is an important requirement of the failure recovery mechanisms in advance reservation networks. The opportunity to reroute also flows not yet affected by a link failure is very important for the whole setting and distinguishes this approach from previous work in immediate and advance reservation scenarios. As will be shown later, this reduces the termination probability and bandwidth loss compared to scenarios which only allow immediate reservation. The usage of information about future network status, i.e., admitted reservations, in this context can be effectively used to significantly improve the performance of the network for clients as well as for network operators.

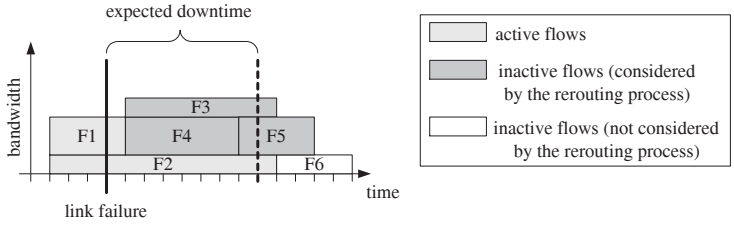


Figure 6.4: Expected downtime and affected flows

An example for the general procedure is given in figure 6.4. The flows F1,

F2, F3, F4, and F5 are affected by the link failure and are rerouted if possible. If F5 cannot be rerouted it will not be canceled but in case the link failure is still present at the start time of F5, the bandwidth broker will then again try to find an alternative path. F6 does not start within the expected downtime and therefore the network management system does not try to find another feasible route for F6.

The other important difference between advance and immediate reservation scenarios is, that once an alternative path with sufficient bandwidth for a given flow is found in an immediate reservation environment, this path can be used until the flow is finished. This means, no further computational effort is required. In an advance reservation scenario, such an approach is not feasible since future requests might block the available bandwidth. Hence, for any alternative path the whole transmission period of a given interrupted flow has to be checked for sufficient bandwidth.

Requests that arrive during the downtime of a particular link and require bandwidth after the expected downtime period for that link can be accepted as usual. Such requests are not influenced by the link failure unless the downtime estimation is incorrect.

The procedures previously described rely on the important assumption that the expected downtime can be accurately determined. The effects of incorrect downtime calculations are described in section 6.3.5. In that section, the effect of not using the expected downtime, i.e., of not rerouting inactive but admitted flows, is also shown. In general, when the bandwidth broker receives up-to-date information about an extended duration of a link failure compared to the original estimation, the post-failure rerouting process is initiated again using the new information. This procedure can be repeated as often as required. Similarly, when a link failure lasts shorter than expected, the bandwidth broker will simply update the status information in its internal database. In case, network rebalancing is required this will be detected and handled by the off-line optimization module. An extended link failure will not require any active flows to be switched to recovery paths since none of those flows are actually routed using the failed link. Instead, the bandwidth broker only needs to reroute inactive flows to alternative paths.

6.2.2 Pre-Failure Strategies

Fault tolerance in an environment as considered in this document is closely related to routing. The actual routing policy as enforced by the bandwidth broker determines how flows are mapped onto the network, i.e., which paths are used, and therefore influences the load that can be put onto the network. For the advance reservation environment, the different aspects of routing were described in chapter 5.

In the context of fault tolerance, routing strategies are seen as pre-failure mechanisms with a twofold purpose. In addition to the on-line admission control process, the routing mechanisms are used to determine paths that can be used in case a failure occurs. This path selection process must be performed as fast as possible, i.e., in the best case without having to individually try to determine feasible alternative paths for any flow to be rerouted.

For that purpose, several approaches can be conceived, three of them are described in the following:

1. k shortest paths
2. maximally disjoint paths
3. on-demand path computation (variant: with path caching)

In [BDF03], two pre-failure strategies were compared that are based on a set of k precomputed paths (see also section 5.1.3). The first simply computes the k shortest paths. The currently fastest known algorithm to compute such a path set is described in [Epp98] (EPP).

A different approach was followed in [LG01], where an approach for providing fault tolerance in immediate reservation environments was proposed. The idea was to compute *maximally disjoint* paths (MAXDIS) in order reduce the probability that too many flows are affected by a single failure, thus achieving also load balancing when applying this strategy also as the main routing algorithm. However, as pointed out in [BDF03] the major drawback of the MAXDIS approach is that the compute path set is relatively small and hence unsuited for both admission control and failure recovery purposes. However, the MAXDIS approach may be used to determine recovery paths for switching active flows if this option is required.

The third approach considered here is based on the on-demand path computation with caching of the computed paths for future usage as outlined in section 5.1.1. In case of a link failure, the paths from the cache are checked for sufficient bandwidth for the affected flows.

When no suitable alternative path can be found after a failure occurred, for any of the pre-failure strategies the on-demand path computation is used as backup solution to try and compute a feasible rerouting path. Thus, each strategy will determine any possible alternative path as long as sufficient bandwidth is available to reroute flows.

Because of the drawbacks of the two precomputing approaches, the on-demand path computation together with the path caching strategy was used as pre-failure strategy.

6.2.3 Post-Failure Strategies

After a link failure is detected and notified to the bandwidth broker, the post-failure strategy determines the reaction to this failure. The following steps are performed in this case:

1. determine the set of affected flows
2. select alternative, feasible path for each flow (using the path set as given by the pre-failure strategy)
3. determine the duration the alternative route is used for

After determining the set of flows affected by a link failure, which includes those flows that are likely affected by the failure in the future, the task of the post-failure strategies is to determine which of the affected flows are rerouted. Formally, when having a number of i affected flows with bandwidth requirements b_i and a total capacity c on the alternative paths, the question is which flows to reroute such that an optimal solution is achieved, i.e., that the amount of bytes

transmitted by flows that are rerouted using the alternative paths is maximized. Another conceivable metric is the number of flows that are successfully rerouted.

Lemma 6.1 (Flow rerouting complexity) *Let $G(V, E)$ be a network, let (u, v) be a link that fails at time t_e, and let $F = (f_1, ..., f_n)$. Furthermore, let $f_i := (s_i, d_i, t_{i,start}, t_{i,stop}, b_i)$ $\forall i \in \{1, ..., \}$, be the set of flows routed using the link $(u, v) \in E$ such that $t_{i,start} \leq t_e \leq t_{i,stop}$ $\forall i \in \{1, ..., n\}$. The problem of finding a subset $\hat{F} \subset F$ to be rerouted on alternative paths such that $b(\hat{F}) := \sum_{j=0}^{|\hat{F}|} b_i * (t_{i,stop} - \max(t_e, t_{i,start}))$, i.e., the amount of bandwidth carried by rerouted flows is maximized, is NP-complete.*

In case, the individual flows f_i are not allowed to be routed in fractions, this problem is an instance of the integral multi-commodity flow problem and hence NP-complete [SM90]. It is possible to obtain a simpler problem by allowing each flow to be routed in fractions using more than a single path in parallel. In this case, the resulting multi-commodity flow problem is solvable in polynomial time. However, the parallel transmission of a single flow using multiple paths was not considered here and in the following sections the focus is on the integer problem.

The complexity of the process that reroutes flows is critical in this environment since the goal is to avoid packet loss which requires a sufficiently fast rerouting process at least for active flows. Computations that last longer can be afforded when dealing with inactive flows.

In the following, a number of different heuristics to solve the flow rerouting problem are proposed which take the properties of the advance reservation environment, i.e., knowledge about the future, into account. The strategy is to firstly order the flows and then try to find an alternative path for the flows in the computed order. The additional amount of information available can all be used to order the flows which leads to a large number of possible strategies. The following seven strategies were chosen, representing a subset of the theoretically possible number of alternative prioritization opportunities.

First-Come First-Served (FCFS) Similar to the basic advance reservation principle, this straightforward strategy prefers flows that have been requested early. FCFS provides a high degree of fairness as perceived by users in the sense that flows admitted at a very early stage are rerouted with a high probability, independent of other properties of the corresponding flow. This is also the basic idea of advance reservations, i.e., early requests for QoS have a high probability that such requests will be admitted. This FCFS strategy however results in a rather unpredictable performance as can be observed in section 6.3. In our experiments, FCFS nearly always ranged between the best and the worst.

Largest Remaining Transmission First (LRTF) LRTF selects transmissions with a high amount of remaining bytes to be rerouted with higher priority.

Smallest Remaining Transmission First (SRTF) SRTF has the opposite effect of LRTF, i.e., transmissions with few amount of bytes left to transfer are preferred.

Widest First (WF) This strategy prioritizes transmissions with the highest transmission rate.

Smallest First (SF) The counterpart of the WF strategy is *Smallest First* (SF), preferring flows with low transmission rate.

Largest Request First (LRF) This strategy uses the totally allocated bandwidth as a metric to order flows, i.e., LRF prefers flows transmitting a large amount of bytes.

Smallest Request First (SRF) Alike WF, LRF also has a counterpart which prefers requests with a low amount of bytes.

In general, any possible metric can be used to prioritize flows during the rerouting process, such as a profit associated with each flow or charge to be paid when a flow is terminated. However, the techniques presented before do not require "external" knowledge about prizing or other prioritization strategy but solely use the available information about flows such as bandwidth requirement or duration which are available in the bandwidth broker.

Path Selection

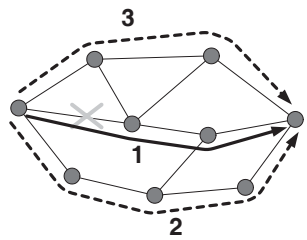


Figure 6.5: Path selection after link failure.

When the previously described ordering of flows is finished, the set of precomputed paths is used in order to select a new path for the affected flows. In case, no suitable path is found among the precomputed paths, the fall-back solution is to compute a feasible path using an on-demand routing strategy described in section 5.1.1. If this computation fails as well, no suitable path is available and the flow in question must be interrupted. In general, the same strategies as described in section 5.1.1 can be used to weight the alternative paths. The actual strategy used during rerouting was **Widest/Shortest**.

The example depicted in figure 6.5 shows a setup with 3 alternative paths. Each flow mapped onto path 1 is chosen for rerouting and either of the two alternative paths can be chosen as long as they provide sufficient bandwidth to carry those flows. In this particular case, path 3 has a lower hop count and therefore will be preferred when applying the rerouting strategy.

Rerouting Duration

As described in section 6.2.1, any flow which uses a broken link during its expected downtime is selected for rerouting. Several alternatives exist for how

long these flows are rerouted. Realistically, only two strategies can be applied:

1. Rerouting each flow for the longest possible period of time, at most for the expected downtime. When the expected downtime is over, each rerouted flow is switched back to its original path. If the link failure lasts shorter than expected, flows can be switched back to the original path if sufficient capacity is available.

2. Rerouting during the whole duration of a flow, independent of expected and real downtime.

While the former has the potential to reroute a higher number of flows since rerouting is only required for a shorter period, the latter approach has the advantage of reduced administrative overhead since a flow needs to be switched at most once. Any of the two approaches can be combined with a flow switching approach as described in section 5.1.4, which means that several different paths can be used in order to increase the amount of successfully rerouted flows. It is clear, that flow switching increases the administrative overhead, as already outlined in section 5.1.4. In this case, the second approach of rerouting for the whole duration of a flow looses its only advantage and cannot be reasonably applied.

When rerouting fails even using the multiple path approach, in case a flow cannot be rerouted during the whole expected downtime it is advantageous to reroute as long as possible and try the rerouting again when this period of time is over. The reason is that flows which had to be terminated release bandwidth that becomes available to other flows, thus avoiding an interruption.

As a result of the previous considerations, the failure recovery module in the bandwidth broker implements the flow switching strategy and reroutes flows only during the downtime of a link.

6.3 Evaluation

6.3.1 Simulation Environment

The environment for the simulations is the same as outlined in chapter 4. The failure model was derived from the examinations in [ICM+02] which studied link failures in IP backbone networks. The study presented in [ICM+02] describes measurements in an IP backbone network made during a 5 month period. Other studies of link failures used a variety of different failure models e.g., with constant failure probability for each link and linearly increasing failure probability over time [BN02]. In [LG01], the examinations are made using a grid topology with uniformly distributed mean time between failures. This does not conform to the observations and measurements real networks and hence is not applied here. Instead, the simulations described in the following use a failure model as described in [ICM+02]. The probability distribution for the failures were model according to the examinations in [ICM+02].

The performance of the pre- and post-failure strategies was assessed using mainly two metrics. Firstly, the *termination ratio* describes the impact of link failures on the flows and is defined as

$$\text{termination ratio} = \frac{|B|}{|A|}.$$

where B denotes the set of terminated flows due to link failures and A denotes the set of flows that was affected by the link failure. The second metric is the *bandwidth loss ratio* which is defined as

$$\text{bandwidth loss ratio} = \frac{\sum_{i \in B} b_i}{\sum_{i \in A} b_i}.$$

with b_i being the amount of bytes carried by flow f_i. Again, in order to reduce the number of figures and for the sake of clarity, only the *cost239* topology (see section 4.3.1) was used to generate the results described in this section.

6.3.2 Performance of the Pre-Failure Strategies

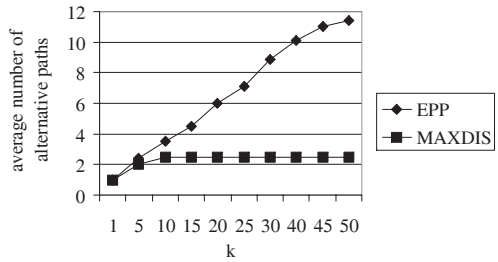


Figure 6.6: Pre-failure strategies: average number of alternative paths between two end nodes depending on the parameter k.

Both k path algorithms, i.e., EPP and MAXDIS, do not compute exactly k paths. In case of EPP, this is due to the fact that this algorithm also returns paths with loops which must be removed, and in case of MAXDIS, this is caused by the way the "maximally disjointedness" is determined by the algorithm. Figure 6.6 shows the average size of the path set compute over any two end nodes of the cost239 topology. This illustrates the general problem of the MAXDIS strategy: this algorithm only computes a very limited number of alternative paths and a "saturation" is reached very soon. For example, with a parameter of $k = 15$, the average path set consists of 4.5 paths for the EPP strategy whereas MAXDIS computes path sets with an average size of only 2.5. Hence, the amount of interrupted flows for which no alternative path can be found is higher than for the EPP strategy. This shows, that MAXDIS in general is less suited to be applied in an environment as considered here although its approach is to compute maximally disjoint paths and therefore to achieve a relatively even distribution of the network load. In contrast, the k shortest strategy using the EPP algorithm always reaches the desired number of alternative paths if they are theoretically feasible and the parameter k is adjusted accordingly. The path caching approach is the third alternative for using precomputed paths. This strategy achieves an average path set size of 14.6.

Since an on-demand path computation is applied as a backup solution for any of the previously described strategies, the possible set of paths is only important

for the time required to compute the backup paths. It is always guaranteed that a path with sufficient bandwidth is found if one exists.

In a situation as discussed here, i.e., with a network utilization near the maximum, the number of links with sufficient amount of free bandwidth for rerouting a flow is limited. Therefore, the MAXDIS strategy with only a few alternative paths cannot exploit the whole set of available paths and leads to an increased termination ratio. Using the backup on-demand path computation will therefore be needed very often which reduces the speed of the algorithm. With large parameter k, this requires a considerable amount of time. Although in the advance reservation scenario a slightly higher time required for rerouting is affordable for most of the affected flows because there can be considerable amount of time between the link failure and the start of those flows, the general approach should be to finish the rerouting process as soon as possible. This is needed particularly for short link outages which make up the largest overall share of failures according to [ICM+02].

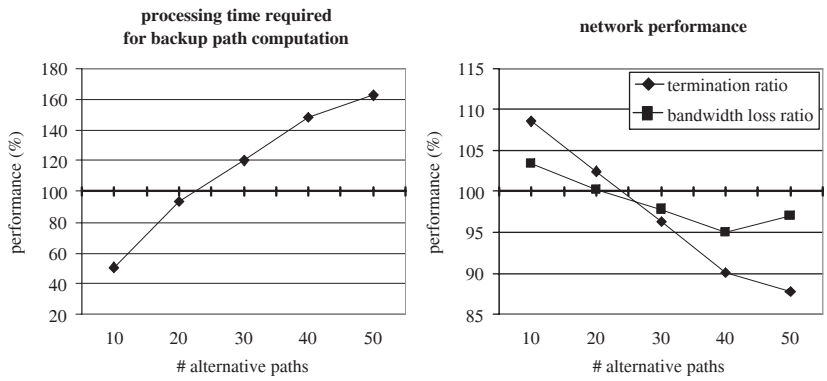


Figure 6.7: Performance of the EPP strategy relative to the on-demand/path caching approach (100%): rerouting speed (*left*), termination ratio, and bandwidth loss ratio (*right*).

In figure 6.7, the average time required for the rerouting process using the k-path approach is depicted relative to the time required using path caching, i.e., the path caching approach exactly results in 100%. It can be observed that up to a path size of approximately $k = 22$, the k-path approach is superior but then the performance decreases drastically. On the other hand, the diagram on the right hand side shows that the parameter k must be relatively large in order to achieve a significantly lower termination ratio and bandwidth loss rate. Adjusting the parameter k requires to make a trade-off between rerouting speed and network performance. Since the k-path strategy is not used to provide recovery paths for recovery switching, but to improve the admission speed for rerouting the affected flows, the path caching approach was used for the subsequent simulations. Its advantage is that no trade-off must be made between rerouting speed and performance.

6.3.3 Performance of the Post-Failure Strategies

As described before, the chosen pre-failure strategy implements the on-demand path computation together with caching of computed paths. The post-failure strategy then determines which flows are rerouted in case of failures.

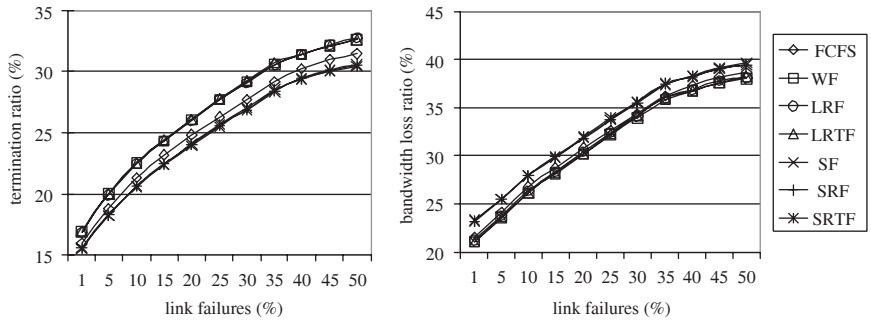


Figure 6.8: Post-failure strategies: termination ratio and bandwidth loss ratio with different amount of simultaneous link failures.

In figure 6.8 the termination ratio and bandwidth loss rate are depicted for variable amount of simultaneous link failures. It can be seen, that differences exist between the different strategies. In general, those strategies which prefer requests with short remaining duration or requests with low remaining amount of bytes to be transferred (SF, SRF, SRTF) perform well with respect to the termination ratio. In contrast, it can be observed that those strategies which prefer flows with long remaining duration respectively high remaining bandwidth requirement (WF, LRF, LRTF) achieve a lower bandwidth loss ratio than the others (see figure 6.9). The FCFS strategy which prefers flows admitted the longest time ago, ranges between the best and the worst strategy.

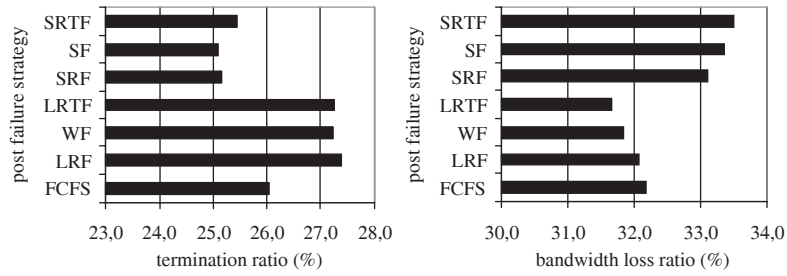


Figure 6.9: Termination ratio and bandwidth loss ratio for the different post-failure strategies with 25% link failures.

While figure 6.8 shows that the performance difference of the strategies persists with increasing amount of link failures, in figure 6.9 the performance of the strategies is examined in more detail. Among the ”S” strategies, SF consistently performs best concerning the termination ratio and SRF is superior with respect to the bandwidth loss ratio, whereas the WF strategy performs best among the ”L” strategies with respect to the termination ratio but cannot compete with

the other concerning the bandwidth loss ratio. LRTF is most successful with respect to the bandwidth loss ratio. This mixed picture shows the difficulty to select a reasonable strategy.

However, since the FCFS strategy always achieves a result between the best and the worst of the other strategies, it is reasonable to use the FCFS strategy in order to implement a policy which prefer those flows admitted very early. This can positively influence the customer satisfaction and meet the clients expectations and therefore can also be of interest for network operators. The usage of FCFS for failure recovery implements an analogue idea as the advance reservation service itself: requests issued at an early stage receive the requested QoS with the highest probability. This important property is examined in more detail in section 6.3.4. Using FCFS as post-failure strategy therefore seems to be a reasonable approach since it provides the highest fairness as perceived by clients, and also can be seen as a trade-off between the other strategies which achieve either good termination ratio or good bandwidth loss ratio. Therefore, unless otherwise stated the subsequent simulations were all made using the FCFS strategy.

6.3.4 Impact on Individual Flows

As mentioned in section 6.3.3, the FCFS strategy provides lower termination probability for flows reserved a long time in advance. This property is fostered further using the pro-active *rerouting in advance* approach as will be shown in this section.

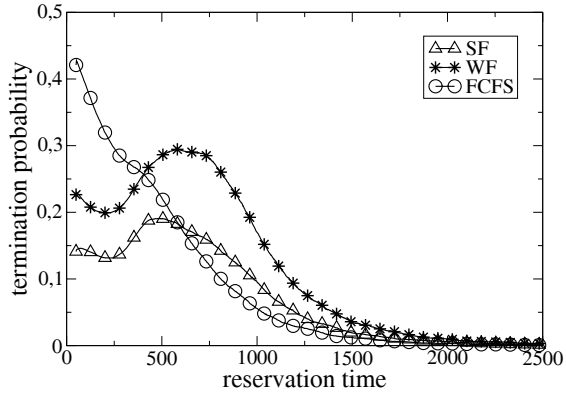


Figure 6.10: Termination probability as a function of the reservation time for different post-failure strategies with a mean reservation time of 500.

The impact of the reservation time on the termination probability is depicted in figure 6.10. The diagram shows the termination probability of advance reservations as a function of the reservation time for different post-failure strategies. It can be observed, that reserving a long time before the usage phase pays off not only in terms of admission probability (see also [WG98]) but also in case of link failures. For each of the simulations, the reservation times of the requests were exponentially distributed with a mean of 500.

The FCFS strategy has the lowest termination ratio for flows with reservation times larger than the mean. For FCFS, the result is a strictly monotonic

decreasing curve, which is a consequence of the ordering of flows which only takes the reservation time into account. In contrast, the WF and SF strategy show a peak at 500 (SF) and approximately 600 (WF), respectively. Both strategies show a local minimum at approximately 250 which is exactly the mean duration of the requests. In general, SF has the lowest average termination probability as was already shown in Sec. 6.3.3. The order according to the transmission rate as made by WF and SF leads to the peak near the mean reservation time from where on the termination probability declines since the total amount of reservation present with reservation times higher than the mean is relatively small.

For other than FCFS strategies, the time from whereon the termination probability decreases can be denoted as a *critical time* (see also [WG98]), which may be conveyed to clients in order to allow them to plan their reservations accordingly.

With the same mean reservation time, the FCFS strategy results in the lowest termination ratio for requests reserved at least the mean reservation time in advance. The reason is two-fold: although all strategies perform rerouting in advance, longer reservation times are preferred for rerouting by FCFS and as a consequence their termination probability is reduced significantly. In contrast, the termination rates of the WF and SF strategies are higher for reservation times around the mean.

6.3.5 Influence of the Downtime Calculation

In the previous simulations, it was assumed that the expected downtime exactly matched the actual downtime. However, it cannot be assumed this is always possible under realistic conditions and hence, it is important to look at the impact of inaccurate downtime calculations on the performance. In particular, it is of interest how inaccurate assumptions, i.e., over- or underestimations of the actual downtime, influence termination ratio and bandwidth loss ratio on the one hand and request blocking ratio and bandwidth blocking ratio on the other hand. In order to determine the impact of inaccurate estimations, the following simulations were always made using the same actual downtime with different estimations, i.e., underestimating a failure with duration 100 by 10% means a downtime of 90 is estimated.

Two cases can be distinguished: the downtime might be calculated too conservatively which means the actual downtime of a link is shorter than expected. In the second case the downtime is underestimated, i.e., the actual downtime is longer than expected. In the simulations, the recomputation of the expected downtime in those cases was made accordingly: the remaining actual downtime was underestimated by the same percentage as for the initial estimation. This process continued until eventually the actual downtime was correctly estimated.

In figure 6.11, RBR and BBR are shown as a function of the downtime deviation, i.e., the difference of the actual downtime compared to the estimate. A negative downtime deviation means an overestimation and positive deviation means an underestimation of the actual downtime. It can be observed that overestimating the actual downtime leads to a reduced amount of flows being accepted. The blocking ratios decrease when the estimated time is reduced. The lower the estimate, the higher the amount of flows that can be accepted. The reason is that rerouting flows on alternative - in this context: less optimal -

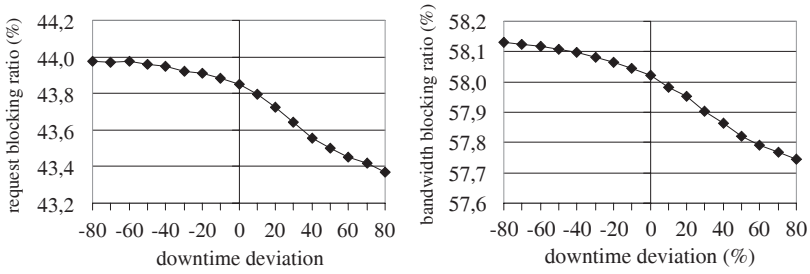


Figure 6.11: Influence of inaccurate downtime estimation on RBR and BBR. Negative values denote an overestimation and positive denote an underestimation of the actual downtime.

paths leads to a higher bandwidth consumption and thus, to a reduced amount of flows that can be accepted. However, the expected downtime computation also affects new requests for bandwidth that fall into the expected downtime: the bandwidth broker does not map those new requests onto a link which is known to be down. This increases the effect of using suboptimal paths and hence, when the actual downtime period is shorter than expected both RBR and BBR rise. However, the figures also show that, using the failure model of [ICM+02], the impact on the overall performance remains low since the failure situations are only exceptions of the normal operation.

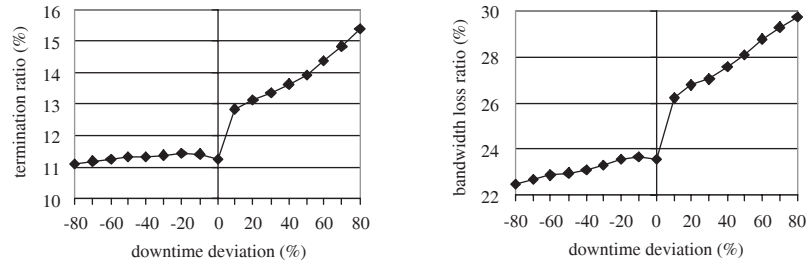


Figure 6.12: Influence of inaccurate downtime estimation on termination ratio and bandwidth loss ratio.

The impact of an inaccurate downtime estimation on termination ratio and bandwidth loss ratio is shown in figure 6.12. It can be observed that overestimating the actual downtime only slightly affects termination rate and bandwidth loss ratio. This is the expected behavior of the system since inactive flows that cannot be rerouted at the time of the failure will not be terminated but checked again at their start time. When the actual downtime is below the expected downtime, flows that cannot be rerouted but start after the end of the downtime are not affected at all by the failure. On the other hand, an underestimation of the actual downtime immediately leads to a significant performance loss compared to the case of an exact estimation. This occurs even with only little underestimation and - compared to the exact computation - results in an

aggravation of up to 34% with respect to the termination ratio and 26% with respect to the bandwidth loss ratio.

When the downtime is overestimated, the network load becomes unbalanced at the time the failed link is active again. In this case, it is possible to apply off-line optimization techniques in order to rebalance the load and thus, avoid the negative impact of the overestimation on the request blocking ratio and bandwidth blocking ratio. However, the actual impact on the network performance is rather low compared to the result when having an exact estimation and additionally, rebalancing cannot completely expel the performance degradation, since incoming requests are not allotted to links known to be down during the requested transmission time.

The conclusion to be drawn from these examinations is that the estimation of the link downtime must be made such that underestimations are avoided, e.g., the actual estimations can be based on the results of the analysis of link failures in a network, as made in [ICM+02]. The results also show, that rerouting in advance, i.e., rerouting inactive flows, is important. When only active flows are rerouted, e.g., using a mechanism from the field of immediate reservations, each inactive, affected flows must be rerouted when they start. Consequently, the result is the same as for the underestimation (positive deviation) shown in figure 6.11, i.e., a significant rise of the termination ratio and bandwidth loss ratio. The importance of rerouting in advance is even higher than the actual choice of the rerouting strategy, since the differences among those are less significant (see section 6.3.3).

With an underlying business model which defines the cost of a terminated flow and the revenue of an accepted flow, it is possible to determine the optimal "overestimation" time by using a trade-off between termination ratio and bandwidth blocking ratio on the one hand and RBR and BBR on the other hand.

Chapter 7

Data Structures for Admission Control

Performance aspects as considered in this thesis do not only cover the field of network performance in terms of request or bandwidth blocking ratio. In addition, another important performance aspect is the time required to manage the network resources. This aspect deals with issues like computational complexity of the tasks to be fulfilled in order to compute routes, determine suitable transmission intervals for large amounts of files, etc. These issues, as far as they concerned the developments described in this thesis, have been covered in the previous sections.

The missing aspect is the question of the performance of the bandwidth broker itself, i.e., the response time of the broker to a given advance or immediate reservation request. In this context, the data structures in which the information about future link utilization is stored, play a key role. Most of the time required to compute routes during admission control is spent when accessing those data structures. This occurs during the common on-line admission control, where in particular the support for malleable reservations demands rapid access to the data structures, and furthermore the off-line mechanisms used to optimize the overall network performance and the failure recovery mechanisms. The different mechanisms presented in the previous chapters require to minimize the computation time for several reasons:

- Short admission times are important in the on-line scenario as they minimize the response time for clients. The response time for an individual client also depends on various other factors, e.g., the message run-time and therefore the processing time of the broker may not be a major concern for individual clients since advance reservations usually also allow for a considerable reservation time. However, the shorter the on-line admission time for a single request, the more requests can be accommodated by the bandwidth broker in general. This is important when the bandwidth broker runs on a device with limited processing capability such as a router. Furthermore, request types such as malleable reservations with their drastically increased complexity may influence the processing time such that it can become the dominating factor.

- Failure recovery mechanisms due to their nature demand for fast response times in order to reduce packet loss and performance degradation of admitted flows. Failure recovery requires fast switching of flows depending, and hence can be observed as admission control task.

- Off-line optimization can result in network performance gains but require also significant computing time. Therefore, the time spent in this process should be minimized.

- Although locking mechanisms enable concurrent processing of on-line and off-line mechanisms, blocking of either process is not desirable. Hence, the shorter the locking periods of either process, the better for the overall processing time.

- The on-line admission control of the network system may only be a single of a large number of different resources, for example in a grid environment. In such a setting, the co-allocation of the requested resources may need more than a single round of requests between the client and the respective resource management systems. In this context, the whole admission process can be delayed considerably when the response times of the individual resource management systems are too long.

The total processing time of the bandwidth broker for a single request consists of various tasks. The most important one in terms of processing time is the check for sufficient bandwidth on the links of the network and the update of the internal data structures. Following that, the routing, i.e., the search for a suitable path using the results from the bandwidth check, is the factor with the next lower amount of processing time in the broker. The remaining tasks each require very short amounts of time. These tasks are related to queuing of incoming requests, the framework around routing and bandwidth check, and the representation of the network in the bandwidth broker. Furthermore, besides updating the data structures, for each accepted flow some more information need to be stored, such as the source and destination node, the path it takes, the start and stop time, etc. In the following, the tasks not directly associated with either routing or the data structures are all referred to as *administrative* tasks. As will be shown later in more detail, the processing time spent in the data structures adds up to up to 60% of the total processing time required by the bandwidth broker, whereas routing requires only about 8% and the administrative tasks the remaining 32% of the total processing time. Therefore, optimizations of the data structures are most likely to improve the processing speed of the bandwidth broker.

As described in chapter 3, the aggregated bandwidth for each link is stored for each time slot. For the purpose of storing the aggregated bandwidth allocation with each link, two data structures are examined with respect to their performance in a bandwidth broker. Besides the basic functionality of efficiently adding and deleting bandwidth, the capability to determine the available bandwidth during longer time intervals is required, for example, in order to perform efficient admission control for malleable reservations. Apart from the analysis and measurement of the processing time, another aspect is the memory efficiency which may also influence the applicability of either data structure.

In order to perform the basic admission control process, two operations must be supported. Before a reservation can be made, it is required to check each link on the path for sufficient bandwidth. Therefore, admission control involves a two-phase procedure:

- `CHECK`: Determine whether a reservation request can be fulfilled.
- `UPDATE`: The data structures for each link involved in a transmission are filled with the new values, i.e., the resource requirement of the newly admitted request is added to the existing entries of the data structure. The `UPDATE` phase is only performed in case the previous `CHECK` phase succeeded.

Before examining the data structures with respect to these two operations, the properties of both arrays and trees are briefly introduced.

7.1 Array

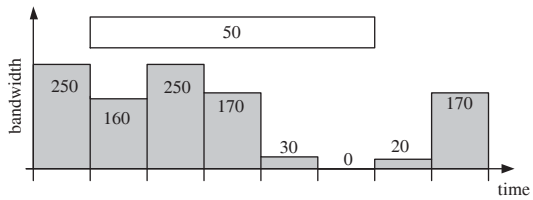


Figure 7.1: Example: Allocation profile and bandwidth values stored in an array. For each affected slot, the bandwidth of the new request (50) is added to the existing quantities.

Using the slotted time model as assumed throughout this document, the application of arrays is straightforward. Each entry of the array represents a single time slot and stores the accumulated resource requirement (in our case: *link bandwidth*) during the corresponding time interval (see figure 7.1). In order to limit the book-ahead interval, the array is implemented as a ring buffer using a pointer to the time slot representing the current time in the bandwidth broker (see figure 7.2). This allows to handle the advancing time in a very elegant and efficient way.

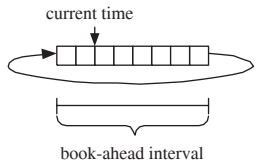


Figure 7.2: Array implemented as ring buffer with pointer to current time slot.

The memory requirement of an array in this form is $b \times s$, with b being the length of the book-ahead interval and s denoting the memory requirement for a single slot.

7.2 Segment Tree

In [SNNP99], a tree structure was developed for supporting the admission control task in advance reservation environments, the so-called *segment tree* which is described in the following.

A segment tree is essentially a complete binary tree with nodes representing the allocated bandwidth during certain time intervals. Each node of the binary segment tree represents one period of the overall book-ahead interval and stores the bandwidth reserved exactly for that period. Both start and stop time of the respective interval of a given node are stored within the node together with two values related to the reserved bandwidth. The first value (denoted by *node value*) is the bandwidth reserved during the particular period of time the node stands for. The second value (*max value*) denotes the maximum sum of node values in any of the subtrees below the current node (see figure 7.3). Starting at the top node, the two child nodes each represent one half of the duration of their father node. In order to perform admission control, each reservation request is processed as follows:

1. Starting at the top node, each reservation whose period is covered by the current node, "falls" through to the next level below the current node, i.e., the admission procedure is done with the node at the level below which covers the particular period of the reservation.

2. If the duration of the request intersects with the duration of more than one node on the current level, the reservation is split at the intersection points and then each of the parts "falls" through, i.e., for each part the admission decision is made independently.

3. If a reservation or a part of it completely fits into the duration covered by a tree node (such a node is called *final node*), the "fall-through" process stops for the respective part or the whole reservation[1].

While the reservation or the parts of it "fall through" the tree, the node values of each tree node visited are added up. This happens also for final nodes. When "fall-through" process is finished, i.e., all final nodes are found and completely cover the duration of a part of the original reservation, it is checked whether the requested bandwidth together with the computed sum does not exceed the link bandwidth. In case the bandwidth is sufficient, the request can be admitted. This has to be checked for each path from the top node to the final nodes. In case sufficient bandwidth is available for any part of the reservation, the node values in the corresponding final nodes can be updated. The traversal process can be stopped in case insufficient bandwidth is detected at any of the visited nodes. Following a successful check, each path has to be traversed backwards in order to update the max values in each node previously visited. Figure 7.3 shows a segment tree before and after a new request (bandwidth = 50, starting slot = 1, finishing slot = 6) is inserted, final nodes are denoted in gray. The bandwidth values stored in the tree before the insertion corresponds to those depicted in figure 7.1.

[1] The final nodes are not necessarily leaf nodes.

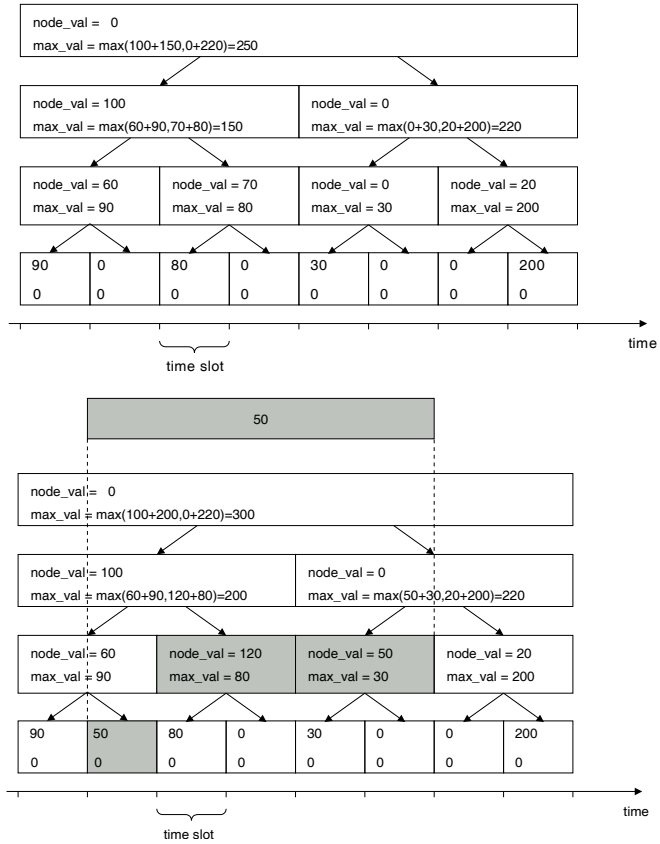


Figure 7.3: A segment tree before (*above*) and after (*below*) insertion of a request.

7.2.1 Details of the Implementation

Two alternatives are conceivable in order to implement the segment tree:

1. pointer based implementation
2. tree embedded in an array

Pointers - although suggested in [SNNP99] - are rather unsuited in this particular case because the segment tree represents a complete binary tree and hence embedding the tree in an array saves the memory space required for the pointers. On the other hand, a pointer based implementation allows one to use dynamic memory allocation of tree nodes during run-time as will be described in section 7.2.2.

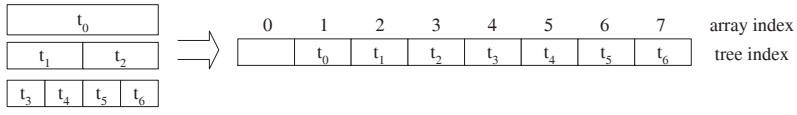


Figure 7.4: Tree embedded in array: root node is placed at array index 1.

Tree traversal as required by the `CHECK` and the `UPDATE` operation require to compute the respective array indices for father and child nodes. This can be supported by embedding the tree in the array starting with the root node of the tree at element with index 1 (instead of index 0, see figure 7.4) this allows to ascend or descend in the tree using only one shift operation (instead of multiplying or dividing by 2) and at most one addition. This means, when n is the array index of the currently visited node, the index of the father node is computed as $n/2 = (n \gg 1)$ (where $\gg$ denotes the *shift right* operation) and the left and right child nodes are computed as $n \cdot 2 = (n \ll 1)$ and $n \cdot 2 + 1 = (n \ll 1) + 1$ respectively.

Furthermore, two other alternatives exist to implement the tree traversal: recursive and nonrecursive descend and ascend operations. However, recursions are costly (although the computational complexity is not affected) in practical implementations due to the computational overhead required for function calls and hence, this alternative is unsuited. Therefore, nonrecursive tree traversal was used for the following examinations. The complete admission control procedure requires at most one descend (`CHECK`) to the final nodes and one ascend (`UPDATE`) from the final nodes in the tree.

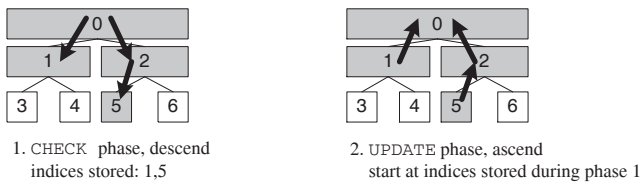


Figure 7.5: `CHECK` and `UPDATE` phase for segment trees, gray boxes denoting visited nodes. The final nodes $(1, 5)$ for each descend/ascend path must be stored during `CHECK`.

Using the nonrecursive tree descend approach requires to keep those nodes where the left and the right subtree must be checked in a separate data structure ("split" nodes). For example, the root node 0 in the situation depicted in figure 7.5 needs to be stored like this. Since the number of those "split" nodes is limited by the total number of nodes visited during the traversal, it is feasible to store these nodes in an array. In the given network scenario, the `CHECK` phase must be performed for any link on a given path. This means, it is required to store the final nodes, respectively their indices in the array the tree is embedded in (see figure 7.5), after each successful `CHECK` phase in order to support efficient retrieval of those nodes during the `UPDATE` phase. For this purpose, final nodes are also stored in an array during the `CHECK` phase.

These considerations lead to the following implementation of the `CHECK` phase (tree descend). Two additional arrays are required, i.e., the array to store final nodes and one to temporarily store "split" nodes:

1. Descend to the next lower level in the tree, checking each node for sufficient bandwidth as described in section 7.2.

2. In case a node is found where left *and* right subtree must be checked, insert this node's index in the array used to store "split" nodes at the rightmost position. Then, scan the left subtree of this node.

3. In case a final node is found during the descend process, i.e., no further descend is required, insert this node's index in the corresponding array storing final nodes. Remove the leftmost index from the array storing "split" nodes (with right subtree still to be checked) and restart descending the right subtree of this node. If this array is empty, the check is over. The index of each final node visited during the whole process is stored in another array. These indices are used to start the update phase from.

4. In case, insufficient bandwidth is determined at any node visited during the descend process, the check phase can be stopped immediately.

In a network management system as considered here, the `CHECK` phase has to be successful not only for a single tree but for the whole number of trees associated with the links on a given path. This means, the `CHECK` phase for the whole admission control process is successful only if sufficient bandwidth is available on all links on a given path.

Following a successful `CHECK` phase, the `UPDATE` process for each link is initiated, starting from the final nodes which were stored in an array during the `CHECK` phase. During this process, the node value of the final nodes and the max value of any other node is updated.

7.2.2 Dynamic Memory Allocation

One of the major drawbacks of the segment tree is the memory consumption. Although the memory consumption of the original implementation [SNNP99] can be reduced by not storing start and stop times with each node (instead they can be computed during tree traversal) which was the approach for the implementations presented here, the memory requirement is far worse than that of arrays.

Starting from the observation that only a limited number of tree nodes is actually required even in case many reservations are stored within the tree, the memory requirement can be reduced by only allocating the nodes that are actually required. This is done dynamically during run-time. At the beginning, only the root node is present. In case a new reservation is added, missing tree nodes are generated on-demand during the tree traversal.

In order to realize this implementation alternative, the pointer-based method must be used.

7.3 Advancing Book-Ahead Interval

A drawback of the segment tree is its unsuitability for a dynamically advancing book-ahead interval. This means, that with advancing time but unchanged duration of the book-ahead interval for each new time slot a new memory cell must be used.

In order to cope with that problem, two trees can be used as depicted in figure 7.6. At the time the book-ahead window completely covers one tree, the content of the other tree can be completely deleted or remain in place for re-usage in order to implement a mechanism similar to the ring buffer. However, in the latter case the node value and may value of each node in the tree must

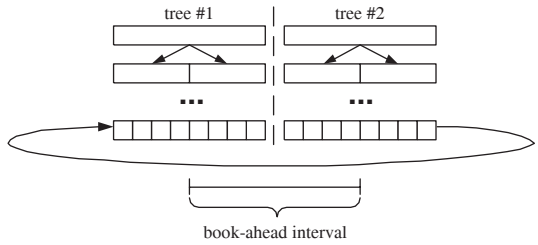


Figure 7.6: Usage of two trees for coping with the dynamically advancing book-ahead interval.

be reset to zero. This process may require more time than the deletion which must be considered.

In any case, the worst-case memory consumption is doubled. In addition to that, in case a single reservation spans more than one tree this does also increase the admission time since reservations must be split and an admission decision has to be made in both trees.

In contrast, arrays can be easily implemented as ring buffers using a single pointer to mark the current time (see figure 7.2). This does neither affect the admission speed nor the memory requirement of the array.

7.4 Performance Analysis

In order to analyze the behavior and performance of both data structures, it is of interest to determine the number of memory cells $a(n)$ accessed during a single round of `CHECK` and `UPDATE`, with n being the duration of the corresponding request[2]. Instead of examining only the worst-case using the O notation, this approach has the advantage to more accurately show the properties of the data structures while revealing also the reasons, which cannot be expected from measurements.

7.4.1 Array

The memory requirement of arrays is fixed and equals the size of the book-ahead interval counted in slots. The number of memory cells accessed during the `CHECK` and the `UPDATE` phase only depends on n and can be easily computed as

$$a_{\text{array}}(n) = 2n. \tag{7.1}$$

7.4.2 Segment Trees

While arrays are easy to analyze, things get more complicated in case of the segment tree. For a given request of duration n and a book-ahead interval b, $a_{\text{tree}}(b, n)$ represents the average number of memory cells accessed during both phases. The average is computed over any possible start time of the request

[2]This corresponds to the steps performed when a request can be admitted.

within the book-ahead interval b. Unlike arrays, in this case a_{tree} also depends on b.

The approach is to determine the average amount of final nodes and their length, i.e., the duration each final node covers, accessed during both phases. With this knowledge, it is possible to determine the average depth of each final node in the tree which is then used to determine to average amount of memory cells $a(n)$.

In the following, the estimations will be made under the assumption that the start time of the reservation is uniformly distributed in the interval $[0, b]$, with b being the book-ahead interval. This means, requests appear at each position in the interval $[0, b]$ with the same probability. The general approach in this analysis is to determine for each such position in the tree the number of final nodes (see section 7.2) and thus the total amount of final nodes f. Then, the average duration l covered by each final node within the tree and the average amount of final nodes $\hat{f}$ for each position is computed. With these results, the distance of those final nodes from the root node in the segment tree can be computed which provides the number of nodes and hence memory cells that need to be accessed during the `CHECK` and the `UPDATE` phases.

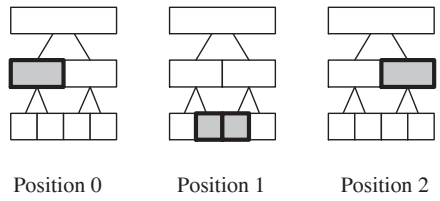


Figure 7.7: Final nodes (gray) and possible positions for a request of duration 2 in a tree with $b = 4$.

The following example illustrates the approach. For the situation given in figure 7.7, with $n = 2$ and $b = 4$ there is a total of 3 different positions of the request of duration 2 in the tree. There are in total 4 final nodes at the three positions. In order to reach a final node during the `CHECK` phase, 2 memory cells (node value and max value) are accessed for each node on the path. Then, each final node is stored in an array and recovered in the `UPDATE` phase which starts at those final nodes. During the `UPDATE` phase, only 1 memory cell is accessed per node, i.e., the node value at the final nodes and the max value at any other node.

The means, for position 0, the `CHECK` phase requires to access $2 \cdot 2 + 1$ memory cells and $1 + 2 \cdot 1$ memory cells during the `UPDATE` phase (see figure 7.8). The same holds for position 2. For position 1, for each of the two final nodes $3\dot{2} + 1$ cells are access during `CHECK` and $1 + 3 \cdot 1$ cells during `UPDATE`. In total, this means 8 cells are accessed for position 0 resp. 2 and $2 \cdot 11 = 22$ cells for position 1. Thus, the average amount of memory cells accessed for `CHECK` and `UPDATE` at a single position is determined as $(8 + 8 + 22)/3 \approx 12.6667$.

In order to generalize this approach, firstly the total number of final nodes $f(n)$ as a function of the duration n of a given request for all possible positions is determined. As depicted in figure 7.7, the number of final nodes for a request of duration n depends on the actual position in the tree, i.e., the start time of the request. W.l.o.g., only the number of *different* positions is considered for

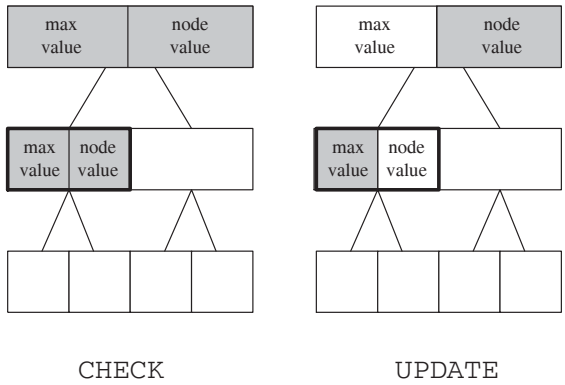


Figure 7.8: Cells accessed during `CHECK` and `UPDATE` phase in position 0. The final node (black frame) is stored in an extra array.

this estimation. This means, for the example shown in figure 7.7 only position 0 and 1 would have been considered. This estimation can be made when n is much smaller than b which is a realistic assumption. The number of different positions depends on the duration of the requests, i.e., for a request of duration n there exist $2^{\lfloor \log n \rfloor}$ different positions until the initial position is reached again.

In the following, it is assumed that $n \ll b$ where $b = 2^k$ denotes the book-ahead interval.

Lemma 7.1 (Number of final nodes) *The total number of final nodes for a given duration n can be determined as*

$$\begin{aligned} f(n) &= 1 + \sum_{i=0}^{\lfloor \log n \rfloor - 1} (i+1)2^i + (n - 2^{\lfloor \log n \rfloor}) \\ &= 1 + 2^{\lfloor \log n \rfloor} \lfloor \log n \rfloor + n - 2^{\lfloor \log n \rfloor} \\ &= 1 + n + 2^{\lfloor \log n \rfloor} \big(\lfloor \log n \rfloor - 1 \big). \end{aligned}$$

Proof. The formula will be proven in two steps: the first step is to prove the validity for $n = 2^k$ for all $k \in \mathbb{N}$. In the second step, the result will be generalized for all $n \in \mathbb{N}$.

Step 1: (complete induction over k). In case $n = 2^k$ for $k \in \mathbb{N}$, the formula can be simplified to

$$f(n) = 1 + \sum_{i=0}^{k-1} (i+1)2^i = 1 + nk.$$

For $k = 1$ follows $f(2^k) = 3$ and hence correctness of the formula (see also figure 7.7).

Let now be $n = 2^{k+1}$. Compared to a duration of 2^k there are 2^k additional positions for the reservation in the tree until the initial position is reached again. For the other 2^k positions, the number of final nodes remains the same as for

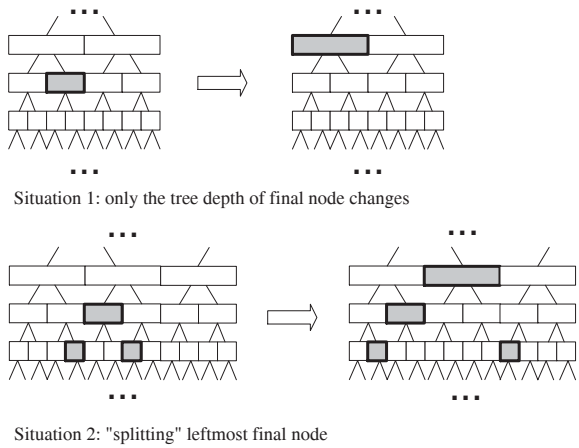


Figure 7.9: Example: Node changes and new final nodes when the duration is doubled ($2^k \rightarrow 2^{k+1}$).

the case $n = 2^k$, i.e., $1 + nk$, and only the depth in the tree of the respective final nodes changes.

The 2^k new positions can be constructed from the old positions by taking the position with the maximum number of final nodes, splitting the leftmost respectively rightmost final node (with duration of at least 2) and adding the split part to the rightmost respectively leftmost node. This is depicted in figure 7.9. For each of those 2^k new positions, there are $\log 2^k + 1 = k + 1$ nodes, and therefore

$$f(2^{k+1}) = 2^k(k+1) + 1 + \sum_{i=0}^{k-1} 2^i(i+1) = 1 + \sum_{i=0}^{k} 2^i(i+1) = 1 + n(k+1).$$

Step 2: In this step, the result from step 1 is generalized to arbitrary values of n. It was already shown, that the formula is valid for $n = 1$. Assuming the validity of the formula was already shown for duration n, for duration $n + 1$ with $n \neq 2^k - 1$ for $k \in \mathbb{N}$, the number of positions does not change compared to duration n. This means, when increasing the duration by 1, two cases can be distinguished:

1. The rightmost final node for duration n has a length of 2^i for some $i \in \mathbb{N}$. Hence, increasing the length by 1 results in the creation of exactly 1 additional node. In total, there are $n/2$ such cases and hence $n/2$ new final nodes.

2. The rightmost final node for duration n has a length of 1. In this case, increasing the request duration by 1 has the effect, that some final nodes are merged. The reason is that 1 node of length 2 comes into existence, and in some cases this node of length 2 merges with another node, etc.

The amount of nodes that actually "disappear" in this process is given by

$$\log n - 1 + \sum_{i=0}^{\lfloor \log n \rfloor - 2} 2^i (\log n - 2 - i) = \log n - 1 + \frac{n}{2} - \log n = \frac{n}{2} - 1.$$

The validity of the formula was already shown for $n = 2^i, i \in \mathbb{N}$ and therefore by combining the results from step 1 and 2, $f(n+1)$ with $n \in \mathbb{N}$ we obtain

$$\begin{aligned} f(n+1) &= f(n) + 1 \\ &= 1 + \sum_{i=0}^{\lfloor \log n \rfloor - 1} 2^i (i+1) + (n - 2^{\lfloor \log n \rfloor}) + 1 \\ &= 1 + \sum_{i=0}^{\lfloor \log n+1 \rfloor - 1} 2^i (i+1) + (n + 1 - 2^{\lfloor \log n+1 \rfloor}) \\ &= 1 + 2^{\lfloor \log\ n+1 \rfloor} \lfloor \log(n+1) \rfloor + (n+1) - 2^{\lfloor \log\ n+1 \rfloor} \\ &= 1 + (n+1) + 2^{\lfloor \log\ n+1 \rfloor} \big(\lfloor \log(n+1) \rfloor - 1 \big). \end{aligned}$$

□

The result stated in lemma 7.1 can now be used to determine the remaining components that are required to determine the average amount of memory cells accessed during admission control.

The average length $l(n)$ of the final nodes can be easily computed as

$$l(n) = \frac{n 2^{\lfloor \log n \rfloor}}{f(n)}. \tag{7.2}$$

This is true because the duration of the request was assumed to be n and a total of $2^{\lfloor \log n \rfloor}$ different positions in the tree exists. Furthermore, the average number of final nodes $\hat{f}(n)$ can then be computed as

$$\hat{f}(n) = \frac{f(n)}{2^{\lfloor \log n \rfloor}}. \tag{7.3}$$

With these results, the average amount of memory cells accessed during a complete `CHECK` and `UPDATE` can now be calculated. It is assumed, that each `CHECK` phase requires to access the memory cells starting from the root node to each final node. During this phase, both the node value and the max value of each node must be accessed, i.e., 2 memory cells per node. During the `UPDATE` phase, in the worst case the same memory cells need to be accessed, in this case only 1 cell per node since either the node value (for final nodes) or the max value (for the nonfinal nodes) are updated.

The nature of the implementation requires that firstly all the `CHECK` phases for each link must be performed and only in case all these checks were successful the updates are made. This requires to store the final nodes[3] found during the `CHECK` phase. The implementation presented here uses an additional array to store these final nodes. This means for each `CHECK` and `UPDATE` phase, $\hat{f}(n)$ additional memory cells must be accessed.

[3]Final nodes are start nodes for the `UPDATE` phase.

The number of nodes accessed during a complete `CHECK` and `UPDATE` phase depends on the book-ahead interval $b = 2^k$ and the request duration n and is given as follows

$$\begin{aligned} a_{\text{tree}}(b,n) &= \underbrace{2\hat{f}(n)}_{\text{array storage for final nodes}} + \underbrace{2\hat{f}(n)\overbrace{(\log b - \log l(n) + 1)}^{\text{distance from the root node}}}_{\texttt{CHECK}} \\ &\quad + \underbrace{\hat{f}(n)\,(\log b - \log l(n) + 1)}_{\texttt{UPDATE}} \\ &= 2\hat{f}(n) + 3\hat{f}(n)\,(\log b - \log l(n) + 1)\,. \end{aligned} \tag{7.4}$$

With given book-ahead interval, this formula allows to determine the minimal duration n for which the usage of trees pays off, i.e., faster admission times are achieved. For example, $a_{\text{array}}(n) = a_{\text{tree}}(65536, n)$ for $n \approx 138.7489$ with $b = 65536$.

When using this formula for the example shown at the beginning of this section, we obtain $a_{\text{tree}}(4, 2) \approx 14.6323$. The difference to the previously computed value of ≈ 12.6667 results from including only positions that are different in (7.4). This requires n being much smaller than b which is not the case in our simple example.

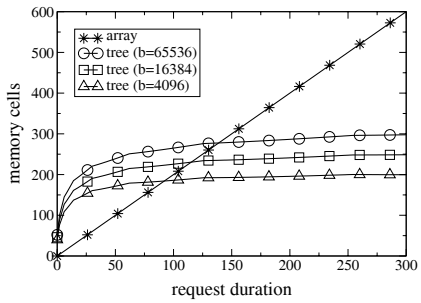


Figure 7.10: $a_{\text{tree}}(b, n)$ compared to $a_{\text{array}}(n)$ for three different book-ahead periods b

In figure 7.10, the values of $a(b, n)$ are depicted for $b = 4096$, 16384, and 65536 and varying n and compared to the respective numbers for arrays.

7.4.3 Support for Flow Scheduling and Malleable Reservations

In order to support more sophisticated allocation strategies such as finding the first suitable interval for a reservation of given length or support for malleable reservations (see section 5.2), it is required to scan intervals larger than the requested duration n and also to determine the average utilization for those intervals. Besides required for flow scheduling, such functionality is also necessary to support routing algorithms based on other than simple shortest path routing, such as **Widest/Shortest** or **MinLoad/Shortest** (see section 5.1). In this section, the considerations of sections 7.4.1 and 7.4.2 are extended to examine the suitability of both data structures for malleable reservations. In case,

a suitable transmission interval is found, the complexity of the `UPDATE` phase remains the same as described in the previous sections. Hence, in the following only the `CHECK` phase is examined which requires scanning the complete search interval.

Two additional request types are distinguished: *malleable reservations* which require accessing each individual time slot within a given search interval and an *interval search* which has the purpose to determine the first available transmission interval for a request of given duration and bandwidth.

Array

Using arrays, such functionality can be realized using a linear scan over the search interval. For given search interval length t, this means the scan can be implemented with at most t memory cells of the array being accessed, independent of the request duration n. This holds for both the worst case for the search for the first time interval with sufficient duration n and bandwidth and for the support of flow scheduling where the average or peak utilization within the whole search interval needs to be determined. Formally, this means

$$a_{\text{array/mall}}(t, n) = a_{\text{array/ival}}(t, n) = t. \tag{7.5}$$

Segment Tree

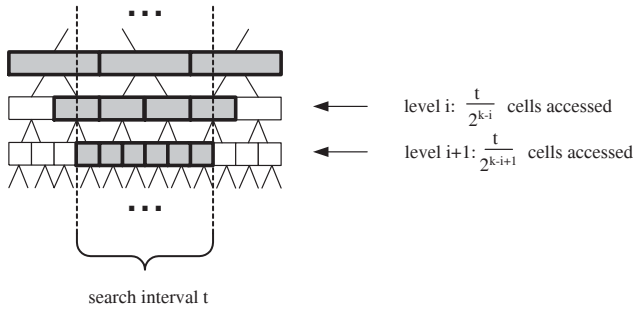


Figure 7.11: Accessed nodes in malleable scenario. Any node covering a part of the search interval must be accessed to obtain utilization for any single time slot within the search interval. The tree descend proceeds until the leaf nodes are reached.

As described before, supporting malleable reservations may require knowledge about the link utilization at every single time slot. However, the tree was designed to hide this information as much as possible and only store information for larger periods of time in higher tree levels. In order to determine the utilization for every single slot with an interval of length t, $t/2^{k-i}$ nodes need to be accessed on tree level i with $b = 2^k$ being the duration of the book-ahead interval (see figure 7.11). In contrast to the `CHECK` phase described in section 7.2, in this case only the node value of each node needs to be accessed and the tree descend always proceeds until the leaf nodes of the tree are reached. When t denotes the duration of the search interval, the amount of memory cells accessed yields

$$a_{\text{tree/mall}}(k,t) = \sum_{i=0}^{i=k} \frac{t}{2^{k-i}} = 2t - \frac{1}{2^k}. \tag{7.6}$$

The search for the first time interval of duration n within a larger search interval of duration t which satisfies the bandwidth constraints - in order to support the strategies `Min/Start`, `Min/End`, etc. - requires less effort, since initially a `CHECK` phase as described in section 7.2 can be performed for a request of duration n starting at the beginning of the search interval, followed by checking each slot within the remaining slots of the search interval. In the worst case, every slot within this remaining interval needs to be checked. The initial check can be bounded using the formula from section 7.4.2 for the `UPDATE` phase because only one cell needs to be accessed for this check. This leads to the following estimation

$$\begin{aligned} a_{\text{tree/ival}}(k,t,n) &= \underbrace{\hat{f}(n)\,(k - \log l(n) + 1)}_{\text{initial check phase}} \quad + \quad \underbrace{\sum_{i=0}^{i=k} \frac{t-n}{2^{k-i}}}_{\text{remaining interval}} \\ &= \hat{f}(n)\,(k - \log l(n) + 1) + 2(t-n) - \frac{1}{2^k}, \end{aligned} \tag{7.7}$$

where t denotes the duration of the search interval, n denotes the requested duration, and k determines the length of the book-ahead period with $b = 2^k$. While equation 7.6 represents an upper bound for the number of memory cells being accessed, in equation 7.7 only an average is given because $a_{\text{tree/ival}}(k,t,n)$ also depends on the requested duration n.

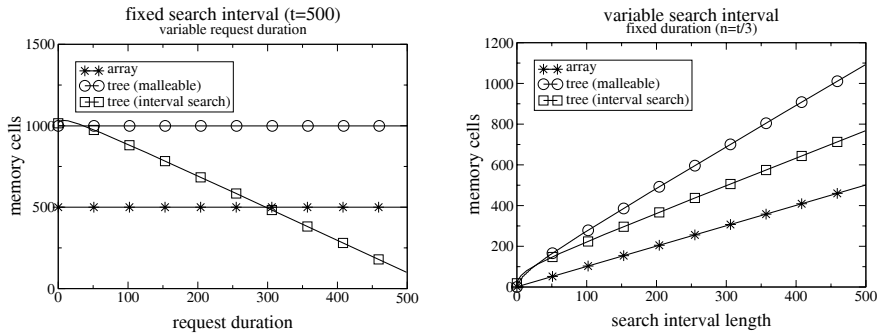


Figure 7.12: Memory cells accessed during `CHECK` phase: fixed search interval size $t = 500$ (*left*) vs. fixed request duration $n = t/3$ (*right*) for $k = 16384$.

In figure 7.12, the amount of memory cells accessed when admitting a flexible reservation request[4] is given. When accessing any single time slot and fixed search interval t, the time required using trees is approximately twice as large as that using arrays, independent of the request duration (see equation 7.6). Whereas the search for the first suitable interval achieves similar results than arrays when the request duration n reaches the size of the search interval t. For

[4] *Flexible* here means movable or malleable.

malleable requests, the requirements using trees is more than doubled compared to arrays. The same holds for the case of variable search interval t and request duration $n = t/3$. The performance advantage of arrays is even more evident in this case. Trees cannot achieve a similar performance at all. Due to the influence of the requested duration n, the curve for the interval search increases more slowly than the one related to malleable reservations.

7.4.4 Worst-Case Memory Requirement

The worst-case memory requirement of the data structures is another important performance aspect which can considerably impact the applicability of a data structure in a given environment. For the following computations, a memory requirement of 4 bytes per integer and 4 bytes per pointer is assumed.

When b is the length of the book-ahead interval, i.e., the total number of time slots covered, arrays require $4 \cdot b$ bytes. The original implementation of the segment tree uses 20 bytes per tree node: 4 bytes each for the max and node values, 4 bytes for the pointer to the child nodes and 4 bytes for start and stop time. The memory requirement can be reduced by 8 bytes by not storing the start and stop times within the nodes but computing them during the traversal. In addition to that, using dynamic memory allocation it is possible to further reduce the requirement during the run-time of the broker. However, in the worst case there is still a worst-case requirement of $(2b - 1) \cdot 12$ bytes when start and stop times are not stored. In case, the segment tree is stored using an array (i.e., each tree node corresponds to an element of the array), the memory for pointers to child nodes can be saved. This leads to a worst-case memory requirement of $(2b - 1) \cdot 8$ bytes which is 4 times the memory allocation to be made for arrays.

Concluding, the analysis shows that arrays have a significant advantage in terms of memory requirement (see section 7.4.4). With respect to the number of memory cells accessed during the `CHECK` and the `UPDATE` phase, it was shown that trees only have an advantage when the requested duration is rather long. Arrays were competitive up to a duration of approximately $90 - 140$, depending on the duration of the book-ahead interval. However, when it is required to obtain the utilization for single time slots within a given search interval as is the case for scheduling malleable reservations, the segment tree cannot compete with arrays at all, i.e., the performance of arrays is never reached. This is independent of the request duration. Only in certain cases, when the first suitable time interval for a transmission is requested, trees can reach the performance of arrays (see figure 7.12).

7.5 Performance Measurements

In order to determine how the results previously described impact the performance in an actual application of the data structures, tests were made using the bandwidth broker implementation as described in chapter 3. The implementation has the full functionality described in chapter 3, i.e., supports fixed and malleable reservations. The results presented here may not be representative for any possible environment, in particular the measurements involving the whole bandwidth broker are very much implementation dependent. However, the figures presented here illustrate the effects when applying the data structures in

an actual environment and give an idea of how the performance is affected.

7.5.1 Environment

Two types of tests were made. Firstly, the performance of only the single data structure (segment tree or array) was measured. Secondly the time required for admission control in the bandwidth broker was examined using the cost239 topology (see section 4.3.1). This allows to evaluate how the performance of the data structure influences the overall admission time of the bandwidth broker. In addition to the admission control task, in the second environment also the path computation in the network is included. In case the available bandwidth of a chosen link is not sufficient for the requirement of a given request, an alternative path must be found and checked. The measurements presented here were made on a 1 GHz Pentium IV PC with 1GB of main memory running Linux.

The segment tree implementation variant embedded in arrays was used for the time measurements. The tests related to the memory consumption (see section 7.5.5) were made using the memory saving variant based on the pointer implementation with dynamic memory allocation.

7.5.2 Performance of the Data Structures

Initially, the admission time of the data structures as a function of the duration of the reservation request was measured. Every single reservation request was made on an empty data structure, i.e., no other reservations were present. For these results, only the time required by the functions accessing the data structures was measured, i.e., there was no overhead introduced by the bandwidth broker framework.

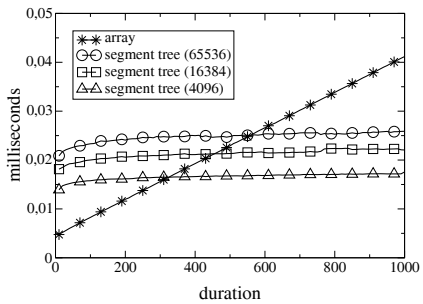


Figure 7.13: Admission speed as a function of the duration for book-ahead intervals of 4096, 16384, and 65536 slots.

Figure 7.13 shows the measure admission times for the array and the tree implementation. In order to generate these results, successful requests of given duration were simulated, i.e., both the `CHECK` and `UPDATE` phase were performed. For each duration, a request was generated for any possible start time within the book-ahead interval. Thus, each sample point represents the average of the measured times for both phases at each possible slot and hence, these measurements are comparable to the analytic result depicted in figure 7.10.

In general, the curves are similar as those determined by the analysis with the only difference that arrays perform even better. It can be observed that

arrays are competitive up to a reservation length of approximately $300 - 500$ slots which is about 3 times more than computed in section 7.4.2. This is due to the actual implementation which, besides the number of memory cells accesses, requires other operations such as function calls or index computations. In this context, trees are more complicated to implement and hence some more administrative framework is required which also affects the actual run-time of `CHECK` and `UPDATE` routines.

7.5.3 Multi-Link Admission Control

The results of the analysis and the previous measurements imply that request sets as used in [SNNP99], i.e., with a book-ahead interval of approximately 8000 slots and extremely short reservation durations between 4 and 32 slots are inappropriate for a performance comparison because trees are not competitive under these conditions. Instead, the parameters as used for the simulations shown in previous sections are applied here, using various mean request durations.

The tests were made using the cost239 topology (see figure 4.5). For each outgoing interface, admission control has to be performed, i.e., a total of $2 \cdot 26 = 52$ data structures representing link utilization is kept in the bandwidth broker. Besides the time require to access the data structures, in this setting also the path computation - in this case using the **Widest/Shortest** algorithm - and the other administrative tasks, e.g., storing information about accepted requests, are included in the run-time. Using this setting, the average path length for accepted requests was 4.04.

The results shown here may of course vary depending on the actual bandwidth broker implementation and network topology. However, the figures give a hint on how the admission speed evolves when applying the data structures in a bandwidth broker.

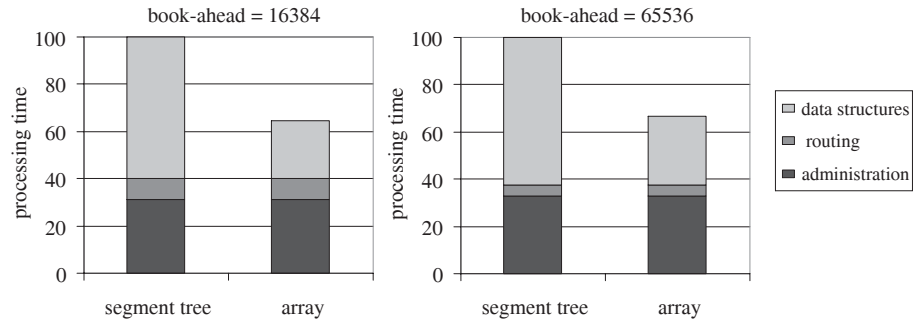


Figure 7.14: The processing time of the bandwidth broker divided into the different tasks *routing*, data structure related processing, and the remaining parts (*administration*) to be performed by the broker. The average request duration was 500 slots.

The processing time of the bandwidth broker is depicted in figure 7.14, decomposed into the time spent for each of the three main parts *data structures*, i.e., mostly `CHECK` and `UPDATE` operations, *routing*, and the remaining *administrative* operations to be performed, e.g., storing information such as source and destination node for admitted requests. The figure shows, that using segment

trees almost 60% of the total processing time of the broker is required for data structure related operations, whereas the processing time can be reduced to approximately 30% when using arrays instead. The overall performance difference results completely from using different data structures, the remaining parts of the broker are not affected in their run-time. The results were measured using an average request size of 500 slots which denoted the "break-even" point of arrays and tree as indicated in figure 7.13.

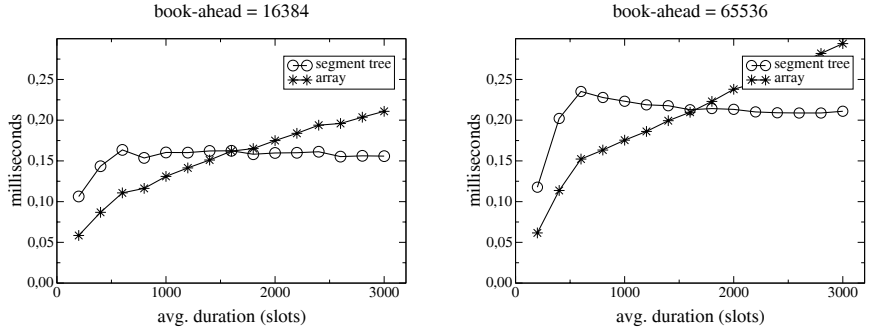


Figure 7.15: Average admission speed for a single request depending on the average requested duration. The request set was used with varying average duration in order to generate these results.

Since this does not hold in the multi-link scenario, the average admission speed depending on the average request duration is depicted in figure 7.15. As indicated by the plots in figure 7.14, the actual length of the book-ahead interval is less important in this case. It can be observed, that using the multi link scenario, the "break-even" point is reached at a request duration of approximately $1,500$ slots, i.e., three times more than measured for a single data structure.

7.5.4 Malleable Reservations

Performance of the Data Structures

In order to examine the impact of the data structures in case of interval search and malleable reservations, time measurements were made that underline the analytical results presented in section 7.4.3.

In figure 7.16, the average time required for the only the `CHECK` phase when scanning an interval of fixed size (500 slots) in a single data structure is depicted. Alike in case of fixed reservations, it can be clearly observed that trees have an even larger disadvantage than shown in the analysis (see figure 7.12). In order to obtain those results, the situation as analyzed in section 7.4.3 was simulated, i.e., for each request the worst-case scenario was simulated, i.e., the maximal number of tree nodes had to be accessed. Using arrays, the actual times measured here are shorter than e.g., those shown in figure 7.13, since here only the `CHECK` phase was measured.

Multi-Link Admission Control

As depicted in figure 7.17, when processing malleable requests in the multiple link scenario, the percentage of the overall processing time spent with data

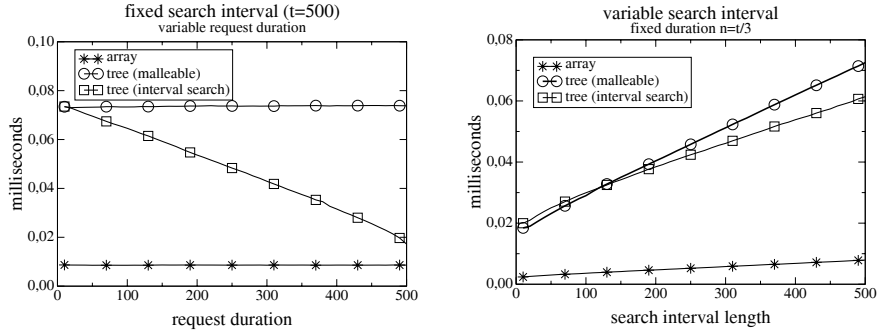


Figure 7.16: Single data structure: Measured time of only the `CHECK` phase for malleable requests: fixed search interval size $t = 500$ (*left*) vs. fixed request duration $n = t/3$ (*right*). The results were measured using a book-ahead period of $b = 16384$, i.e., $k = 14$.

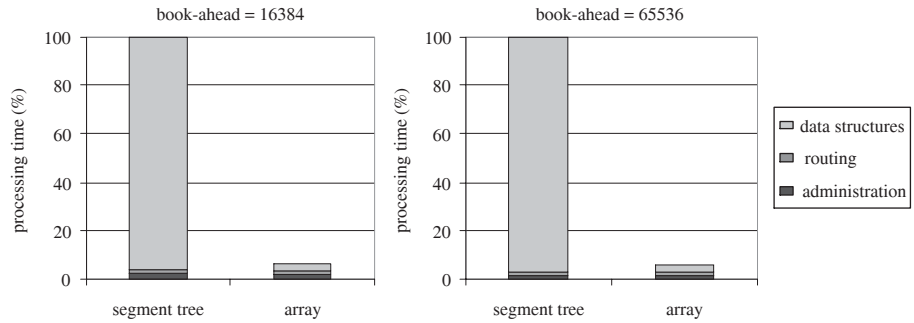


Figure 7.17: Processing time of the bandwidth broker for malleable requests using the `Min/Start` strategy. The average search interval length was 1000 slots.

structure related functions is significantly increased compared to the results shown in figure 7.14. It can be observed, that the time spent in the data structures nearly completely determines the total processing time. The diagram shows the average processing time required for the whole request set in the medium load scenario (1.4 GB/s). The request sets used to generate these figures only contained 30% malleable requests. These figures show, that the actual choice of the data structures plays an important role for the admission speed of the bandwidth broker, in particular, using segment trees results in about 10 times higher processing time compared to arrays.

In figure 7.18, another important factor with considerable impact on the processing time is depicted. It can be observed that, the maximum processing time for a single malleable request, computed over the whole request set, is up to 100 times larger than the average. In particular, the maximal processing time was several seconds for a single request. The reason is that using the **Min/Start** strategy, some malleable requests can be admitted very quickly without performing a large number of checks, whereas in the worst case, i.e., a rejection of the malleable requests, the amount of checks to be performed is significantly higher, as outlined in section 5.2.2. Therefore, the actual network load is an important factor with respect to the admission time required for a

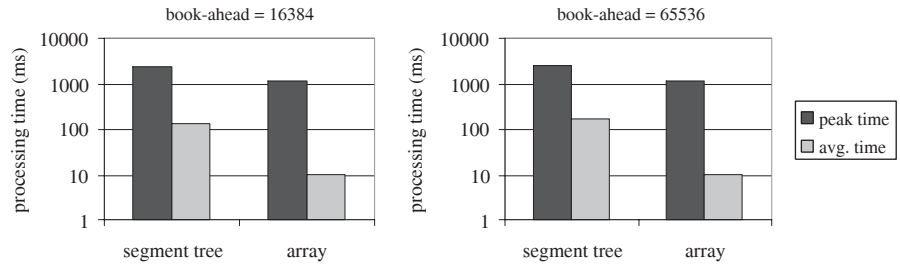


Figure 7.18: Peak processing time for a single malleable request compared to the average (logarithmic scale). The average search interval length was 1000 slots.

single malleable request.

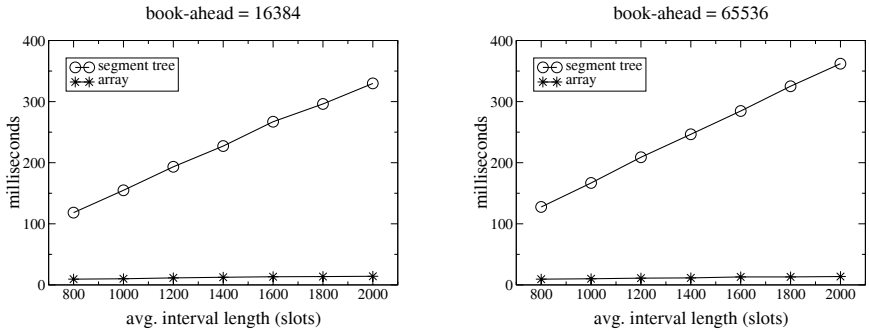


Figure 7.19: Processing time as a function of the search interval length.

The dependency of the processing time on the average search interval length is depicted in figure 7.19. The diagram shows the similar effect as outlined figure 7.16: the advantage of array increases with the search interval length. The length of the book-ahead interval is less important. Although the processing times rise slightly in the scenario with 65536 slots, in general the difference between trees and arrays remains stable. The reason is, that using trees the malleable scenario requires frequently accessing leaf nodes, i.e., traversing 14 (using a book-ahead interval of 16384) or 16 (using a book-ahead interval of 65536) tree nodes. Hence, only 2 additional nodes must be accessed for each traversal which influence the overall processing time only moderately.

7.5.5 Memory Consumption

In the previous sections, only the admission speed using trees or arrays was examined. Memory consumption is another important factor for the applicability of a data structure in the given environment. In this case, using the tree statically embedded in an array has a great disadvantage over arrays with 4 times higher memory consumption (see section 7.4.4). In section 7.2.1, another implementation alternative was presented, based on pointers and dynamic allocation of tree nodes during run-time. Although this tree variant has an even larger disadvantage in terms of admission speed, for the sake of completeness it is also tested with respect to the memory performance. For that purpose, the same

parameters as previously used were chosen to simulate the multi-link scenario with the cost239 topology.

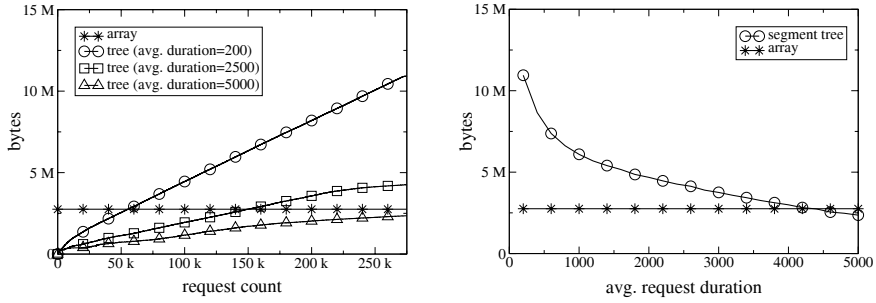


Figure 7.20: Memory consumption of array and segment tree. The diagrams show the dynamic behavior for a complete set of requests with varying mean request duration (*left*) and the final memory requirement (*right*).

In figure 7.20, it can be observed that the usage of dynamic memory allocation has a significant advantage over the static tree implementation which requires 4 times more memory than arrays. However, trees are still not as memory efficient as arrays. On the left, the development of the overall memory consumption of all data structures in the bandwidth broker is depicted using different average request durations. The memory consumption heavily depends on this parameter. On the right hand side, this is outlined. It can be observed that trees retain the worse memory performance up to an average duration of $4,500$ slots. This means, trees in this case show an even worse performance than in terms admission speed where equal performance could be achieve at approximately $1,500$ slots. Hence, the dynamic memory allocation does not provide any advantage.

7.6 Other Approaches

In addition to the segment tree, in [SNNP99] a *binary search tree* was presented which also uses slotted time however, the number of tree nodes depends on the amount of admitted requests. Furthermore, this approach has the drawback of getting unbalanced over time. Hence, balancing is required periodically which results in significant performance disadvantages and therefore, this tree was not considered here.

7.7 Conclusion

The most important result of the examination of both data structures is that arrays, being one of the simplest conceivable data structures, have a significant advantage over segment trees below a certain average request duration which in the previously described environment covered any of the expected application scenarios. However, when it is guaranteed that the average request duration is above this point, it may be reasonable to use segment trees, when the higher memory consumption can be tolerated. The picture is different when investigating reservation methods which need to scan larger intervals, in particular with

the additional requirement of having access to the resource allocation of every single time slot. In such a case, trees are unsuitable and - apart from some rather unlikely cases such as requests covering the whole book-ahead interval - always result in a worse performance. Since request types such as malleable requests are one of the key improvements in advance reservation environments compared to an immediate scenario, it is likely that in a real-world implementation especially this request type will be of significant importance.

The easy implementation, also for the dynamically advancing time, is an additional factor in favor of arrays. Concluding it can be stated, that the usage of arrays is favorable in the examined environments. In particular, when using the flow switching approach described in section 5.1.4, only short time intervals need to be checked, which leads to a significant advantage of arrays. The analysis showed that the superiority of arrays results from the property, that the information is stored locally at each array element rather than distributed across several nodes as is the case for the tree.

The memory consumption of trees was far worse compared to arrays using any of the tested implementations. Even with dynamic memory allocation, trees were only competitive with very long average request durations. In particular, those durations were several times longer than those were the static tree implementation became as fast as arrays.

The measurements presented here were made using the application in network management systems. However, the analytical results and the measurements of the individual data structures can also be applied to any other resource management system with support for advance reservations, for example the one described in [FKL+99]. Although the measured times depend on the actual implementation and hardware platform, the general result that arrays provide a much better performance up to a certain average request duration can be generalized to other areas of applications.

Chapter 8

Multi-Domain Architectures

The techniques discussed in the previous chapters were aimed at providing an advance reservation service within a single domain. As is the case in larger environments, it is important to study the impact of these techniques on the scalability of the system in terms of the ability cover multiple domains. Scalability of a single bandwidth broker mostly concerns the capabilities to deal with a large quantity of flows at a time. This affects the data structures studied in chapter 7. The other aspect of scalability is how to deal with the presented techniques in a multi-domain environment, such as the Internet, where networks domains of more than a single ISP must be traversed. Furthermore, even within the network of a single ISP several bandwidth brokers may be installed for load balancing purposes.

While in general, communication between different brokers can be modeled alike the communication between a normal client and the respective bandwidth broker, the additional performance optimization techniques presented in this thesis need to be examined with respect to their suitability for networks with more than a single domain. The failure recovery mechanisms described in chapter 6 can be excluded from this examination since the demand for quick responses to failures prohibits long lasting communication among different brokers. Furthermore, also all the optimization techniques that are transparent for clients can be excluded from this analysis. These are the general routing decision which must be handled within each domain, the flow switching approach, and also the off-line optimization which can be performed within a single domain or sub-domain without interacting with other brokers. In contrast, malleable reservations may require additional considerations in multiple domain scenarios since the timing affects each domain between source and destination of a given traffic flow.

Before describing details of the different opportunities to realize inter-broker communication for the previously described purposes, a brief overview of previous work on this field is given.

8.1 Overview

The aspect of communication between different network management systems has been discussed by several previous publications, however almost exclusively for the immediate reservation scenario.

The concept of bandwidth brokers as described in [NJZ99] also addressed the issue of multi-domain environments. In such a case, it was proposed to use domain-by-domain forwarding of reservation requests. The general approach was to provide only coarse grain reservations for flow aggregates. In such a case, it is possible to define a certain amount of traffic that is routed through other domains without explicit reservation. This means, ISPs define how much traffic from other domains can be accommodated by their own network, e.g., using SLAs. Such an SLA can be conceived as a coarse grain advance reservation that lasts as long as the corresponding SLA between two ISPs [ZOS00].

In the advance reservation scenario this SLA-based approach is also applicable. Furthermore, it is possible to make SLAs time-dependent, i.e., SLA parameters are dependent on the time and can be conceived as advance reservations for aggregated flows. For example, it may be reasonable to require less stringent bandwidth constraints during night time which usually low traffic. Thus, the dynamic SLA parameters can be taken into account by bandwidth brokers.

In [GB99], different signaling strategies for notifying and negotiating SLAs, i.e., flow aggregates, in DiffServ environments were proposed and compared. The focus was the trade-off between fine grained per-flow reservation and coarse grained aggregate reservations specified by an SLA. The authors consider the per-flow reservation approach infeasible due to the extensive signaling overhead and instead propose two approaches that limit the number of signaling messages by using massive overprovisioning or signaling only in case of "significant" changes of the flow characteristics. The two basic approaches, i.e., per-reservation signaling and the definition of SLAs, proposed in [GB99] were examined to their suitability to support advance reservations, and the results are presented in this chapter.

In contrast to reserving only coarse grain flow aggregates, which is suitable for inter-ISP traffic, in order to reserve resources between the different bandwidth brokers of a single ISP, more fine-granular reservation requests must be considered. Solutions for dividing a large network into smaller sub-domains each managed by a bandwidth broker was presented in [RG02]. Also within such a network of sub-domains, support for the different new services provided in the advance reservation environment (see section 3.2) is required.

The advance reservation bandwidth broker presented in [San03] includes only a brief description of such inter-domain mechanisms, assuming that the corresponding protocols work exactly like in immediate reservation environments. While this may be conceivable for inter-ISP traffic, a more elaborate view is required for the case of intra-domain traffic within a single network domains. In such a case, the brokerage mechanisms supported by the BB, for example, malleable reservations, may also be of interest on the intra-domain level. Although it is possible the reserve resources for flow aggregates, this may result in a wastage of bandwidth within an ISPs network. In section 3.4.1, the SIBBS protocol [Tea01] for inter-broker communication was mentioned which can be used to manage such SLAs.

In [SP98a], the problem of *reservation agents* (bandwidth brokers) in multi-

domain environments with support for advance reservations was addressed. The main idea of the paper - similar to the consideration sin [GB99] - was to aggregate flows in order to reduce the complexity of the signaling and negotiation process and keep the architecture scalable. For each request with the destination domain *outside* the source domain, the agent located within the source domain is responsible for identifying an agent in a domain closer to the destination. Thus, the request is forwarded from to domain to domain until eventually the destination is reached. In order to aggregate flows sent across domains and thus, to reduce the number of messages sent between different agents, the idea is to make large bulk reservations in advance and to aggregate individual microflows into these bulk reservations.

8.2 Architectures

In the following sections, the notion *domain* means a large network which in turn can be composed of several smaller domains, called *sub-domains*. For example, a domain in this context describes the network of a single ISP which, for administrative reasons, contains a number of sub-domains. Each sub-domain is managed by a single bandwidth broker. The different implementation alternatives as indicated in the previous section are depicted in table 8.1.

Table 8.1: Multi-Domain Reservation Architectures

SLA based	SLA negotiated between domains determine the amount of inter-domain traffic domain-by-domain signaling
per-flow reservation	direct communication between two bandwidth brokers: 1:1 domain-by-domain signaling 1:n multicast
hybrid	SLAs between ISP domains per-flow reservation within ISP sub-domains

8.2.1 SLA-based Inter-Domain Reservations using Flow Aggregates

This approach defines only coarse-grained reservations for flow aggregates based on SLA negotiation. This means, no signaling is required for a single reservation. Instead, the total amount of traffic that is send through a given domain is negotiated in advance. This quantity of traffic may depend on the time.

The advantage of this approach is, that no signaling is required for a single reservation. This is particularly useful for the services described in section 3.2. Determining a suitable reservation period for a given amount of data may require more than a single round of negotiation.

The setup is depicted in figure 8.1, showing an example for an SLA with four domains being involved. The SLA for traffic from domain A to domain D consists of three single SLAs between domain A and the three other domains B,C, and D respectively. The result is, that upon successful negotiation - which

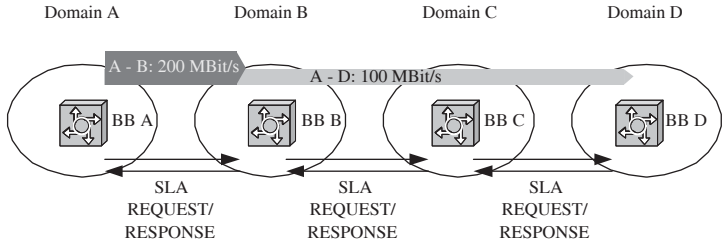


Figure 8.1: SLA negotiation and setup between different domains.

is done using the domain-by-domain forwarding of the request - the resulting SLAs allow 100 MB/s of traffic from domain A to be injected into each of the other domains. This covers only the situation for establishing an SLA for the traffic from A to D. Further SLA may be established for example for the traffic from A to B of another 100 MBit/s. Consequently, in this example the total amount of traffic allowed to be sent from A to B, independent of its final destination within B, is 200 MBit/s.

The major disadvantage is its inefficiency. In case, the SLA provides more than the required bandwidth, the unused part cannot be allotted to other reservations and hence is wasted. However, it is also in the reserving domains interest not to waste bandwidth which must be paid for. Hence, renegotiation may be dynamically carried out in order to adjust the SLA to the actual needs. However, it must be avoided to perform renegotiations on a per-flow basis. In order to overcome this problem, in [GB99] it is proposed to implicitly define SLAs by measuring the traffic capacity with adjacent network domains. Furthermore, when sufficient bandwidth between two domains is available, an SLA may not be renegotiated when higher capacity within the available constraints is required. This scenario was called *limited notification*.

Time-dependent SLAs - which are in turn also advance reservations - can reduce the necessity for renegotiations. Such SLAs can be applied whenever the amount of required bandwidth is known in advance. If an SLA is time-dependent, renegotiation can be very easily achieved by adjusting the bandwidth requirement during certain periods of time with similar traffic profile in advance. For example, less bandwidth may be required at night than during the day.

If more than one domain is to be traversed, SLAs must be established with each domain on the path from the source to the destination domain. Within source and destination domain, the actual reservations must be made. In this context, cooperation between the respective bandwidth brokers is necessary. The coordination in case of reservations without fixed reservations can be made using one of the per-reservation approaches described in the following sections.

8.2.2 Per-Reservation Signaling

This approach may be more useful within the domain of a single ISP since it uses per-reservation signaling for every single request. On a large scale, this is infeasible due to the large communication and negotiation overhead.

Several opportunities exist to implement per-reservation signaling which are described in the following sections. In general, it is assumed that n denotes the

number of domains involved in a reservation request.

Domain-by-Domain Reservation (1:1)

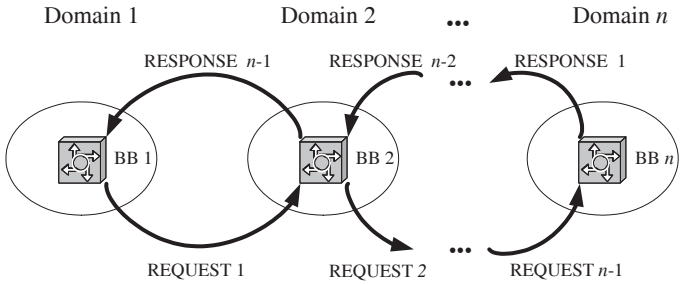


Figure 8.2: Domain-by-domain reservation model

This approach was also described in section 2.1.2: each bandwidth broker contacts its successor on the path from source to destination domain in order to establish a reservation. If the reservation fails in a domain, the process is finished and the negative result returned to the source domain (see figure 8.2).

The disadvantage in this case is that malleable reservations are costly to implement. Using the basic approach, in order to test n different "shapes" for a given transmission a message must be sent n times through the whole path of domains.

Table 8.2: Domain-by-Domain Forwarding: Parameters

	fixed	malleable
# messages	$2(n-1)$	$2(n-1)(t_{\max}-t_{\min})(b_{\max}-b_{\min})$
admission time	$t_{rtt}+nt_{adm}$	$(t_{rtt}+nt_{adm})(t_{\max}-t_{\min})(b_{\max}-b_{\min})$

Table 8.2 shows the message complexity and the resulting admission time for processing a single reservation request. It is assumed, that t_{rtt} denotes the total round trip time required for sending a request message to the destination domain and the response message back to the source domain. Furthermore, n denotes the number of domains involved and t_{adm} denotes the admission time on a single broker. For malleable requests, the same notation as in section 5.2 is used to denote the upper respectively lower bounds of transmission rate (b) and search interval (t).

Multicast Communication (1:n)

In this case, one bandwidth broker acts as master and coordinates the other bandwidth brokers involved (see figure 8.3). Using multicast, this architecture works as for the domain-by-domain approach with the only difference, that time can be save by multi-casting the request to any of the brokers involved. However, this approach suffers from the same drawbacks as the domain-by-domain approach and requires knowledge about the network status, i.e., which domains will be traversed by the flow for which the reservation is made.

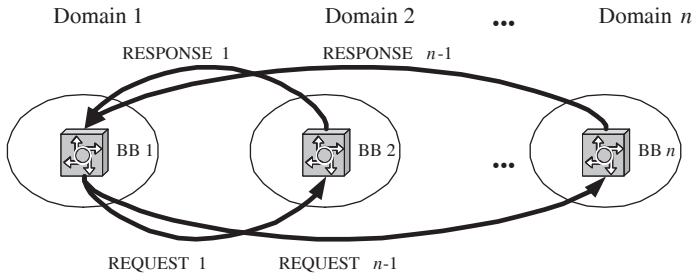


Figure 8.3: Domain-by-domain reservation model for multicast communication.

Table 8.3: Multicast Communication: Parameters

	fixed	malleable
# messages	$2(n-1)$	$2(n-1)(t_{\max}-t_{\min})(b_{\max}-b_{\min})$
admission time	$t_{rtt}+t_{adm}$	$(t_{rtt}+t_{adm})(t_{\max}-t_{\min})(b_{\max}-b_{\min})$

As depicted in table 8.3, the multicast solution achieves parallelism of the computations, thus saving the time waiting for the result from a preceding bandwidth broker.

Exchange of Status Information

In order to reduce the admission time on cases where several bandwidth brokers need to communicate, it is possible to exchange status information about the affected period of time between the brokers in the $1:n$ and the hierarchical $m:n$ model. This reduces the number of message exchanges whenever malleable reservation requests are processed. This approach requires to send the status of each sub-domain's network links to a single broker which then acts as the only admission instance for the given request on behalf of the bandwidth brokers involved. Besides the exchange of status information, which raises security issues, this approach requires to accept the admission decision of the master broker. While this is relatively easy to implement within the network of a single ISP, it is unrealistic to assume an implementation across different ISP networks.

Table 8.4: Exchange of Status Information

	fixed	malleable
# messages	$2(n-1)$	$2(n-1)$
admission time	$t_{rtt}+t_{adm}$	$t_{rtt}+t_{adm}$

In table 8.4, the parameters are depicted when using the status exchange approach together with the $1:n$ multicast solution. While the time and complexity for fixed reservations does not change, the time for admitting malleable requests is considerably lower when using this approach.

However, this approach requires to exchange status information for each malleable reservation. Thus, the protocol overhead caused by the increased

length of each message can rise considerably with large number of malleable reservations.

8.2.3 Hybrid Model

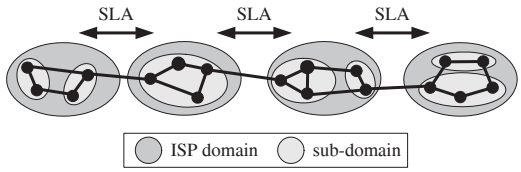


Figure 8.4: Hybrid model: SLAs on inter-ISP layer and per-flow reservation signaling on sub-domain layer.

It is conceivable to implement a hybrid model which uses SLA-based aggregate reservation between ISP networks and per-reservation signaling for traffic within the sub-domains of each ISP network (see figure 8.4) in a straightforward manner. Furthermore, in case the negotiated SLAs do not suffice for a given bandwidth reservation, one of the per-reservation models can be used to adjust the available bandwidth.

8.3 Conclusion

Using advance reservation across multiple domains increases the overhead of the whole reservation system. Using the SLA-based approach which does not require frequent updates of the SLAs, the overhead is tolerable. As stated in [SP98a, GB99], the usage of per-flow reservations on a large scale increases the amount of message exchanged between brokers considerably and hence cannot be applied in a sensible manner. In particular, if malleable reservations are to be processed this overhead adds up to an enormous amount. Thus, the SLA-based approach is the most applicable, since it allows brokers to reserve larger bandwidth quantities and to aggregate the microflows within these larger reservations as described in [SP98a]. In certain cases, using the exchange of status information for a certain, limited number of malleable reservations can be a realistic alternative to reduce the total amount of messages. However, in general the advance reservation model is not designed to cover environments with millions of flows. Instead, applications such as grid computing, e.g., using a dedicated network partition will not generate the same amount of flows as the common Internet.

Chapter 9

Conclusion

In this thesis, the different performance aspects important in the context of advance reservations in computer networks were presented and examined. Based on a bandwidth broker architecture using MPLS in order to control the network, a number of additional services and techniques were developed which improve the network performance compared to the implementations in today's networks. The term *performance* in this context comprises the various aspects important for network operators and clients, i.e., the performance of the network in terms of the amount of admitted flows, and the amount of bandwidth carried by admitted flows. For clients, also the increased admission probability, the response time of the management system, and the behavior in case of network failures are important and were examined within this thesis.

While for many applications, such as web browsing and transmission of small or medium amounts of data, QoS guarantees are not required, future scientific applications in the field of grid computing with vast amounts of data to be transmitted over networks demand for more than currently possible. In computer networks, even the opportunity to reserve resources in advance is currently not available and its reliability to plan transmissions in advance can be defined as a quality-of-service guarantee of its own.

In general, advance reservations improve the usability and performance of the network in various ways and are an interesting opportunity for customers and operators of a network. The improvements for clients, such as increased admission probability or additional services, are evident. However, also network operators can benefit from the implementation of advance reservation mechanism when not only the very basic opportunity to reserve in advance is offered but also the additional mechanisms for improving the overall network performance are implemented. Although in general the efforts to enable advance reservations rise compared to an immediate reservation service, the additional processing time for routing, flow switching, or the usage of malleable reservations, is worthwhile as shown by the performance results.

Besides the additional services that can be implemented in an advance reservation environment, such as search for suitable transmission intervals, also the performance of the network can be improved considerably when applying the techniques described in this thesis. In particular, those strategies that a transparent for clients, for example, flow switching or off-line optimizations provide a useful opportunity to improve the network performance. While the services,

most of which are new in computer networks, are likewise implementable in other environments supporting advance reservations, e.g., cluster and grid management, the routing methods are unique in the network environment.

The application fields for advance reservations are mainly seen in the area of grid computing, although several approaches considered using this reservation type for content distribution networks, video conferencing, or even wireless scenarios. However, even in the wireless application scenarios, advance reservations are only implemented in a static manner. Environments with highly dynamic route or topology changes do not provide sufficiently reliable input to the advance reservation mechanisms discussed in this thesis, because these scenarios are difficult to predict or to anticipate but rather dependent on actual user behavior which is complicated to model. The same holds for wireless, mobile ad-hoc networks, where the dynamics of the environment in general contradict the idea of reliable planning which is the essence of advance reservations. However, the discussion of fault tolerance mechanisms shows, that also in a dynamically changing network a considerable gain can be achieved using the suitable recovery mechanisms. But this cannot be seen as the foundation for implementing advance reservations in ad-hoc scenarios, since failures occur far more often than in wireline networks. Furthermore, the failures only change the topology for a short period of time before the original status is recovered in contrast to ad-hoc scenarios, where a certain topology may only be valid for a short period of time and never occur again.

The convergence of network and application research towards a large-scale, global compute platform, the *Grid*, will also require for technologies that provide QoS guarantees for the behavior of the systems as a whole. QoS in this context describes the behavior of the entire system and requires co-allocations of various resource types, i.e., also computer networks. In this application environment, advance reservations already play an important role. Extending the scope of this reservation type to cover also computer networks is only a logical step and the developments described in this thesis can serve as one piece in the puzzle which will eventually constitute a large-scale, worldwide computational Grid with possibly billions of interconnected nodes [BdLH+03]. The results of this thesis have shown, that an advance reservation service with its various opportunities to enhance the functionality and performance of a network is affordable to implement with respect to the complexity of the advance reservation framework while giving operators the opportunity to also improve the performance and usability of the network. Thus, the advance reservation service as presented in this thesis can - among others - be conceived as one building blocking for the future Grid.

Appendix A

Network Topologies

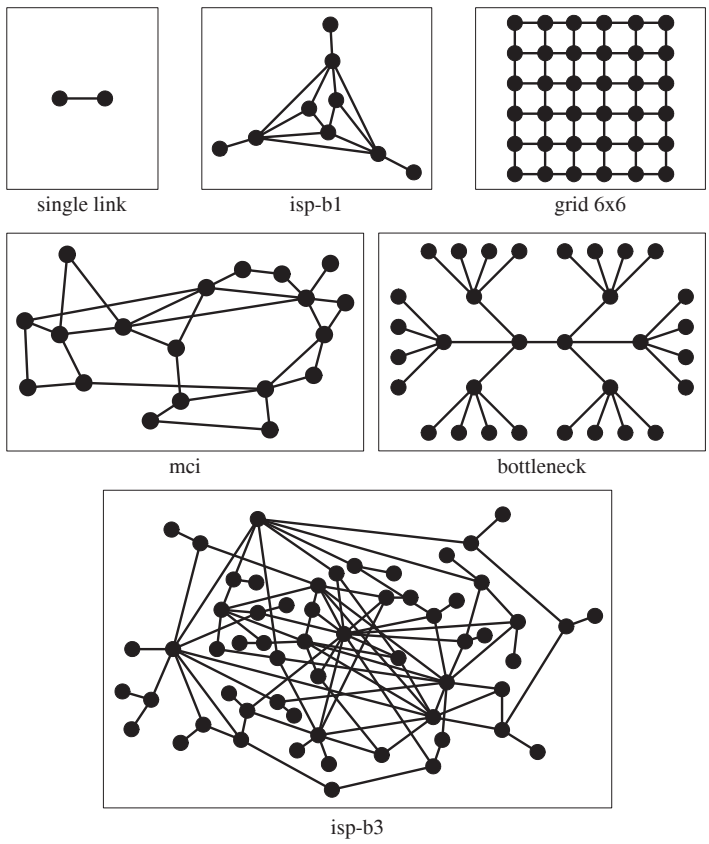


Figure A.1: The six additional network topologies used for the simulations.

Appendix B

Performance Analysis

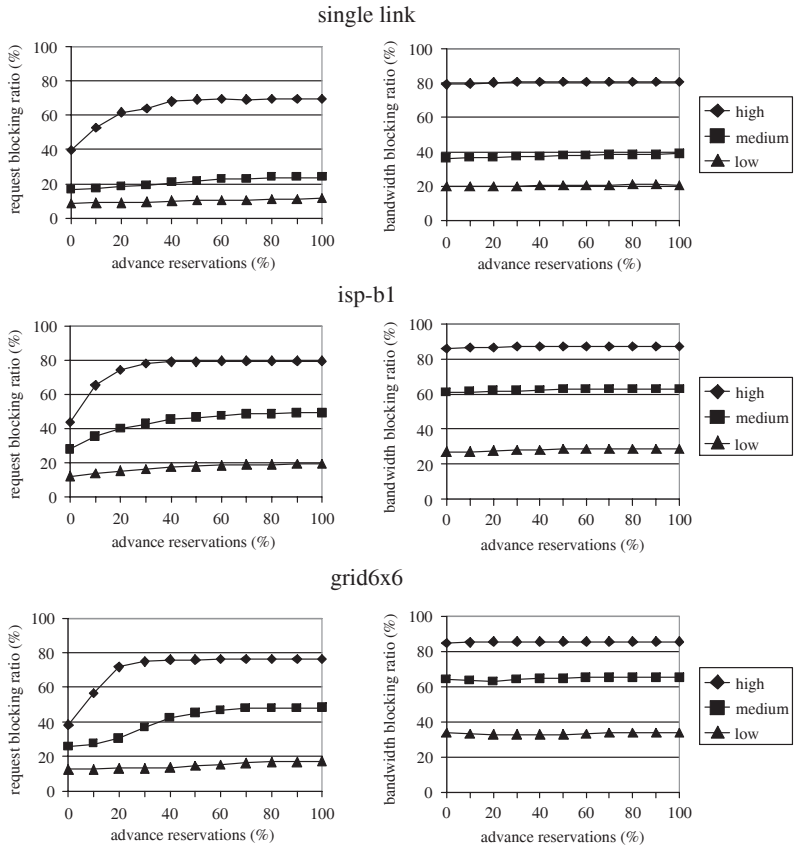


Figure B.1: RBR and BBR for different percentages of advance reservations under three different load conditions for the topologies *single link*, *isp-b1*, and *grid6x6*.

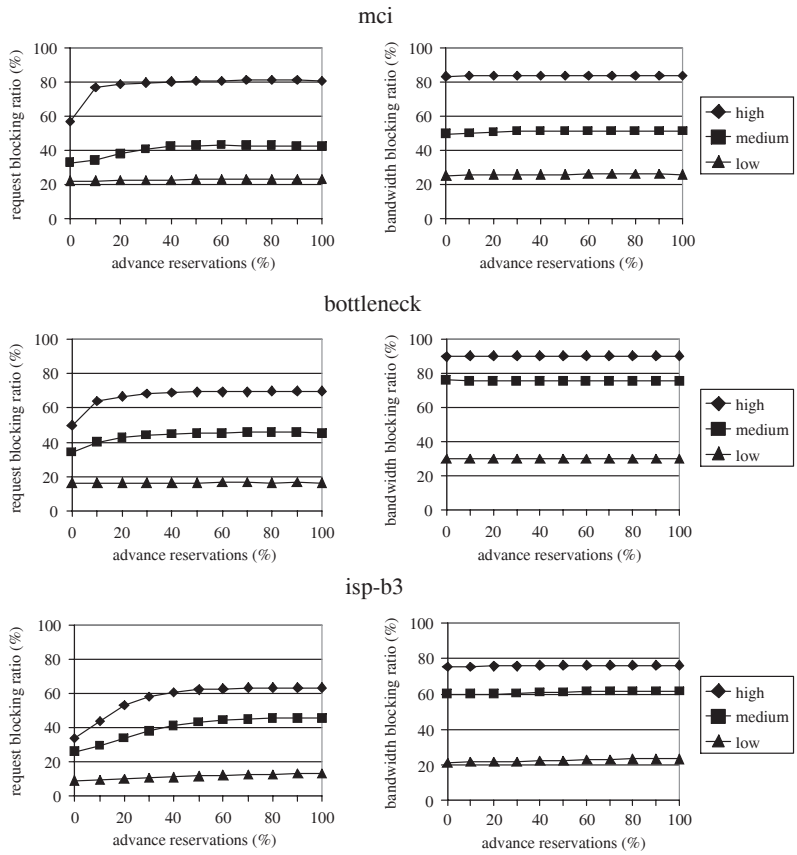


Figure B.2: RBR and BBR for different percentages of advance reservations under three different load conditions for the topologies *mci*, *bottleneck*, and *isp-b3*.

Appendix C

Performance Optimizations

The results of the performance optimization mechanisms for the different topologies under low and high load are depicted in this section. For the sake of clarity, only the summary of the results is given.

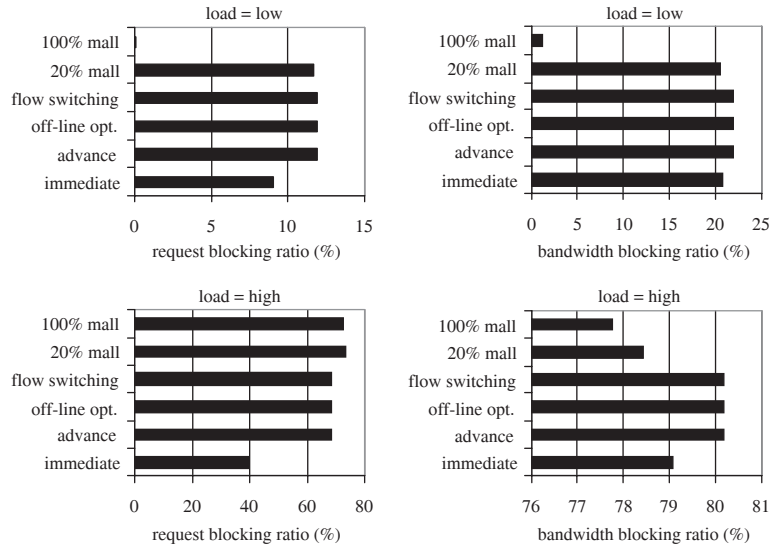


Figure C.1: Single Link: RBR and BBR

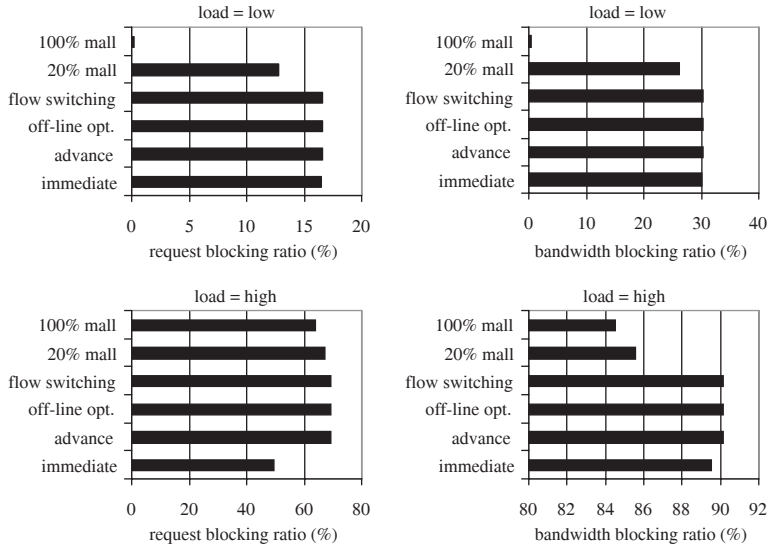


Figure C.2: BOTTLENECK: RBR and BBR

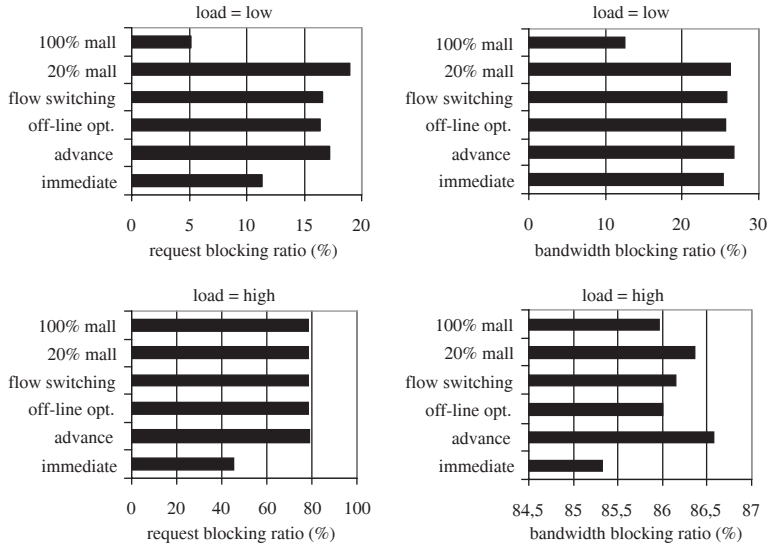


Figure C.3: ISP-B1: RBR and BBR

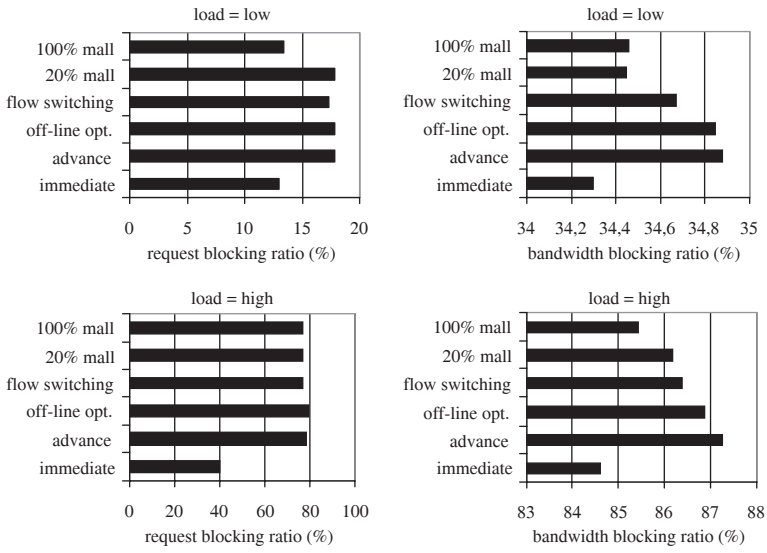


Figure C.4: Grid6x6: RBR and BBR

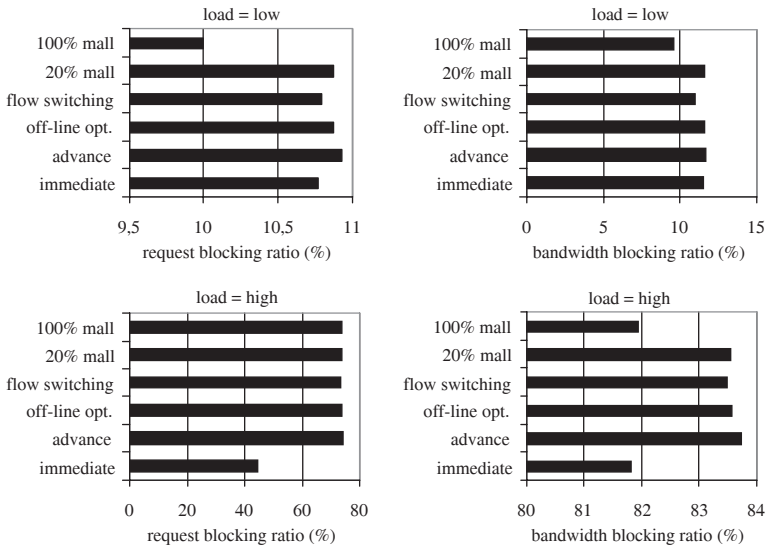


Figure C.5: MCI: RBR and BBR

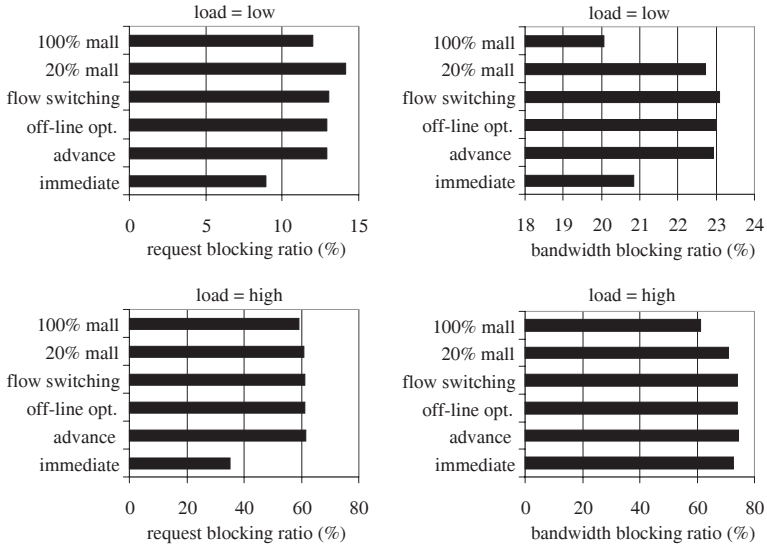


Figure C.6: ISP-B3: RBR and BBR

Bibliography

[AAA+02] Abrahamson, H., B. Ahlgren, J. Alonso, A. Andersson, and P. Kreuger. A Multi Path Routing Algorithm for IP Networks Based on Flow Optimization. In *Third COST 263 International Workshop on Quality of Future Internet Services (QofIS), Zurich, Switzerland*, volume 2511 of *Lecture Notes in Computer Science (LNCS)*, pages 135–144. Springer, 2002.

[AAP93] Awerbuch, B., Y. Azar, and S. Plotkin. Throughput Competitive On-line Routing. In *34th IEEE Symposium on Foundations of Computer Science (FOCS), Palo Alto, USA*, pages 32–40, 1993.

[ABG+01] Awduche, D.,, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. ftp://ftp.isi.edu/in-notes/rfc3209.txt, December 2001. RFC 3209.

[ABW04] Aiken, R., J. Boroumand, and S. Wolff. Network and Computing Research Infrastructure: Back to the Future. *Communications of the ACM*, 47(1):93–98, January 2004.

[ADF+01] Andersson, L., P. Doolan, N. Feldman, A. Fredette, and B. Thomas. LDP Specification. RFC 3036, January 2001.

[AK02] Authenrieth, A., and A. Kirstaedter. Engineering End-to-End IP Resilience Using Resilience-Differentiated QoS. *IEEE Communications Magazine*, 40(1):50–57, January 2002.

[AKW+99] Apostolopoulos, G., S. Kama, D. Williams, R. Guerin, A. Orda, and T. Przygienda. QoS Routing Mechanisms and OSPF Extensions. RFC 2676, August 1999.

[Bac] Bachelor, P. et al. Ultra High Capacity Optical Transmission Networks: Final Report of Action COST 239. http://web.cnlab.ch/cost239.

[BCD+98] Black, D., M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998.

[BDF03] Burchard, L.-O., and M. Droste-Franke. Fault Tolerance in Networks with an Advance Reservation Service. In *11th International Workshop on Quality of Service (IWQoS), Monterey, USA*, volume 2707 of *Lecture Notes in Computer Science (LNCS)*, pages 215–228. Springer, 2003.

[BdLH+03] Bal, H., C. de Laat, S. Haridi, K. Jeffery, J. Labarta, D. Laforenza, P. Maccallum, J. Masso, L. Matyska, T. Priol, A. Reinefeld, A. Reuter, M. Riguidel, D. Snelling, and M. van Steen. Next Generation Grid(s), European Grid Research 2005-2010. European Expert Group Report, European Union, Brussels, 2003.

[BFM+94] Banerjea, A., D. Ferrari, B.A. Mah, M. Moran, and D.C. Verma. The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences. Technical Report TR-94-059, Department of Computer Science, University of California at Berkeley, 1994.

[BH02] Burchard, L-O., and H.-U. Heiss. Performance Evaluation of Data Structures for Admission Control in Bandwidth Brokers. In *Intl. Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), San Diego, USA*, pages 652–659. Society for Modeling and Simulation International, 2002.

[BHK+04] Burchard, L.-O., M. Hovestadt, O. Kao, A. Keller, and B. Linnert. The Virtual Resource Manager: An Architecture for SLA-aware Resource Management. In *4th Intl. IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid), Chicago, USA, to appear*, 2004.

[BKL03] Bhatia, R., M. Kodialam, and T.V. Lakshman. Fast Network Reoptimization Schemes for MPLS and Optical Networks. In *11th International Workshop on Quality of Service (IWQoS), Monterey, USA*, volume 2707 of *Lecture Notes in Computer Science (LNCS)*, pages 249–256. Springer, 2003.

[BL00] Burchard, L.-O., and R. Lüling. An Architecture for a Scalable Video-on-Demand Server Network with Quality-of-Service Guarantees. In *5th Intl. Workshop on Distributed Multimedia Systems and Applications (IDMS), Enschede, The Netherlands*, volume 1905 of *Lecture Notes in Computer Science (LNCS)*, pages 132–143. Springer, 2000.

[BN02] Bhatnagar, S., and B. Nath. An Edge Router Based Protocol for Fault Tolerant Handling of Advance Reservations. In *IEEE Intl. Conference on Communications (ICC), New York, USA*, pages 1086 –1093, 2002.

[Bol94] V.A. Bolotin. Modeling call holding time distributions for CCS network design and performance analysis. *IEEE Journal on Selected Areas in Communications*, 12(3):433–438, April 1994.

[Bra97] Braden, R. (ed.). Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification. ftp://ftp.isi.edu/in-notes/rfc2205.txt, September 1997. RFC 2205.

[Bur03a] Burchard, L.-O. On the Performance of Computer Networks with Advance Reservation Mechanisms. In *11th IEEE International Conference on Networks (ICON), Sydney, Australia*, pages 449–454, 2003.

[Bur03b] Burchard, L.-O. Source Routing Algorithms for Advance Reservation Mechanisms. Technical Report 2003-3, Technische Universitaet Berlin, http://kbs.cs.tu-berlin.de/publications/res_mgnt/tr-2003-02.pdf, February 2003. ISSN 1436-9915.

[BZB+97] Braden, R., L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, September 1997.

[CCM+00] Chen, J., A. Caro, A. McAuley, S. Baba, Y. Ohba, and P. Ramanathan. A QoS Architecture for Future Wireless IP Networks. In *12th Intl. Conference on Parallel and Distributed Computing and Systems (PDCS), Las Vegas, USA*, November 2000.

[CMPS02] Case, J., R. Mundy, D. Partain, and B. Steward. Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, December 2002.

[CN98] Chen, S., and K. Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, 12(6):64–79, 1998.

[CSD+01] Chan, K., J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. COPS Usage for Policy Provisioning (COPS-PR). RFC 3084, March 2001.

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

[DCB+02] Davie, B., A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246, March 2002.

[DdLM+03] DeFanti, T., C. de Laat, J. Mambretti, K. Neggers, and B. St. Arnaud. TransLight: A Global-Scale LambdaGrid for E-Science. *Communications of the ACM*, 46(11):34–41, November 2003.

[DKPS95] Degermark, M., T. Köhler, S. Pink, and O. Schelen. Advance Reservations for Predictive Service. In *5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Durham, USA*, volume 1018 of *Lecture Notes in Computer Science (LNCS)*, pages 3–15, 1995.

[DR00] Davie, B., and Y. Rekhter. *MPLS - Technology and Applications.* Morgan Kaufman Publishers, 2000.

[dSAU02] de Oliveira, J.C., C. Scoglio, I.F. Akyildiz, and G. Uhl. A New Preemption Policy for DiffServ-Aware Traffic Engineering to Minimize Rerouting. In *IEEE INFOCOM, New York, USA*, pages 695–704, 2002.

[Dur00] Durham, D. (ed.). The COPS (Common Open Policy Service) Protocol. RFC 2748, January 2000.

[Epp98] Eppstein, D. Finding the k Shortest Paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[Erl02] Erlebach, T. Call Admission Control for Advance Reservation Requests with Alternatives. In *3rd Workshop on Approximation and Randomization Algorithms in Communication Networks, Rome, Italy*, pages 51–64, 2002.

[FGV95] Ferrari, D., A. Gupta, and G. Ventre. Distributed Advance Reservation of Real-Time Connections. In *5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Durham, USA*, volume 1018 of *Lecture Notes in Computer Science (LNCS)*, pages 16–27. Springer, 1995.

[FK99a] Foster, I., and C. Kesselman. Globus: A Toolkit-based Grid Architecture. *In [FK99b]*, pages 259–278, 1999.

[FK99b] Foster, I., and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufman Publishers, 1999.

[FKL+99] Foster, I., C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *7th International Workshop on Quality of Service (IWQoS), London, UK*, pages 27–36, 1999.

[FRS00] Foster, I., A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In *8th International Workshop on Quality of Service (IWQoS), Pittsburgh, USA*, pages 181–188, 2000.

[GB99] Günter, M., and T. Braun. Evaluation of Bandwidth Broker Signaling. In *IEEE 7th Intl. Conference on Network Protocols (ICNP), Toronto, Canada*, pages 145–152, October 1999.

[GG92] Garay, J. A., and I. S. Gopal. Call Preemption in Communication Networks. In *IEEE INFOCOM, Edinburgh, Scotland, UK*, pages 1043–1050, 1992.

[GHMN95] Gupta, A., W. Howe, M. Moran, and Q. Nguyen. Resource Sharing for Multi-Party Real-Time Communication. In *IEEE INFOCOM*, pages 1230–1237, 1995.

[GLO] The Globus Project. http://www.globus.org/.

[GO00] Guerin, R., and A. Orda. Networks with Advance Reservations: The Routing Perspective. In *IEEE INFOCOM, Tel Aviv, Israel*, pages 118–127, 2000.

[Gro02] D. Grossman. New Terminology and Clarifications for Diffserv. RFC 3260, April 2002.

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

[HBWW99] Heinanen, J., F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, June 1999.

[ICM+02] Iannaccone, G., C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of Link Failures in an IP Backbone. In *2nd ACM SIGCOMM Internet Measurement Workshop 2002, Marseille, France*, pages 237–242, Nov 2002.

[Jam02] Jamoussi, B., (ed.). Constraint-Based LSP Setup using LDP. ftp://ftp.isi.edu/in-notes/rfc3212.txt, January 2002. RFC 3212.

[Jan02] Jansen, K. Scheduling Malleable Parallel Tasks: An Asymptotic Fully Polynomial-Time Approximation Scheme. In *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy*, pages 562–573, 2002.

[Jun99] Jungnickel, D. *Graphs, Networks and Algorithms*. Springer, 1999.

[Kar02] Karakostas, G. Faster approximation schemes for fractional multicommodity flow problems. In *13th ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, USA*, pages 166–173, 2002.

[KB00] Khalil, I., and T. Braun. Implementation of a Bandwidth Broker for Dynamic End-to-End Resource Reservation in Outsourced Virtual Private Networks. In *25th IEEE Conference on Local Computer Networks (LCN), Tampa, USA*, pages 160–174, 2000.

[KBWS99] Karsten, M., N. Berier, L. Wolf, and R. Steinmetz. A Policy-Based Service Specification for Resource Reservation in Advance. In *Intl. Conference on Computer Communications (ICCC), Tokyo, Japan*, pages 82–88, September 1999.

[KR01] Keller, A., and A. Reinefeld. Anatomy of a Resource Management System for HPC Clusters. In *Annual Review of Scalable Computing, vol. 3, Singapore University Press*, pages 1–31, 2001.

[LG01] Lee, S., and M. Gerla. Fault-Tolerance and Load Balancing in QoS Provisioning with Multiple MPLS Paths. In *9th International Workshop on Quality of Service (IWQoS), Karlsruhe, Germany*, volume 2092 of *Lecture Notes in Computer Science (LNCS)*, pages 155–169. Springer, 2001.

[LNO02] Lewin-Eytan, L., J. Naor, and A. Orda. Routing and Admission Control in Networks with Advance Reservations. In *5th Intl. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, volume 2462 of *Lecture Notes in Computer Science (LNCS)*, pages 215–228. Springer, 2002.

[LT94] Ludwig, W., and P. Tiwari. Scheduling Malleable and Nonmalleable Parallel Tasks. In *Fifth ACM/SIAM Symposium on Discrete Algorithms (SODA), Arlington, USA*, pages 167–176, 1994.

[MA02] Moon, B., and H. Aghvami. Reliable RSVP Path Reservation for Multimedia Communications under an IP Micromobility Scenario. *IEEE Wireless Communications*, 9(5):93–99, October 2002.

[MCRW96] McCloghrie, K., J. Case, M. Rose, and S. Waldbusser. Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1905, January 1996.

[MLMB01] Medina, A., A. Lakhina, I. Matta, and J. Byer. BRITE: An Approach to Universal Topology Generation. In *9th IEEE Intl. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Cincinnati, USA*, pages 346–356, 2001.

[Moy98] Moy, J. OSPF Version 2. RFC 2328, April 1998.

[MS97a] Ma, Q. and P. Steenkiste. On Path Selection for Traffic with Bandwidth Guarantees. In *5th IEEE International Conference on Network Protocols (ICNP), Atlanta, USA*, pages 191–204, 1997.

[MS97b] Ma, Q. and P. Steenkiste. Quality-of-Service Routing for Traffic with Performance Guarantees. In *5th International Workshop on Quality of Service (IWQoS), New York, USA*, pages 115–126, 1997.

[MSZ96] Ma, Q., P. Steenkiste, and H. Zhang. Routing High-Bandwidth Traffic in Max-Min Fair Share Networks. In *SIGCOMM*, pages 206–217. ACM, 1996.

[NBBB98] Nichols, K., S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, December 1998.

[NJZ99] Nichols, K., V. Jacobson, and L. Zhang. A Two-Bit Differentiated Services Architecture for the Internet. RFC 2638, July 1999.

[NT01] Norden, S., and J. Turner. DRES: Network Resource Management using Deferred Reservations. In *IEEE Globecom, San Antonio, USA.*, pages 2299–2303, November 2001.

[OS00] Orda, A., and A. Sprintson. QoS Routing: The Precomputation Perspective. In *IEEE INFOCOM, Tel Aviv, Israel*, 2000.

[PD00] Peterson, L., and B. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufman Publishers, 2nd edition, 2000.

[PF95] Paxson, V., and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[PS00] Pan, P., and H. Schulzrinne. Lightweight Resource Reservation Signaling: Design, Performance and Implementation. Technical Report CUCS-019-00, Department of Computer Science, Columbia University, New York, USA, 2000.

[Rei95a] Reinhardt, W. Advance Resource Reservation and its Impact on Reservation Protocols. In *Broadband Islands, Dublin, Ireland*, September 1995.

[Rei95b] Reinhardt, W. Advance Resource Reservations for Multimedia Applications. In *2nd Intl. Workshop on Advanced Teleservices and High Speed Communication Architectures (IWACA), Heidelberg, Germany*, pages 23–34, 1995.

[RG02] Riedl, A., and J. Glasmann. On the Design of Resource Management Domains. In *Intl. Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), San Diego, USA*, pages 744–748. Society for Modeling and Simulation International, 2002.

[RR01] Rekhter, Y., and E. Rosen. Carrying Label Information in BGP-4. RFC 3107, May 2001.

[RVC01] Rosen, E., A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. ftp://ftp.isi.edu/in-notes/rfc3031.txt, January 2001. RFC 3031.

[San03] Sander, V. *Design and Evaluation of a Bandwidth Broker that Provides Network Quality of Service for Grid Applications.* PhD thesis, John von Neumann Institute for Computing, Jülich, Germany, February 2003. ISBN 3-00-010002-4.

[SBK98] Schill, A., F. Breiter, and S. Kühn. Design and Evaluation of an Advance Reservation Protocol on Top of RSVP. In *4th International Conference on Broadband Communications, Montreal, Canada*, pages 430–442, 1998.

[SH03] Sharma, V., and F. Hellstrand. Framework for Multi-Protocol Label Switching (MPLS)-based Recovery. RFC 3469, February 2003.

[SKB97] Schill, A., S. Kühn, and F. Breiter. Resource Reservation in Advance in Heterogenous Networks with Partial ATM Infrastructures. In *IEEE INFOCOM, Kobe, Japan*, pages 611–618, April 1997.

[SM90] Shahroki, F., and D. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318–334, 1990.

[SNNP99] Schelen, O., A. Nilsson, J. Norrgard, and S. Pink. Performance of QoS Agents for Provisioning Network Resources. In *7th International Workshop on Quality of Service (IWQoS), London, UK*, pages 17–26, 1999.

[SP97] Schelen, O., and S. Pink. An Agent-based Architecture for Advance Reservations. In *22nd IEEE Conference on Local Computer Networks (LCN), Minneapolis, USA*, pages 451–459, 1997.

[SP98a] Schelen, O., and S. Pink. Aggregating Resource Reservations over Multiple Routing Domains. In *6th International Workshop on Quality of Service (IWQoS), Napa, California, USA.*, pages 29–32, 1998.

[SP98b] Schelen, O., and S. Pink. Resource Reservation Agents in the Internet. In *8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98), Cambridge, UK*, 1998.

[SPG97] Shenker, S., C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service. RFC 2212, September 1997.

[TBA01] Talukdar, A., B. Badrinath, and A. Acharya. MRSVP: A Resource Reservation Protocol for an Integrated Services Network with Mobile Hosts. *IEEE Wireless Communications*, 7(1):5–19, 2001.

[Tea01] QBone Signaling Design Team. QBone Signaling Design Team, Final Report. http://qos.internet2.edu/wg/documents-informational/20020709-chimento-etal-qbone-signaling/, 2001.

[Uni] UNICORE Forum e.V.: UNICOREpro. http://www.unicore.org.

[VD03] Vadhiyar, S., and J. Dongarra. SRS: A Framework for Developing Malleable and Migratable Parallel Applications for Distributed Systems. *Parallel Processing Letters*, 13(2):291–314, June 2003.

[WC96] Wang, Z., and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal of Selected Areas in Communications*, 16(7):1228–1234, September 1996.

[WDSS95] Wolf, L. C., L. Delgrossi, R. Steinmetz, and S. Schaller. Issues of Reserving Resources in Advance. In *5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Durham, USA*, volume 1018 of *Lecture Notes in Computer Science (LNCS)*, pages 28–39. Springer, 1995.

[WG98] Wischik, D., and A. Greenberg. Admission Control for Booking Ahead Shared Resources. In *IEEE INFOCOM, San Francisco, USA*, pages 873–882, 1998.

[WN02] Wan, J. and K. Nahrstedt. Hop-by-Hop Routing Algorithms For Premium-class Traffic in DiffServ Networks. In *IEEE INFOCOM, New York, USA*, pages 705–714, 2002.

[Wro97a] Wroclawski, J. Specification of the Controlled-Load Network Element Service. RFC 2211, September 1997.

[Wro97b] Wroclawski, J. The Use of RSVP with IETF Integrated Services. RFC 2210, September 1997.

[WS97] Wolf, L. C. and R. Steinmetz. Concepts for Resource Reservation in Advance. *Kluwer Journal on Multimedia Tools and Applications*, 4(3):255–278, 1997.

[XN99] Xiao, X., and L. M. Ni. Internet QoS: A Big Picture. *IEEE Network*, 13(2):8–18, March 1999.

[ZDE+93] Zhang, L., S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource Reservation Protocol. *IEEE Communications Magazine*, 9(31):8–18, 1993.

[ZOS00] Zhao, W., D. Olshefski, and H. Schulzrinne. Internet Quality of Service: an Overview. Technical Report CUCS-003-00, Columbia University, New York, February 2000.