Service-Tailored QoS Management in High Performance Networks

Roland Bless, Matthias Jacob, Claudia Schmidt
Institute of Telematics, University of Karlsruhe
Zirkel 2, 76128 Karlsruhe, F.R. of Germany,
Tel.: +49 721 608-{6402,6408}, Fax: +49 721 388097,
{bless,jacob,schmidt}@telematik.informatik.uni-karlsruhe.de

Abstract

Service integrated communication systems need to serve concurrently a variety of applications, probably each of them with different service requirements. Therefore, an integrated Quality-of-Service (QoS) management is needed. Within this paper, a flexible approach of QoS management in high performance networks is presented. QoS management functions can be configured in order to build a service-tailored management. Moreover, performance measurements of the presented management functions are discussed.

1 INTRODUCTION

Service integrated communication systems need to provide concurrent support for an increasing variety of applications. Typically, such applications communicate simultaneously via several data streams, including audio, video, and conventional data traffic. Each stream is characterized by highly different service requirements commonly expressed in terms of so-called quality of service parameters (QoS parameters). Most importantly, these applications require QoS guarantees from *application-to-application*. Thus, an *integrated QoS management* is necessary, covering all management functions in end and intermediate systems.

As an integral part of a communication system, certain mechanisms (e.g., scheduling, protocol functions) are needed to support service requirements. Typically, basic QoS guarantees can be given for resources by an adequate *resource management* (Nahrstedt & Steinmetz 1995). QoS guarantees provided by a resource management solely are not sufficient for applications. Especially, for coordination and enhancement of basic guarantees from resource managers, *integrated* QoS management functions are highly needed (Hutchison et al. 1994). Integrated QoS management is concerned with an enhanced application-oriented QoS specification, QoS mapping, QoS negotia-

©IFIP 1997. Proceedings of the 5th IFIP International Workshop on Quality of Service (IWQOS'97): *Building QoS into Distributed Systems*, edited by A. Campbell and K. Nahrstedt, published by Chapman & Hall, ISBN 0–412–80940–0.

tion, QoS maintenance and QoS control (configuration and monitoring). Thus, *QoS management architectures* (Campbell et al. 1996) for an integrated QoS management must support a QoS-based API at which applications are capable to specify their service requirements in an application-oriented syntax and semantics.

Several aspects have to be considered while designing and realizing an integrated QoS architecture and its management functions. First, QoS management functions running complex tasks needed by a certain service specification must be able to keep up with the high data rates of networks, such as ATM. Second, management functions, such as QoS negotiation and QoS monitoring, have different time requirements (Campbell et al. 1996) and thus, have to be separated according to their time domains. It is crucial, that management functions with stringent time bounds are capable to operate autonomously and in parallel to functions with less severe time requirements. Finally, applications demand very different service support. Therefore, besides configuration of service supporting mechanisms, configuration of management functions based on the service specification plays an important role. Only a service-tailored QoS management is able to meet the varying demands of forthcoming applications.

This paper presents an approach towards a highly flexible and service-tailored QoS architecture for high performance networks. In section 2, the COSIMA architecture for integrated QoS management is introduced in some detail. Section 3 presents the QoS-Manager, a central component of the COSIMA architecture. First, tasks of the QoS-Manager and its architecture are described. In a next step, the implementation and performance results are presented. Afterwards, in section 4 another component of the architecture, a flexible QoS Monitor is introduced. Flexibility of the QoS Monitor and performance measurements are discussed. Finally, the paper is concluded by a summary and a description of future work.

2 COSIMA - A QOS MANAGEMENT ARCHITECTURE

The *COSIMA-Architecture* (COmmunication Systems with Integrated services MAnagement) represents a QoS management architecture that strictly separates resource and integrated QoS management functions. The data path is represented by a sequence of resources, each of them managed by a resource manager. QoS management functions are used to coordinate and enhance the underlying resource management in order to provide an integrated management requested by applications. All management functions are stringently separated from data transfer and are located in the management architecture as shown in Figure 1. However, this architecture is not derived from hierarchical layered communication systems, but from the functionality of management tasks.

The overall behavior of the management system is presented to applications at a high-level application interface, where applications are capable to specify

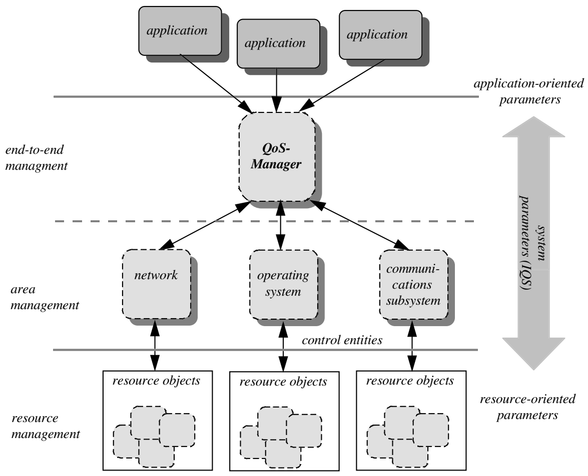


Figure 1 COSIMA: QoS Management Architecture

their communication needs in an application-oriented syntax and semantics. They communicate with a so-called *QoS-Manager* that coordinates all operations of the underlying management. At the COSIMA Application Programming Interface (API), applications are able to request and modify communication services for *sessions* and *data streams* (Frick & Schmidt 1996). A data stream is a simplex point-to-point or multi-point connection between applications associated with a certain quality-of-service. Several data streams between two or more distributed applications and *relations* among these streams are grouped into a session.

Communication resources are grouped into three areas and are managed by one *control entity* in each area. COSIMA comprises control entities for *network*, *operating system*, and *communications subsystem* (Schmidt & Zitterbart 1995). Each control entity consists of a *control agent* and a *QoS monitor*. The control agent is responsible for configuration and reconfiguration of resources and resource managers and the QoS monitor continuously monitors the achieved QoS. Management-related information of each resource manager and managed resources is presented to the corresponding control entity by so-called *resource objects*. Examples of such information are resource capacity, control mechanisms for resource management (e.g., scheduling algorithms) and information about data streams that currently use the resource. All communication between control entities and resource managers is done through resource objects, and thus a clearly defined service interface exists. Since resources as well as resource managers are operating in the data path, they are very time critical. Therefore, based on resource objects they are clearly decoupled from control operations performed by control agent and QoS monitor. However, control functions must be able to keep up with high data transfer rates and therefore, both components of a control entity are designed for high performance environments.

In order to supply integrated services, the COSIMA-Architecture needs an entity that coordinates the cooperation of its autonomous area management components. This entity is the *QoS-Manager*. As a central integrating component of the communication system the QoS-Manager has to manage exchange of service control information with all applications that use integrated communication services in the local end-system. To provide integrated communication services from application-to-application it distributes QoS management information among local components and other QoS-Managers at remote end-systems.

3 THE QOS-MANAGER

Generally, tasks of the QoS-Manager can be coarsely divided into following functional areas:

- Communication with applications using an application-oriented service interface.
- Distribution of information about the session (locally as well as to remote end-systems). This especially includes negotiation and renegotiation of QoS.
- Adaptation of information between different levels of service specification (i.e. QoS-Mapping).

The QoS-Manager supplies access to integrated services of the communication system by offering an application-oriented service interface. Thus, applications express their communication requirements in an application-oriented syntax and semantics by using so-called *data stream objects*. A list of data stream objects together with a list of *relation objects* constitute a *session object*. A relation object describes a *logical* or *temporal* relationship between a pair of data streams. Examples of logical relationships are lifetime-relation (existence of one data stream is required by another), identical participants or identical QoS. The temporal relation specifies synchronization conditions between data stream by use of a *skew-interval* (Blakowski & Steinmetz 1996). In order to provide a very flexible but still easy-to-use service interface, applications can select among predefined session and data stream types, individually composed sessions or even single data stream specifications. Applications control a certain session by invoking service primitives as methods on the associated session object (e.g., `OpenReq, ModifyAddRsp`). There exist service primitives for opening a session, changing a session by modifying the characteristics of data streams, deleting or adding data streams, joining or leaving a session, notifications and closing a session. By use of *management parameters* for each data stream, such as priority, report policy, action policy, and cost, applications are able to direct management behavior in their intended way. The degree of autonomous QoS management actions is determined by the action policy. Additionally, a report policy determines the amount of information flowing from QoS-Manager to application. Hence, each application can choose its preferred management behavior. Because the QoS-Manager is only concerned with management and control of communication services it is not involved in actual data transfer. Thus, its operation is not really time critical, but it must be able to process several requests from different applications simultaneously. This is necessary because there exists only one QoS-Manager entity in an end-system, and communication with one application should not suspend other applications from using integrated communication services. Consequently, the *QoS-Manager* uses asynchronous message exchange for communication with applications and holds state information for each application.

During establishment of a session its properties are negotiated in terms of QoS parameters among all participants. In the course of an active session dynamical changes can occur such as renegotiation of a data stream's QoS parameters, joining or leaving participants and addition or removal of data streams. Additionally, monitoring information or reports about QoS violations can be exchanged. In the COSIMA-Architecture, end-to-end communication between QoS-Managers of session participants is accomplished by use of a *QoS Management Protocol (QMP)*, whereas local communication between QoS-Manager and system components is determined by a *Local Communication Protocol (LCP)*. The latter is necessary, because control entities of the areas *network*, *operating system* and *communication subsystem* are involved in the negotiation process. Furthermore, for negotiation of resources in intermediate and end-systems at network level a common network reservation protocol such as RSVP (Zhang et al. 1993) or ST2+ (IETF 1995) is used.

On one hand the QoS-Manager uses application-oriented QoS parameters at the service interface (e.g., frame size, frame rate and color depth of a video data stream), but on the other hand it communicates with control entities in system-oriented QoS parameters (e.g., throughput, delay, loss rate, etc.). Both representations of a data stream must be mapped onto each other (system-oriented parameters cannot be always mapped uniquely onto application parameters). To reduce the need for further mapping operations, all components of the COSIMA-Architecture operate only on application-oriented and system-oriented parameters.

3.1 QoS-Manager architecture

The architecture of the QoS-Manager depicted in Figure 2 is derived from the required functionality described above. Therefore, the QoS-Manager is divided internally into three parts reflecting its functional separation: The *coordinator* part controls the sequence of events within the QoS-Manager and handles communication with applications. It manages all session-related information and operates on subsets of the session's data streams or the whole session. All negotiation tasks and other information exchange with local components as well as remote systems are managed by the *communicator* part. The Communicator negotiates complete sessions (including several data streams) at once and is even capable of managing many different negotiations in parallel. Processing different primitives on several data streams for one session and various sessions simultaneously is possible, too. This leads to faster response times because waiting for responses belonging to one negotiation does not suspend the course of other negotiations. Coordinator and Communicator operate extensively independent and mutually exchange messages to interchange information, i.e. they are active entities (specified as enhanced finite state machines). Additionally, the QoS-Manager contains a library of QoS mapping functions (a so-called *QoS-Mapper*) in order to map application-oriented QoS parameters onto system-oriented QoS parameters and vice versa. In contrast to Communicator and Coordinator the QoS-Mapper is a passive entity. Furthermore, QoS specifications of data streams and other management related data is stored in a database called Internal QoS Structure (IQS).

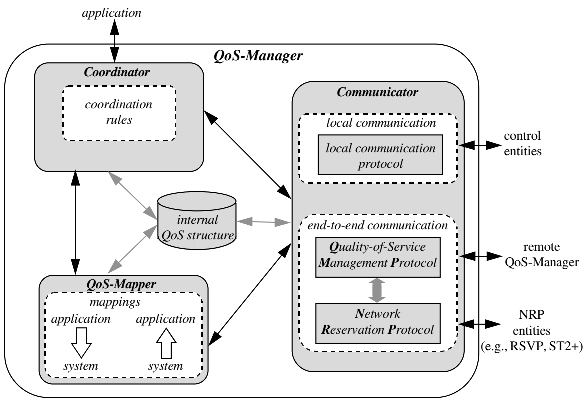


Figure 2 The QoS-Manager

3.2 QoS Negotiation

As mentioned above, a specific protocol was defined for end-to-end communication: the *QoS Management Protocol* (QMP). It was designed from the beginning for efficient support of arbitrary group communication scenarios. By distinguishing only the roles of *initiator* and *responder* of negotiation or management requests it is independent of the directions of data flows. This concept similar to the QoS-Broker (Nahrstedt & Smith 1995) yields more flexibility and opens up the complete multitude of application scenarios. Additionally, it is possible to negotiate several data streams at once, thus reducing necessary negotiation steps and session setup delay. By negotiating several streams simultaneously end-to-end delay has to be considered only once, whereas other negotiation protocols execute end-to-end negotiation for each stream separately. The latter results in an n-times higher end-to-end delay for n streams and insufficient support for group communication scenarios. Moreover, if streams are negotiated singly and the last stream in such a negotiation process becomes rejected all previous negotiation steps would have been of no avail. Furthermore, sender- or receiver-oriented reservation protocols can be used to request resources at network level if those mechanisms are not already embedded into the network itself.

To outline the features of QMP a brief description of a QoS negotiation process is given. It is needed for session opening, when a new participant is going to join a current session, and when data streams are added or their parameters are changed. The negotiation process can be executed for all data streams simultaneously and is coarsely divided into four steps: at first the initiator carries out negotiation with its local components (operating system, communication subsystem, and network). In the second step an end-to-end negotiation with the involved remote end-systems follows, that perform a local negotiation on their part and inform the initiator about the results. The latter tries to find a consensus out of the different responses and sends it back to the responders. The third step consists of resource reservation within the network by using a network reservation protocol. Corresponding to the selected reservation protocol each end-system requests resources for either incoming or outgoing data streams and awaits results from this reservation procedure. Afterwards, every system has got the same view of committed network resources. Finally, the application-oriented parameters have to be adapted according to negotiated network parameters in the fourth step using a 'backward' mapping function for QoS parameters (Schmidt 1997). This step is completed by releasing over-allocated resources ('relaxing') locally and within the network. The latter follows from the fact that most application-oriented parameters cannot be continuously mapped onto network parameters, so formerly negotiated network parameters have to be possibly rounded-off again. Subsequently, the application is notified about the negotiated QoS. This particular sequence of negotiation steps was chosen because no unnecessary reservation of network resources is done when local resources are short or essential application parameters are not supported by the other peers.

3.3 Implementation and Performance

The QoS-Manager was implemented in C++ on Alpha-Workstations under Digital Unix 3.2 which is based on a Mach kernel. By using an object-oriented language it is easily possible to enhance the existing class hierarchy of streams. Common attributes of all streams form an abstract base class from which specialized stream classes (e.g., audio, video) are derived. The QoS-Manager is an autonomously running entity (process) that communicates with all applications using integrated communication services. Inside the QoS-Manager, Coordinator and Communicator are realized as separate threads which use Mach facilities for interchanging messages. By using threads a high possible degree of parallel processing at a small overhead is reached. This results in shorter response times during multiple simultaneous negotiations.

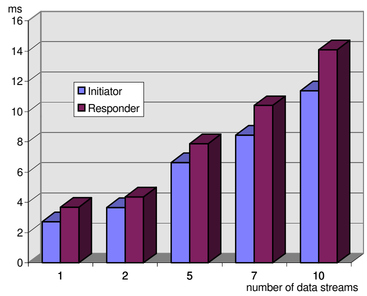


Figure 3 Internal processing times of QoS-Manager for negotiation

Performance of the QoS-Manager was measured by using the `thread_info()` Mach-Kernel call (Boykin et al. 1993). It returns among other things the time a particular thread spent in user space. Results for a QoS negotiation process are depicted in Figure 3. The measured times do not include activities of other components like control entities or network resource reservation entities. Measurements were executed by opening sessions consisting of 1, 2, 5, 7 and 10 data streams. Every scenario contained 200 measurements. The highest value (14.1 ms) was measured at the responder side for a session consisting of 10 data streams. Conditioned by the increasing amount of specification data, negotiation time grows linear with number of data streams. But, in spite of the additional complexity it is *only* growing linear, not more. However, a complete QoS negotiation in one end-system will last some milliseconds. Naturally, the required end-to-end delay as well as response times of all involved entities have to be added in order to complete one negotiation. But one can expect that those response times are considerably smaller than actual end-to-end delays.

4 QOS MONITORING

In this section another QoS management function located closer to the data path is addressed. QoS monitoring continuously determines QoS parameters from protocol information. Additionally, QoS violations are detected by a comparison of measured parameters with parameters negotiated in the traffic contract. A flexible and service-tailored QoS monitor has to monitor efficiently service requirements of a variety of forthcoming applications and, furthermore, it must be able to work besides different communications protocols on transport and network level. Therefore, a configurable QoS monitor is needed whereas high performance communications require a small monitor with little overhead.

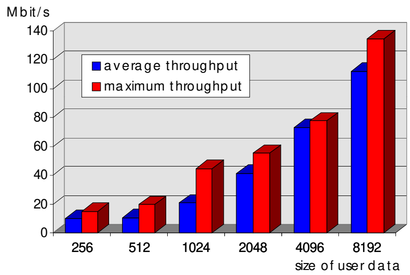
Based on the architecture and specification in (Schmidt & Bless 1996) the presented QoS Monitor entity was extended with configurable interfaces. These comprehend two different components: *initial protocol-dependent QoS configuration*, and, *flexible and dynamic configuration per connection*. The initial protocol-dependent QoS configuration adapts the QoS Monitor to QoS parameters supplied by different communication protocols. Controlled by the accompanying control agent (see section 2) the flexible and dynamic configuration per connection allows to vary the set of parameters to be monitored and to change the report behavior of the QoS Monitor dynamically during operation. Implementing these design issues means to be careful with performance behavior of the monitor because monitoring of real-time systems has got very sensitive time constraints. Therefore, the following performance evaluation is an essential matter of this section.

4.1 Performance Evaluation

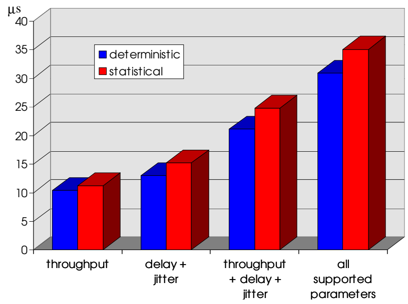
For evaluation of the achievable performance a test network consisting of two Alpha workstations (AXP 3000/500) and a DEC GIGAswitch was used. The protocol stack is based on a test protocol over UDP/IP and AAL5 over a 155 Mbit/s ATM link. The test protocol adds a small header with fields for packet type, time-stamp, and sequence number to user data. Considering protocol overhead, maximum achievable throughput within the test protocol is about 135 Mbit/s (Schmidt 1997).

Examining the performance issues consists of two parts: first, influences of high data rates on the QoS Monitor, and second, overhead entailed by the QoS Monitor. The first topic addresses performance of the QoS Monitor in high performance networks. In these environments, the QoS Monitor must be able to keep up with communication speed without any performance influence on the protocol. Throughput was measured for a varying size from 256 to 8192 Bytes of user data units (cf. Figure 4(a)). For each size, average (over 100 events) and maximum throughput were determined in 1000 measurements. While the measured maximum throughput for small data units was only about 15 Mbit/s, large data units yielded throughput values over 134 Mbit/s. Thus, transmitting large data units leads to throughput values near to the theoretical achievable throughput. In contrast, throughput reduction for small data units is based on the high amount of packet processing time compared to the small amount of data. But generally, the QoS Monitor was able to keep up with all data rates and to determine all QoS parameters in time.

Measuring monitor overhead was the next step. Again, 1000 measurements were performed for each configuration. Statistical monitoring is based on an



(a) Receiver throughput for various user data packet sizes



(b) Execution times: deterministic versus statistical monitoring

Figure 4 QoS Monitor performance

interval of 100 measurement points in which 10 QoS violations are allowed. To achieve a comparable value for variations of different configurations two distinct test configurations have been applied: monitoring of different parameter combinations, and, comparison between deterministic and statistical monitoring. Because the QoS Monitor is implemented by using threads (Schmidt & Bless 1996), too, the `thread_info()` Mach-Kernel call was applied here also (cf. section 3.3). Figure 4(b) shows a comparison of the monitor overhead for different parameter configurations. Generally, determination of one QoS parameter and a test for QoS violation needs about $10\mu s$. Therein, a basic overhead of around $4\mu s$ is produced by internal copy and loop operations within each monitoring step. The comparison between deterministic and statistical monitoring shows an additional overhead of 2-$3\mu s$ per parameter for statistical monitoring.

Comparing these results with performance measurements of communication protocols in user space (Braun & Diot 1996) shows a monitor overhead lower than 10% of the protocol overhead. But, while protocol processing time grows with the size of data units, the measured monitoring time is totally independent of data unit sizes.

The presented performance results reflect directly the flexibility of the QoS Monitor. It can be configured in order to monitor one or more selected QoS parameters for a connection. Additionally, the flexible protocol interface allows monitoring in changing environments. These characteristics and the results of the performance evaluation show that the developed QoS monitor is flexible and can be tailored to the requested service while operating under high performance conditions.

Comparing performance results of QoS-Manager and QoS Monitor shows, that both components operate on different time scales. Typical operating times of the QoS Monitor are located in the range of μs while the QoS-Manager operates in ms. These operating times confirm directly the design decisions of the COSIMA architecture. It is crucial that management components operating on different time scales are located in autonomous entities. Thus, time critical components, such as the QoS Monitor are able to work autonomously and are not influenced by complex management tasks of other components.

5 SUMMARY AND FUTURE WORK

Generally, integrated QoS management forms an increasingly important component of service integrated communication systems. The presented QoS management architecture COSIMA provides integrated services from application-to-application. Derived from the functionality of management tasks it consists of autonomous entities that are coordinated by a QoS-Manager. The latter supplies an application-oriented service interface to applications and configures the functionality of system components appropriately. Furthermore, it uses flexible and comprehensive mechanisms for distributing QoS management information, including QoS negotiation and renegotiation. The QoS-Monitor for the communication subsystem was presented as an example of a management component located closer to the data path. It is flexible according to monitored QoS parameters, thereby leading to a reduced overhead. Finally, comparison of performance measurements for QoS-Monitor and QoS-Manager confirm the separation of management functions with different time constraints into autonomous components.

The cooperation with RSVP as network reservation protocol is planned next. Moreover, management strategies of the QoS-Manager have to be examined and tested in several scenarios. This especially includes heterogeneous scenarios where session participants have very different capabilities to support QoS. However, the achieved results show the general practicability of QoS management systems for high performance networks.

REFERENCES

Blakowski, G. & Steinmetz, R. (1996), 'A Media Synchronization Survey: Reference Model, Specification, and Case Studies', *IEEE Journal on Selected Areas in Communication* **14**(1), 5–35.

Boykin, J., Kirschen, D., Langerman, A. & Lo Verso, S. (1993), *Programming under Mach*, Addison-Wesley.

Braun, T. & Diot, C. (1996), 'Performance Evaluation and Cache Analysis of an ILP Protocol Implementation', *IEEE/ACM Transactions on Networking* **15**(3), 318–330.

Campbell, A., Aurrecoechea, C. & Hauw, L. (1996), A Review of QoS Architectures , *in* 'Proceedings of the 4th IFIP International Workshop on Quality of Service', *in* 'GMD-Studien Nr. 282', Sankt Augustin, Germany, 173–195.

Frick, O. & Schmidt, C. (1996), Service Support for Multiuser Multimedia Applications, *in* 'Proceedings of the Third International Workshop for Multimedia Systems (PROMS'96), October 1996', Madrid, Spain.

Hutchison, D., Coulson, G., Campbell, A. & Blair, G. S. (1994), Quality of Service Management in Distributed Systems , *in* 'Network and Distributed Systems Management' (ed. M. Sloman), Addison-Wesley, 273–302.

IETF Stream Protocol Working Group (1995), Internet Stream Protocol Version 2 (ST2), Protocol Specification – Version ST2+, *in* 'RFC 1819' (eds. L. Delgrossi & L. Berger), IETF.

Nahrstedt, K. & Smith, J. M. (1995), 'The QoS Broker', *IEEE Multimedia*, Spring 1995, 53–67.

Nahrstedt, K. & Steinmetz, R. (1995), 'Resource management in networked multimedia systems', *IEEE Computer* **28**, May 1995, 52–63.

Schmidt, C. (1997), *Integrierte Management Architektur für qualitätsorientierte Kommunikationsdienste*, PhD thesis, internal document, University of Karlsruhe, Institute of Telematics.

Schmidt, C. & Bless, R. (1996), QoS Monitoring in High Performance Environments, in *Proceedings of the 4th IFIP International Workshop on Quality of Service*, see (Campbell et al. 1996), 199–207.

Schmidt, C. & Zitterbart, M. (1995), Towards Integrated QoS Management, *in* 'Proceedings of the 5th IEEE Computer Society Workshop on Future Trend of Distributed Computing Systems', Cheju Island, Korea.

Zhang, L., Deering, S., Estrin, D., Shenker, S. & Zappala, D. (1993), 'A New Resource Reservation Protocol', *IEEE Network* **7**(5), 8–18.

6 BIOGRAPHY

Roland Bless received his diploma degree in Computer Science from the University of Karlsruhe in 1996. Since 1996 he is working as a research assistant at the Institute of Telematics, University of Karlsruhe, Germany. His main interests are high performance integrated networks, QoS management, and, network and operating system support for real-time applications.

Matthias Jacob is a graduate student in Computer Science at the University of Karlsruhe, Germany. His main interests are high performance networking and parallel computing. He is also an active student member of the ACM.

Claudia Schmidt is working as a research assistant at the Institute of Telematics, University of Karlsruhe, Germany, since 1992. She received her diploma degree and PhD in Computer Science from the University of Karlsruhe in 1992 and 1997, respectively. Her main interests include integrated QoS management, resource reservation, and flexible protocol support for multimedia applications.