A Quality of Service Abstraction Tool for Advanced Distributed Applications

A. Schill, C. Mittasch, T. Hutschenreuther, F. Wildenhain

Dresden University of Technology, Faculty of Computer Science, 01062 Dresden, Germany
e-Mail: [schill|chris|tino|wildhai]@ibc.inf.tu-dresden.de

Abstract

High performance communication protocols are being used in our lab as an experimental base for advanced distributed applications with a special focus on quality of service (QoS) characteristics. During our recent work, we recognized that quality of service abstractions would prove very useful for facilitating distributed application development. As a first base, we present a management tool for inspecting and controlling quality-of-service related characteristics on top of XTPX (eXpress Transfer Protocol Extended) and TCP/IP. Based on a classification of QoS requirements of various multimedia data streams, the design and implementation of a higher-level QoS manager tool is described then. We finally illustrate how such an approach will be integrated with traditional client/server platforms such as the OSF Distributed Computing Environment.

Keyword Codes: C.2, C.2.3, C.2.4

Keywords: Computer Communication Networks, Network Operations, Distributed Systems

1. Introduction

High performance transport systems have to support flexible QoS characteristics in order to enable advanced distributed applications, including multimedia communication. XTPX /MIB94/ as an extension of XTP /STR92/ allows the specification, usage, and monitoring of such QoS related parameters. It is therefore being used as a base for experimenting with high performance communication in our lab. TCP/IP is also used in parallel.

However, during our initial experimentation and analysis, we also identified several advanced requirements. These are mainly concerned with network management support, with the level of abstraction of QoS specifications, and with the integration of high performance communication facilities into existing distributed systems frameworks:

- *Management support:* Based on the existing management specification for XTPX /MIT94/, advanced runtime and tool support for QoS-based management is required. An appropriate management protocol with additional management policies is desirable. This will also enable dedicated QoS management applications or at least a QoS supervision in case of TCP/IP.
- *QoS specification:* The level of QoS specification of existing protocols is relatively low; it is required that applications directly specify parameters such as data rate, jitter, error rate etc. It is desirable to introduce facilities for higher-level, application-related QoS specifications. The task of an appropriate support tool is to map these specifications onto low-level QoS parameters automatically while providing sufficient flexibility, too.

- **Framework integration:** Distributed applications are widely supported by standardized distributed systems frameworks with services such as remote procedure call (RPC), naming, and security. It will provide significant benefits to integrate QoS-related high performance communication facilities with the services of such distributed platforms.

The paper addresses these requirements by the following contributions: After a brief discussion of related work in section 2, section 3 presents the architecture and implementation of a new management framework for QoS support. Advanced concepts for a semi-automatic threshold management as a base for QoS related control mechanisms are also discussed. Section 4 first presents an analysis of distributed multimedia application requirements concerning QoS. The relationship of the most important communication facilities and media encoding standards with QoS parameters is discussed furthermore. Based on the results, a higher level QoS management tool is described. It supports an abstract, application-related QoS specification and maps it onto concrete QoS parameters. Section 5 illustrates how these concepts can be integrated into a distributed systems architecture such as the OSF Distributed Computing Environment (DCE) /OSF94, SCM93/. Finally, section 6 concludes with an outlook to future work.

2. Related Work

Similar QoS management approaches and distributed multimedia systems have been developed recently. The *Cinema* architecture /BDH93/ provides programming abstractions for distributed multimedia applications with explicit QoS mapping functionality. Abstract components of an application are partitioned into more detailed components that are finally mapped onto basic resources (such as CPU, communication bandwidth or buffer storage). Future research within *Cinema* will address the details of this mapping functionality. Within the Touring Machine Project /TMS93/, an infrastructure for advanced multimedia communication is developed, too. Abstract ports and connections enable the representation of point-to-point and point-to-multipoint interactions for audio, video and other data streams. Connections are embedded in logical sessions with an explicit session management component. An effective abstraction for multimedia programming is offered, based on a high-level API. However, details of QoS management are not addressed yet.

Such QoS issues are addressed by /VOG94/ to a QoS negotiation approach, for example. An extended QoS architecture that also addresses flow control of media streams is presented in /CCH94/; it is similar to our approach but is more oriented towards the transport layer. /MBA94/ discusses a QoS monitoring approach to identify QoS problems dynamically, similar to the threshold management facility discussed below. In /ADB94/, an extension of the ANSA architecture with continuous media communication facilities is presented. Based on multimedia device and stream abstractions, distributed multimedia applications can be configured. Moreover, media synchronization requirements can be specified explicitly in a script language. An integration with OSF DCE has also been outlined. Considering these and other related approaches discussed below, we present our QoS abstraction facilities in the following sections.

3. Management Architecture

Our design and implementation of a QoS-related management architecture is based on the following requirements: (1) It shall be possible to analyze and modify QoS parameters of the transport protocol in a flexible, decentralized way based on managed object (MO) specifications. (2) Management shall be supported by a graphical interface, and MOs shall be grouped into logical categories. (3) The implementation shall be based on SNMP (Simple

Network Management Protocol) and on public domain components. (4) A major focus shall be put on layer management and on mid-term management tasks.

General model:

The approach is based on the traditional SNMP interaction model /CFS90/. A management application on the manager's site observes, analyzes and controls several management agents representing protocol instances. Each protocol instance comprises a MIB that is specified in a standardized way and is mapped onto internal protocol parameters by kernel-level access functions. The interaction between a manager and an agent is based on SNMP Get, Next, and Set requests for accessing single MOs, for reading groups of MOs, and for modifying MOs, respectively. Moreover, it is possible for an agent to issue a trap for immediately notifying a manager of a significant status change concerning MO characteristics.

Management architecture and implementation:

Based on this management model, we designed and implemented a management architecture for the XTPX protocol (see fig. 1) and also partially applied it to a TCP/IP environment. It specifically addresses layer-3 and -4 management. On the site of the manager, a MIB catalog has been defined in ASN.1. It comprises the major MOs for configuration and performance management. The specification has been transformed into an internal representation using the MOSY compiler of the ISO Development Environment (ISODE).

Figure 1: Management architecture

The manager provides access to the remote agents based on the existing MO specifications. It has been implemented by using the tricklet v1.4 tool of Delft University. In particular, it offers an API that has been used for integrating a MIB browser. This XMIB browser is based on the xmib1.0b implementation of Hebrew University in Jerusalem. The agent implementation uses snmp1.5 of Carnegie Mellon University. It accesses a uniform agent MIB that is eventually being mapped onto the internal XTPX MIB (or the simpler TCP/IP MIB) via access routines.

All components have been integrated and have been ported onto the OSF/1 operating system. The relevant MOs have been defined and have been integrated into the architecture. Examples of typical MOs according to the management areas of layer 4 of XTPX are:

Configuration management:

- ConfigMaximumSizeReceiveWindow (receiver's default window size)

- ConfigReceivingBufferSize (receiver's buffer size)
- ConfigSendingBufferSize (sender's buffer size)
- ConfigMaximumNumberCNTLRetries (retries until acknowledgement is delivered)
- ...

Performance management:

- PerfConnectionsAttempt (number of attempted connection establishments)
- PerfConnectionsEstablished (number of successfully established connections)
- PerfNumberOfSentPackets
- PerfNumberOfReceivedPackets
- ...

Some of these parameters are for monitoring only (such as the number of sent packets), and can be used for long-term decisions concerning the adaptation of QoS related parameters. Others can be directly adapted (such as the buffer sizes) and can therefore be used to implement QoS management strategies (for example, addressing jitter control features).

Support for the MIB-II specification has been enabled by defining various MIB-II MOs at the agent site. MIB-II support is already being provided by the manager and the browser. The manager can also be configured in a flexible way by adapting the timeout interval and retry counter for agent interactions. The tools initially did not support write access to MOs. Therefore, several extensions for enabling the set routines were made. Moreover, the Next operation was also implemented by supporting flexible browsing of tables within the manager.

In summary, the approach enables a flexible interactive management of XTPX. It provides a comfortable graphical interface with an explicit MO tree representation, enables standardized access to potentially heterogeneous agents via SNMP, and is based on highly portable public domain tools. In the case of TCP/IP, it is simpler due to the lack of specific QoS related parameters but still provides a base for the threshold management as discussed below.

Figure 2: Architecture of threshold management

Extension: Threshold management:

A recent extension includes the definition of a management API at the manager's site, as well as the design and implementation of management applications. In particular, dedicated threshold management is supported based on the implemented layer management functionality (see fig. 2). A separate threshold demon observes the status of the respective protocol instance and notifies a trap demon on the manager's site in case of threshold violations. After initializing an appropriate threshold (1), an acknowledgement is sent to the manager (2). The threshold monitoring is based on a local interactions between the threshold demon and the SNMP demon (3). Threshold violations are propagated via SNMP traps (4), and the trap demon may inform the management tool about it (5).

The thresholds themselves can in principle be modeled and implemented via different alternatives:

- *Explicit MOs*: In this case, a threshold is represented as an MO explicitly. The advantage is relative simplicity, the disadvantage is the overhead associated with the various MOs.
- *Global table*: With this alternative, only one MO per agent is used. It represents a global table with references to all other MOs that are to be monitored, together with the respective threshold values. This alternative is more flexible and can be implemented more efficiently.
- *Logical names*: With this approach, the manager specifies the MOs to be monitored by logical names. This way, a flexible naming approach is introduced that is easy to use. However, a mapping to the internal MOs must be provided.
- *Semantic specification*: This alternative is even more far-reaching. MOs are specified by names or by semantic constructs, and the threshold calculation functions can also be given by the manager based on semantic expressions. This approach is most flexible but difficult to implement.

We selected the second alternative due to its relative efficiency and implementability. The approach also requires the definition of reasonable thresholds for QoS related parameters such as roundtrip time, throughput, or dropped packets. Based on these definitions, the agents perform a decentralized threshold monitoring by explicit threshold checks, and can notify the manager using the existing trap feature of SNMP within our architecture. This also requires new facilities for threshold initialization. In particular, the threshold values have to be specified in advance, also providing reasonable defaults. These issues are addressed in the next section.

4. Quality of Service Abstractions

The management facilities discussed above support monitoring and manipulation of explicit QoS parameters. However, for distributed applications, a higher level of abstraction concerning QoS specification is desirable. An application should not be required to explicitly deal with parameters such as buffer sizes, delay or jitter. It should rather specify its requirements in terms of the data streams or media to be communicated. This should be supported by reasonable defaults and should cover a wide range of communication scenarios, not limited to multimedia only.

Analysis of QoS requirements:

Based on these considerations, we performed a detailed analysis of QoS requirements of existing distributed communication scenarios and media encoding schemes. The major goal was to specify a set of predefined QoS classes that can automatically be mapped onto QoS parameters. The overall classification is illustrated in fig. 3. According to /SCZ93/, we defined 4 major classes, dividing application requirements into reliable and unreliable communication

and time-dependent and time-independent interactions, respectively. The resulting classes may partially overlap; for example, the transfer of a pixel image may either be time-constrained or not although no isochronous traffic characteristics are required.

isochronous	non isochronous		
unreliable	reliable		unreliable
time-dependent		time-independent	
I	II	III	IV
speech, music, video, HDTV realtime text, graphics, images	realtime text, graphics, compressed images, data	text, data graphics, images	text, graphics, compressed images

Figure 3: Classification of distributed interaction scenarios

For each kind of interaction, the basic characteristics and the associated QoS parameters have been analyzed. Results for audio and video are summarized in figure 4. For low-quality interactive audio communication, relatively high residual error rates can be tolerated, and relatively low throughput is sufficient. This changes significantly for high-quality audio based on CD or MPEG encoding. The delay and jitter values have to be constrained in any case in order to guarantee reasonable quality of service characteristics. This is similar for video, however, much higher throughput must be supported. The residual error rate, on the other hand, can be significantly higher without compromising quality of service.

For conventional file transfer and RPC interactions, there are no jitter constraints, of course. However, the delay should be constrained in the range of several milliseconds up to one second for data-intensive interactions. The residual error rates must be close to zero. Analysis of various other kinds of communication scenarios has been performed, too. As an example for mapping the results onto QoS parameters in more detail, a sample QoS specification for JPEG-based binary image transfer is shown below:

	audio				video						
encoding formats	IMA 8 kHz	IMA 11,025 kHz	IMA 22,05 kHz	IMA 44,1 kHz	CIF	SIF	CCIR 601	HDTV Europa	H.261	MPEG	MPEG II
throughput (Mbit)	0,064	0,0861	0,1722	1,378	37	30,5	206	2373	0,064 - 1,92	1,5	4 - 10
residual error rate	10 ⁻²	10 ⁻⁸	10 ⁻⁸	10 ⁻⁸	10 ⁻⁴	10 ⁻⁴	10 ⁻⁴	10 ⁻⁴	10 ⁻⁶	10 ⁻⁶	10 ⁻⁵
delay (ms)	50 - 100										
jitter (ms)	5 - 10										

Figure 4: QoS characteristics of audio and video

```
jpeg.throughput.min = 1000; /* bit/s, default minimum */
jpeg.throughput.target = FileSize / 100000; /* for a transfer in typically less than a second */
jpeg.throughput.max = 5000000; /* assumption */
```

```

jpeg.throughput.lower_threshold = jpeg.throughput.min;
jpeg.throughput.upper_threshold = jpeg.throughput.max;
jpeg.transit_delay.min = 5; /* ms */
jpeg.transit_delay.target = 50;
jpeg.transit_delay.max = 100;
jpeg.max_delay_jitter = NO; /* no jitter requirement */
jpeg.max_error_rate = 10-6;
jpeg.max_establishment_delay = 200; jpeg.max_establishment_failure_probability = 10-3;
jpeg.max_transfer_failure_probability = 10-6;
jpeg.max_resilience = max;
jpeg.max_release_delay = 500;
jpeg.max_release_failure_probability = 10-3;
jpeg.target_error_handling = 4; /* error correction for lost packets and bits */
jpeg.target_guarantee_class = 1; /* reliable service, therefore bandwidth reservation desired */
jpeg.target_protection = 0; /* no protection */
jpeg.target_priority = 100; /* medium priority */
jpeg.max_cost = min; /* no realtime service */

```

QoS manager tool:

Based on these results, we designed and implemented a QoS manager tool in order to provide the desired higher level of QoS abstraction. The integration of the tool is shown in fig. 5. An application requests to establish a connection for a specific data stream. Rather than specifying the QoS requirements in detail at the transport layer interface, it sends a request to the QoS manager tool, specifying the desired QoS class and the selected kind of media encoding (the tool is typically co-located with the application or even residing within the same process). The tool analyzes the input and issues a complete low-level QoS data structure according to the given requirements. The result is passed back to the application and is used for connection establishment. Should the connection fail due to limited resources, the application contacts the tool once more, requesting a reduced set of default QoS parameter values. This way, the connection establishment may afterwards succeed, providing some reduced but still acceptable QoS characteristics. In the case of TCP/IP, QoS characteristics can not be specified directly at the protocol level but can at least be monitored by our threshold management, leading to a similar approach.

Figure 5: Integration of QoS manager tool

In summary, the application is relieved from specifying QoS details at the transport system interface. It can make use of predefined QoS characteristics; these definitions can also be easily extended and adapted according to specific application requirements. They are structured as a class library of predefined media communication classes; the inheritance structure of the library is shown in fig. 6. The abstract class *channel* provides basic mechanisms for connection management. Basic, time-independent communication facilities are offered by *BulkDataChannel* (for regular mass data transfer) and *COControlDataChannel* (for connectionless transfer of the control data of an application). As opposed to that, *TimeDependentChannel* enables the dynamic supervision of time-related QoS parameters based on our threshold management approach. Further distinctions are made concerning the error handling at the protocol level; for various compressed data streams with less redundancy (such as MPEG or adaptive PCM), reliable channels are required while unreliable transfer is sufficient for other encoding formats. The inheritance tree is extensible in order to introduce other kinds of media encoding formats (such as MPEG2, for example).

Figure 6: Media object class library

At instantiation time, predefined QoS characteristics can be overridden for a specific object of one of these classes. For example, a high- or low-quality video according to the application requirements and the expected available bandwidth can be selected. The QoS manager tool can provide additional adjustments based on the described two-phase approach as discussed above. The detailed QoS specification is performed by using the (overloaded) constructors of the media channel classes. Various levels of abstraction are available as illustrated by the following example:

```

MPEG1Channel (...);
MPEG1Channel (... , int min_throughput);
MPEG1Channel (... , QoS *user_qos);
MPEG1Channel (... , int resolution, int frame_rate, QoS *user_qos);

```

In addition to some regular control parameters (not shown here), QoS-specific parameters can be specified. In the first case, however, they are defaulted by the system. The second constructor allows the specification of throughput requirements with defaults for other QoS parameters, still at a rather low level. The third constructor enables a direct layer-4-level-specification of QoS with all details. The last constructor given above provides for application-level QoS specification (for example, a resolution of 768x576, a frame rate of 30 per second and a maximum delay of 90 milliseconds). Even higher levels of abstraction have been considered (namely, abstract media quality classes), but have not been implemented yet. The corresponding QoS data structure used in two of the constructors has the following outline (based on QoS in XTPX, but generalized and now protocol-independent):

```

struct QoS (
    long min_throughput;
    long lower_threshold_throughput;
    long target_throughput;
    long upper_threshold_throughput;
    long max_throughput;

```

```

long target_delay;
long upper_threshold_delay;
long max_delay;
long max_jitter;
long max_establishment_delay;
long max_release_delay;
long target_error_rate;
long max_error_rate;
long target_priority;
long min_protection;
long max_cost;
);

```

The structure comprises absolute limits, target values, and threshold values. Absolute limits specify the intervals of QoS values that have to be guaranteed in any case. Target values are recommended and should be achieved if possible. Threshold values are within the absolute intervals and are used for threshold management. This way, the system can react and take appropriate measures before the limits are reached, using our threshold management approach. For all fields of the QoS structure, explicit values can be given. However, the application can also request that the system should use an appropriate default (indicated by *REQ_EMPTY*) or can state that the parameter should not be used for QoS management at all (*REQ_NO*).

Several alternatives and extensions have also been considered. It would also be possible to implement the tool as a complete intermediate layer between the application and the transport system. This way, the tool would be responsible for connection management, too. We initially rejected this solution as we wanted to keep the QoS tool independent from the regular connection management. This way, mapping of QoS details onto various different protocols is enabled in a flexible way. In particular, with the emerging IPng (next generation) protocol in the Internet, this will become very important. It will also be interesting to use various underlying reservation protocols, namely RSVP. Such alternatives will be investigated in more detail in the near future.

5. Integration into Client/Server Programming Models

Motivation:

The presented approach provides QoS abstractions for relatively straightforward communication scenarios. However, general distributed applications require more advanced support based on standardized distributed platforms /NWM93/. Most approaches of this kind are based on the client/server model and are using an RPC-style interaction mechanism. In particular, hetero-geneous environments are supported by canonical data transfer formats. These facilities are typi-cally being enhanced with directory services for mapping servers to client requests by logical na-mes, and by security services for guaranteeing authorization and privacy in distributed environments.

However, such approaches are not yet using high performance communication protocols nor considering QoS aspects at all. On the other hand, protocols such as XTPX do not provide any facilities above the transport layer and therefore lack the already well-established functionality of distributed platforms. For these reasons, an integration of both approaches is most desirable. As a base for our integration work, we are using the OSF DCE /OSF94/.

Integration approach:

The basic integration architecture is outlined in fig. 7. A distributed application consists of several decentralized components that regularly communicate via remote procedure call. Initially, components locate each other by querying the directory service for peer components based on their name and on their static characteristics. Moreover, RPC communication can be protected by encryption and access control. Even in advanced multimedia applications, the control part of all interactions (such as the management of dynamic participation in a videoconference) be implemented in a comfortable way using RPC. Major advantages are similarity with local procedure calls, masking of operating system and hardware heterogeneity, and relative reliability.

Figure 7: Integration with OSF DCE

However, it is also possible to pass control objects via RPC that initiate the establishment of an media channel in addition to existing RPC interaction paths. For example, an RPC parameter may comprise a reference to a document with audio and video annotations. The server of an RPC interaction does not receive the actual media data but just the media control information this way. Thereafter, it may initiate additional calls to the received parameter objects in order to access and display the media information. Due to the discussed QoS requirements, this cannot be implemented effectively by conventional RPC; rather, it is reasonable to bypass RPC, open a connection in a transparent way, and transfer the required data while guaranteeing QoS characteristics. This will be achieved by furnishing RPC control objects with an interface to the QoS manager tool and to the transport system.

This initial integration will present a first step towards high performance client/server applications. However, a more seamless integration is desirable. Therefore, we are currently designing a QoS-based class library that offers high performance RPC interaction channels (to be mapped onto the transport protocol), and various media object classes imbedded into an inheritance hierarchy. Instances of these classes represent specific media with given encoding

formats and individual characteristics (such as frame rates of video objects). By parameterizing channels with media objects, it will be possible to automatically select QoS characteristics for applications in the traditional world of client/server interactions.

6. Conclusions and Future Work

This paper presented a management framework with special respect to QoS characteristics. Based on SNMP, a management architecture has been designed and implemented; it allows for interactive, graphical access to QoS parameters via managed objects. Moreover, it supports interactive modifications of such parameters and enables functional extensions towards threshold management. We also presented a classification of application-level QoS requirements and derived a QoS abstraction architecture and tool. This approach supports a high-level specification of QoS requirements, relieving the application from QoS details. It has been outlined how the approach can be integrated with traditional client/server platforms for distributed systems.

Future work will focus on the refinement of QoS requirements and characteristics. It will be investigated how the QoS requirements of combined media streams can be specified and how such specifications can be imbedded into an overall application framework. The results will be used as a base for implementing a more advanced QoS manager tool. The implementation will also address a seamless integration with OSF DCE and shall be validated by several examples. Moreover, the installation and usage of a local ATM network /BOU92/ is also planned for the near future in our lab. This will raise the question how the discussed QoS characteristics can be mapped onto ATM and how applications can be supported in a mixed environment with ATM networks and conventional Ethernet-based communication.

Acknowledgements

We would like to thank all colleagues and students who contributed to the design and implementation of the presented approach. In particular, we are most grateful to Stephan Albrecht, Frank Breiter, Samer Habib, Ulf Kieber, Olaf Kiese, Jörg Kretschmar, Sabine Kühn and Jens Watzek for their important contributions. An earlier version of the paper was presented at the Workshop on new protocols for multimedia systems, Berlin, June 1994.

References

- /ADB94/ Adcock, P., Davies, N., Blair, G.: Supporting Continuous Media in Open
Distri- buted Systems Architectures; Lancaster University, Dep. of Computing,
1994
- /BDH93/ Barth, I., Dermier, G., Helbig, T., Rothermel, K., Sembach, F., Wahl, T.:
CINEMA: Eine konfigurierbare, integrierte Multimedia-Architektur;
GI/ITG Workshop "Verteilte Multimedia-Systeme", Stuttgart, 1993
- /BES94/ Besse, L., Dairaine, L. et.al.: Towards an Architecture for Distributed Multi-
media Applications Support; Proceedings of the International Conference on
Multimedia Computing and Systems, Boston, Massachusetts, USA, May 1994
- /BOU92/ Boudec, J.Y.: The Asynchronous Transfer Mode: A Tutorial; Computer
Networks and ISDN Systems, Vol. 24, 1992, pp. 279-309
- /CCH94/ Campbell, A., Coulson, G., Hutchison, D.: A Quality of Service Architecture;
ACM SIGCOMM Computer Communications Review, Vol. 24, No. 2,

- April 1994, pp. 6-27
- /CFS90/ Case, J.D., Fedor, M., Schoffstall, M.L., Davin, C.: Simple Network Management Protocol; RFC 1157, May 1990
- /MBA94/ Mourelatou, K., Boulloutas, A., Anagnostou, M.: An Approach to Identifying QoS Problems; Computer Communications, Vol. 17, No. 8, Aug. 1994, pp. 563-570
- /MIB94/ Miloucheva, I., Bonnesz, O.: XTPX - Version 1.3 Technical Documentation; Technical University of Berlin, 1994
- /MIT94/ Miloucheva, I., To, K.: XTPX Management Information Base (MIB); Design and Implementation; Technical University of Berlin, PRZ Document, 1994
- /NWM93/ Nicol, J.R., Wilkes, C.T., Manola, F.A.: Object-Orientation in Heterogeneous Distributed Computing Systems; IEEE Computer, Vol. 26, No. 6, 1993, p. 57-67
- /OSF94/ Distributed Computing Environment - An Overview; Open Software Foundation, 1994
- /SCM93/ Schill, A., Mock, M.: DC++: Distributed Object-Oriented System Support on top of OSF DCE; Distributed Systems Engineering Journal, Vol. 1, No. 2, 1993
- /SCZ93/ Schill, A., Zitterbart, M.: A System Framework for Open Distributed Processing; Journal of Network and Systems Management, Vol. 1, No. 1, 1993, 71-93
- /STR92/ Strayer, W.T., Dempsey, B.J., Weaver, A.C.: XTP: The Xpress Transfer Protocol. Addison-Wesley, 1992
- /TMS93/ Touring Machine System Project: Touring Machine System; Comm. of the ACM, Jan.1993, Vol.36, No. 1
- /VOG94/ Vogel, A., Kerherve, B. et.al.: QoS-Negotiation for Distributed Multimedia Presentational Applications; Proceedings of Workshop on Distributed Multimedia Applications and Quality of Service Verification, Montreal, Quebec, Canada, May /June, 1994