A Framework for Negotiable Quality of Service in Distributed Multimedia Systems

Gabriel Dermler, Walter Fiederer, Ingo Barth, Kurt Rothermel

CR-Klassifikation: C.2.4, D.4.7, H.5.1

**A Framework for
Negotiable Quality of Service in
Distributed Multimedia Systems**

G. Dermler, W. Fiederer, I. Barth, K. Rothermel

Fakultätsbericht 10/1995
Technical Report
Dezember 1995

Fakultät Informatik
Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20 - 22
D-70565 Stuttgart

Abstract

Distributed multimedia applications offer to clients degrees of freedom for selecting Quality of Service (QoS). This paper describes a framework for application level negotiation as a means to regulate client requested QoS under QoS constraints imposed on an application. A QoS architecture is motivated interrelating levels for media specific and transport level QoS handling. The necessity of corresponding protocol levels is explained. The coupling between these levels is demonstrated for a concrete protocol developed for an example application. The embedding of end-to-end delay into the protocol is discussed.

1 Introduction

Distributed multimedia applications are employed to generate, process and comsume (e.g. present) continuous (e.g. audio, video) data streams across distributed locations. An application client has a service oriented view towards QoS provided by the application. It specifies QoS with respect to consumed data (e.g. presented video frame size and rate) and expects the application to agree on a specific QoS selection. Providing such an agreement requires a process orchestrating client QoS requests and QoS constraints imposed on the application (for instance) by restricted resource availability. We refer to this process as application level QoS negotiation.

So far, description of negotiation is in analogy to negotiation at transport level as currently pursued for various transport system designs (e.g. [VHN92], [MMR93]). However, a number of features make QoS selection at the application level a different and potentially much more complex task. It is the goal of this paper to elaborate these features and to interrelate them in an overall framework allowing the development of negotiation protocols for distributed multimedia applications. In particular, such a protocol is developed for an example application demonstrating in this way the feasibility of introduced concepts.

The paper is structured as follows. In Section 2, we introduce an abstract model for constructing distributed multimedia applications. We use this model in Section 3 to motivate and position two distinct QoS levels required during negotiation and describe the relative position of application clients issuing QoS requests. The relationship between the QoS levels is completed by appropriate mapping functions.

Section 4 motivates the necessity of two protocol levels referring to the existence of different QoS levels and application topologies. It describes the role of each level by introducing a negotiation protocol in the context of an application example, in particular highlighting the coupling required between the protocol levels. Section 5 indicates how issues concerning end-to-end parameters such as delay can be integrated into the introduced protocols. Section 6 summarizes important conclusions and gives a status report on current implementation work.

2 Application Model

In this section we introduce a concept for constructing distributed multimedia applications. Similar concepts are pursued by various research groups ([KHSM95], [BCA$^+$92], [MKSD90]) including the group defining IMA MMS [IMA93]. We describe here the terminology used for our *Cinema* development platform. For a detailed description of *Cinema* refer to [RBH94].

Cinema allows a client to compose an application out of components and links. Components encapsulate processing of multimedia data, e.g. for generating, presenting or manipulating data. To provide a uniform data access point for the components, ports are used that deliver data units to the component (input port) or take the data units from the component (output port). A client constructs an application by specifying a topology of components interconnected via links. A link provides an abstraction from underlying communication mechanisms which may be used to perform the transport of data units. Figure 1 shows an example topology composed of two video components generating two video streams which are mixed by a video mixer and displayed by a 3D monitor component.

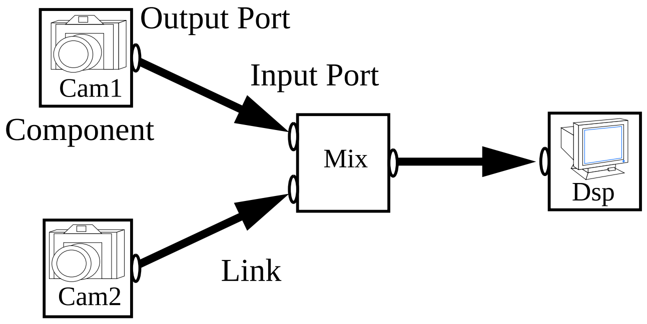

Figure 1: Example of an application topology

Before using an application, a client has to indicate desired QoS to *Cinema*. For this, *Cinema* offers the concept of a session. A session is the unit of resource reservation allowing the client to specify the application to be instantiated and the QoS expected with respect to output generated by sink components. The goal of negotiation is then to settle on the best possible QoS complying with client requests under two kinds of restrictions: limited functional capabilities of components (e.g.: design enforcing processing of video frames only up to a maximal size) and limited resource availability for links and components.

3 QoS Architecture

The necessity of different QoS levels arises from the type of data encountered at component ports. Such ports are employed to feed components for processing and to obtain processed data for distribution to subsequent components. Internal processing of a component requires data in a formatted form. For instance, a video component may expect data as a sequence of frames with a certain picture size and picture rate. For each existing component port, a description has to exist in order to indicate externally the characteristics of data expected. Such information is a prerequisite to negotiation, for instance, checking whether two component ports can be interlinked in a compatible fashion.

Characteristics of processed and communicated data are media specific. For video, description parameters frequently include video picture size, picture rate, compression scheme and ratio. For audio, similar parameters may be used including sample rate, sample size and encoding scheme. Media specific differences are being given at least in form of different implied dimensions and value ranges. In terms of abstraction levels, such parameters are media specific since they have no unambiguous transport level representation. For instance, a pair of (video frame size, frame rate) values can be mapped in various ways onto a pair of (packet size, packet rate) values.

From the above, it is obvious that any distributed multimedia system has to include two levels of QoS: (a) a “higher” media specific level describing the communication load requested by interconnected components and (b) a “lower” level describing the communication load requested at the service interface of transport systems. Figure 2 shows the positioning of QoS levels in *Cinema*. At the level of components and links, media specific parameters are exchanged in so-called application flowspecs (AFS), while transport level parameters are employed only

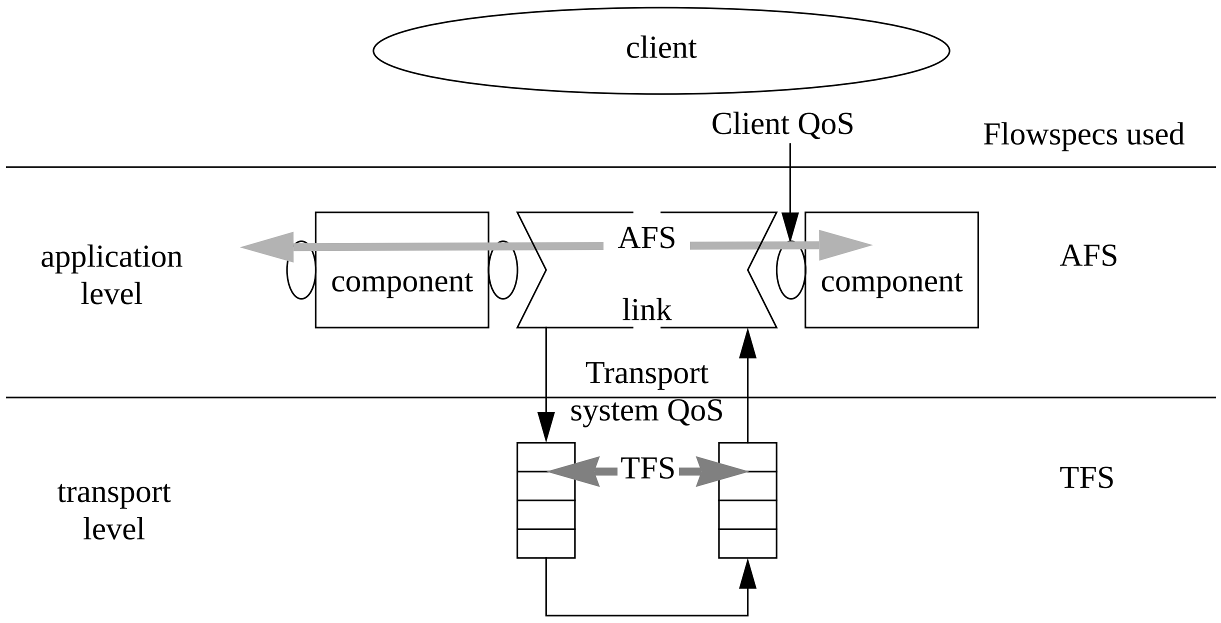

Figure 2: QoS Architecture

inside links within transport flowspecs (TFS). The figure shows that a client specifies its QoS requests with respect to the input ports of sink components, i.e. in terms of media specific parameters expected there. The following discussion motivates the architecture.

Cinema offers a basic level for QoS specification. The client itself may be given as a program converting media specific parameters into higher level abstractions for (human) end users (e.g. “high”, “mid” and “low quality” video). Client requests are related to sink component ports, assuming that the client has knowledge about sensed output QoS and corresponding media specific QoS required as input to sink components.

As mentioned earlier, component design involves the selection of a media specific parameter set to be used by a component at a certain port. Since we do not expect such parameter sets to be defined too often, we make a specific set of parameters available in form of streamtype objects including, besides the media specific parameter list, the possible value ranges for each parameter. In consequence, a component designer has to associate with each component port the streamtype to be used making all related information available at the port.

Components are designed to be reusable in various application topologies.[1] In particular, component design is kept independent of data transport characteristics. Differences exist here since current transport systems expect and handle transport level QoS differently. Link objects abstract from such peculiarities and offer a uniform external view. As shown in Figure 2, a link object receives media specific parameters at its interface from which it derives the QoS representation needed by the encapsulated transport system. But in order to reduce the number of implemented link types, link implementation is to be kept independent of any media specific characteristics. These requirements lead to the following structuring of the mapping from media specific to transport system parameters.

A first mapping, termed DownQM, is provided by streamtype objects at component ports.[2] DownQM maps media specific parameter values (e.g. video frame size and rate) onto a so-called uniform communication load representation (UCL) defined by *Cinema*. UCL is positioned at the transport level, its purpose is to abstract from (QoS) interface peculiarities of a specific transport system. Given DownQM, a link implementation has to include only a function mapping a UCL description onto the QoS representation expected by the specific transport system. A link object handles media specific parameters since it has access to both described functions. It can use its internal mapping and can call DownQM since it has access to component ports connected to the link. Given this structuring, a new link implementation is required only if it encapsulates a new transport system (e.g. for radio communication).

Mapping to UCL

The selection of UCL parameters was based on parameter selections pursued by current multimedia transport systems. It was guided by the requirement to describe application generated load both for compressed and uncompressed media streams as shaped for instance by MPEG [Gall91] or JPEG [JPEG93] compression. Optimal selection of parameters for load description

[1] Two interconnected component ports have to be associated with the same streamtype.
[2] Another function, NextQM, is also provided by a streamtype object (see Section 4 for motivation).

is a continuing research issue. In particular, a future standardized transport QoS description may be used for UCL, should one emerge. The following UCL parameters are currently defined:

- peak data unit size (*peak*)
- average data unit size (*avg*)
- data unit rate (*rate*)
- average interval (*iv*)

For uncompressed streams the use of these parameters is straightforward. For audio, a UCL data unit would encompass the amount of audio processed by a component at once (e.g. 40 ms of Audio. *Peak* and *avg* would be both set to a corresponding number of bytes, while *rate* would be fixed to 25 data units/s. For uncompressed video, a UCL data unit would encompass the amount of video processed at once, typically a picture size, while *rate* would be set to the video frame rate. However, different mapping schemes could be defined.[1].

c. quality	192 x 144	256 x 192	384 x 288	512 x 384	768 x 576
10	4.1	6.0	8.6	16.0	29.2
25	4.7	6.6	9.4	17.5	31.8
50	5.9	8.4	12.2	22.1	40.0
70	7.8	10.4	15.4	28.6	58.1
85	9.8	15.2	24.1	45.0	80.0
95	16.6	26.7	44.8	69.8	131.6

Table 1: UCL data unit sizes in KB

For JPEG compressed video, we have to include the compression effect. We took an empiric approach to measure the size of a UCL data unit resulting from a picture of a given size, color depth (fixed to 24 bit) and compression quality (given as a number between 1 to 100). Table 1 contains some measurement results. Calculating UCL values then amounts to:

[1] Recall that DownQM encapsulates this mapping.

peak = *avg* = table entry for desired picture size and compression quality

rate = f_rate

The mapping may be defined differently, for instance by using *peak* and *avg* to reflect variations of compressed picture sizes over a longer time interval. Our UCL description scheme does allow for this, though further experimentation is required for deriving corresponding (*peak*, *avg*) value pairs as table entries. This latter approach is currently taken for MPEG compressed streams, where we measure (*peak*, *avg*) pairs. *Peak* denotes the maximum compressed picture size encountered (typically an I coded frame), while *avg* is an average value for an average interval which encompasses at least one picture sequence starting from an I frame up to the next I frame (e.g. IBBPBBPBB).

Mapping to a multimedia transport system

In order to be efficient, such a mapping should be supported by a transport system capable of transferring both constant and variable bit rate streams. In case the latter is not possible, either (inefficient) worst case reservation or (time consuming) traffic shaping has to be performed. Below, we describe the mapping of UCL parameters onto parameters defined for the Tenet Protocol Suite [BFM$^+$94]. Tenet allows QoS representation of a variable bit rate data stream using the following parameters: minimum inter-message time, average inter-message time, averaging interval and maximum message size. In contrast to UCL description, which is application processing oriented, this load description is network oriented. Tenet expects a fixed packet size and a variable packet rate, whereas UCL offers a variable data unit size, but a fixed rate corresponding to a more ‘natural’ description of periodic data streams.

The mapping from UCL to Tenet parameters is performed as follows:

maximum message size = *seg*

minimum inter-message time = $1/(\lceil peak/seg \rceil * rate)$[1]

[1] assuming corresponding temporal smoothing in the link

average inter-message time = $1/(\lceil avg/seg \rceil * rate)$

averaging interval = iv

where *seg* is currently a value derived experimentally and fixed for the transport system in advance in order to avoid reservation inefficiency (i.e. overbooking of resources).

Without going into detail we indicate that similar parameters are offered by other transport systems, e.g. XTPX [MMR93] or HeiTS [VHN92] (which incorporates ST-II [Topo90] as reservation protocol). Another reservation protocol, RSVP [ZDE+93], defines no communication load parameters so far.

4 Negotiation Protocols and Protocol Coupling

The last section introduced two abstraction levels for QoS. Current QoS oriented protocols are defined for the transport or network level. Their primary purpose is to provide for resource reservation along connections between computing endsystems. Protocols have been developed to cover point-to-point and point-to-multipoint connections (Tenet, XTPX, ST-II), while RSVP realizes a concept for interconnecting m sources with n sinks.

Except for RSVP, these protocols imply that data sent at source sides is delivered unchanged (i.e. unprocessed) at receiving sides. RSVP allows for a restricted form of QoS scaling for receivers by employing filters at the network level. In all cases, QoS representation adheres to the transport or network level only. None of the protocols offers support for negotiation between transport connection users beyond transparent user data transfer.

Distributed multimedia applications feature further requirements. First, if resource reservation were to be performed in a complete end-to-end manner, it has to include all application components and intermediate transport connections influencing a data stream. For applications implying processing hops between source and sink components (Figure 1), neither of the mentioned protocols is applicable.

Second, even if no resource reservation is to be performed (or only in topology parts), QoS negotiation is still required to regulate interworking between interconnected components and links. The reason is that QoS support by components may already be limited by their implementation and that any limited QoS support influences interconnected components. Finally, as motivated in Section 3, QoS negotiation at the application level cannot be carried out in terms of transport level parameters.

The example of Figure 1 shall illustrate these requirements. The possibility of negotiation shall be given with each of the involved components being able to support various picture sizes. The components shall be such that it has to be ensured that all components handle the same picture size. For instance, the mixer component shall require equality of picture sizes at its two input ports. A limited resource availability of any component or intermediate link affects QoS to be provided by all other components or links. In order to regulate QoS according to such dependencies a protocol is required encompassing the whole application topology.

These requirements motivate the approach taken by *Cinema* to define, in analogy to QoS levels, two protocols aiming at QoS negotiation and resource reservation at different levels. At the transport level, existing protocols are taken as they are provided currently by external sources. At the application level, a new protocol type is required. Below, we develop such a protocol (termed negotiation and resource reservation protocol NRP) for the example application of Figure 1, in particular to indicate required couplings between protocol levels. We refer the reader to [BDFR95] for a protocol design covering more complex application topologies.

Negotiation Example

We assume that client QoS requests and negotiation relates to one media specific parameter, namely video picture size (assuming for instance, that the frame rate is fixed in advance to 25 frames/s). Note that the description could easily be extended to the case of multiple parameters, if priorities are specified (by the application client) indicating which parameter is to be reduced first, in case that QoS reductions become necessary.

As mentioned in Section 3, negotiation is triggered by an application client when issuing QoS requests with respect to the input ports at sink components. We assume the client requests for Dsp the best possible frame size from an acceptable range of (256x192 ... 512x384) (in pixels x pixels). At this point, NRP is triggered.

NRP is carried out by a corresponding protocol engine (PE) in three phases during which (in phase 1) the AFS available at the sink port is propagated towards the source components for resource reservation, (in phase 2) AFS from source ports are propagated back to the sink to prepare resource relaxation, and (in phase 3) resource reservations are relaxed while propagating the AFS towards sources again. We describe the phases below, first skipping the descriptions concerning negotiation inside links.

Phase 1 starts with the PE using the client QoS request to furnish initial AFS for the input port of Dsp: AFS(256x192 ... 512x384). Dsp is invoked with the AFS and responds to the PE with an unchanged AFS, thus indicating that it could reserve resources for the most demanding frame size. Next, the PE invokes the link between the mixer and the display. Both sending and receiving side of the link are invoked as motivated later. Assuming that the link is able to support all requested frame sizes, the link returns an unchanged AFS. The PE passes the AFS to the mixer by invoking it. Assuming that the mixer can support all frame sizes, two unchanged AFS are returned to the PE, one for each of the two input ports of the mixer.

The PE invokes the two links leading to the mixer input ports. We assume the link from Cam1 to the mixer can reserve for the best of requested frame sizes and returns an unchanged AFS. The PE invokes Cam1 with this AFS and receives it back unchanged. For the link from Cam2 the sequence of actions is similar. However, we assume that the link can support only a reduced frame size range and returns a reduced AFS(256x192 ... 384x288) to the PE. The PE invokes

Cam2 with this AFS and receives it back unchanged assuming that Cam2 can reserve correspondingly.

The first NRP phase ends upon reaching all source components. The second phases serves merely to match media specific QoS between mixer input ports. In consequence, links are bypassed during this phase. The phase starts with the AFS obtained from Cam1 and Cam2 being propagated by the PE to the mixer. The mixer matches the two AFS QoS ranges into a single range (256x192 ... 384x288) and returns it in its AFS (for the output port) to the PE. The PE delivers the AFS to the sink component Dsp. Upon invocation, Dsp installs the final video frame size to 384x288 and relaxes resource reservation accordingly. This last step marks already the begin of the third phase. The rest of this phase is similar to the first phase, except that the PE invokes resource relaxation methods of components and links instead of reservation methods. Upon reaching all source components, NRP is completed and the reservation session initiated by the client is set up.

Role of Links - Protocol Coupling

We explain here the effect of link invocations identified in the example above. The discussion shall highlight issues resulting from coupling QoS levels as well as their corresponding protocols. Three aspects are addressed: (a) the way how a link interacts with the PE and the transport system for connection set-up, (b) how a link does the mapping between media specific and transport level parameters and (c) how a link is to react in case of resource shortages in a transport system.

Interaction between the (NRP) PE and link objects was designed to offer to the PE a uniform view towards components and link objects. For this, link objects are made visible by decomposing them into receiver and sender side link objects denoted as L_s resp. L_r. Each of these objects is treated by the PE like a component with one output and one input port. Figure 3 depicts the sequence of actions for invoking links for resource reservation. Since the NRP phase proceeds from sinks to sources, the PE invokes first the receiver side link object L_r (arrow 1) receiving back a result AFS (2) just as when interacting with a component. Next, the PE invokes the sending side of the link L_s (3) from which it also receives a result AFS (7). Besides ensuring this uniform view of the PE, the approach detaches the PE from any specific connection establishment schemes used inside the link.

To demonstrate this feature, we assume that inside the link of Figure 3 a transport system is employed for which connection set-up initiation and completion has to be done at the sending side (e.g. HeiTS). When invoked by the PE (1), L_r responds with an unchanged AFS to the PE, since the receiving side does not initiate connection set-up (the PE is unaware of this). In contrast, L_s upon invocation may respond with a modified AFS (8) due to QoS reductions enforced inside the link.

Internally (to the link), L_s invokes the transport system for connection set-up (4), while L_r acknowledges the connection setup indication (5 and 6) without interacting further with the PE. This behaviour is sufficient, since L_r does not have to support negotiation between interconnected components using the link (given that the PE already does this). The set-up confirmation (7) is used by L_s to determine the response AFS for the PE (8).

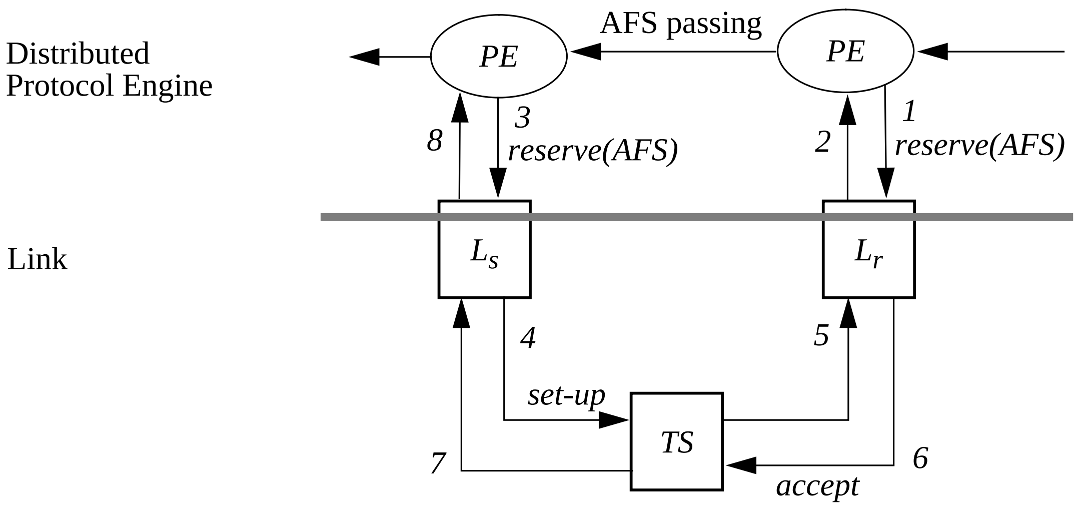

Figure 3: Protocol Coupling Example

L_s receives and delivers media specific parameters from/to the PE. In order to provide a transport flowspec (TFS) to the transport system, it first invokes the DownQM function (introduced in section 3) available at the port[1] to which L_s is connected. DownQM maps the QoS range of the AFS onto the UCL representation for the best media specific selection of this range. Given this UCL representation, L_s applies its internal mapping function to obtain the TFS. In case the transport system reserves successfully according to this TFS, the L_s returns the AFS unchanged to the PE.

In case resource reservation was not successful, L_s may be faced with two situations. In one case, the transport system indicates its QoS values (e.g. packet size and rate) for which reservation was possible. L_s uses these values to compare them with transport values derived for successively worse media specific value selections. For this, L_s invokes at its connected port a second mapping, NextQM, delivering two results: a shrunk QoS range excluding the highest possible media specific selection and UCL values corresponding to the new best media specific selection of the range. From the latter, L_s can again derive a reduced TFS. Repeating this process allows L_s to find the best possible media selection implying a TFS which is in line with the initial resource reservation of the transport system.[2] L_s returns the possibly shrunk media specific QoS range in its AFS response to the PE.

A second case is given for a transport system which only indicates resource reservation failure. In such a case, L_s can still calculate successively reduced TFS as described above. In addition, for each reduced TFS L_s has to invoke the transport system for connection set-up. This additional overhead is unavoidable here.

The interactions between the PE, L_s, L_r and the transport system are completely analogous for the relaxation phase of NRP. Beside different invocations (*relax* instead of *reserve*, *change*

1 Recall that ports are associated with streamtype objects

2 Given that the number of values for a media specific parameter is in most cases low, the implied overhead of this try-and-retry approach is low as well.

instead of *set-up*), a second difference is that relaxation never requires iterative use of QoS mappings.

5 End to end parameter handling

QoS requirements for multimedia applications concern, besides described media specific QoS, more generic parameters such as delay, jitter or loss-rate. These are usually referred to as end-to-end parameters, since they accumulate contributions of components and links between data stream source and sinks in order to yield QoS to the client. We currently work on concepts for including such generic QoS into NRP.

For delay, it is easy to see that NRP PE invocations can be extended to include a delay parameter filled in by links and components to indicate their contribution to delay. The PE would sum up these contributions to derive end-to-end values and compare them with client specified limits. For our negotiation example, this could be done during an additional fourth phase by summing up delay from sources towards sinks (Dsp). Accumulated values would have to be below the client's limit for Dsp (implying that the same limit applies to both paths from Cam1 and Cam2), otherwise the negotiation protocol would be stopped informing the client, that the delay cannot be kept.

The situation for jitter is similar, if component and link scheduling does not provide for jitter compensation. Otherwise, jitter values are accumulated prior to jitter compensating stages and compared with the tolerance limits of these stages. Considering data loss across many processing stages and communication links requires an adequate model describing their impact on data loss. Both number of stages and implied different abstraction levels make this a challenging task. The definition of such a model for *Cinema* is for further research.

6 Related Work

Media specific quality issues have been considered in restricted contexts. IMA MSS [IMA93] introduces the concept of media formats independent of QoS parameters. An IMA MSS format defines the encoding used for a specific medium. Format selection is done considering two adjacent component ports only, an end-to-end approach for applications consisting of many (more than two) processing component stages is not defined.

Several schemes have been developed for defining the mapping between media specific and transport level parameters ([KrLi94], [BDF^{+}94], [SMHW95], [NaSm95]). [BDF^{+}94] uses a management information base to map application specific QoS to transport system QoS. [SMHW95] uses a QoS manager tool to do the mapping. The QoS manager tool is called with the selected kind of media encoding and the desired QoS class and returns the transport level (XTPX-based) QoS data structure. [NaSm95] introduces a QoS Broker which negotiates end-to-end QoS at application level for client/server settings only. Application level negotiation in the sense described in this paper is not considered in either case and neither of these approaches considers topologies beyond client/server.

7 Conclusions

Provision of QoS by distributed multimedia applications is subject to QoS constraints. Application clients can specify QoS requests as QoS ranges in order to allow best possible QoS selection, while avoiding a time-consuming and possibly complex try-and-retry approach with fixed QoS-value requests. Application level negotiation differs from transport level negotiation. It requires a separation of QoS abstraction levels for media specific and transport level characteristics and definition of mappings between them. Application clients have to be offered a media specific QoS view.

Application level negotiation has to encompass all components and links, even if resource reservation is not performed or restricted to some application parts only. Application level protocol design can be kept independent from transport system design by providing link objects encapsulating transport system peculiarities. Coupling between application and transport level negotiation can be hidden from any of the two levels and confined to the link objects. This applies both to QoS mappings and connection set-up schemes.

Besides motivating introduced concepts, the paper demonstrated their feasibility for an example negotiation protocol. The QoS framework presented is currently introduced into our CINEMA system implementation. It prepares the implementation of an application level protocol which is applicable to a wide class of application topologies.

8 References

[BCA+92] G. Blair et al. An Integrated Platform and Computational Model for Open Distributed Multimedia Applications. In *3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, p. 209–222, 11 1992.

[BDFR95] I. Barth, G. Dermler, W. Fiederer, K. Rothermel. A Negotiation and Resource Reservation Protocol (NRP) for Configurable Multimedia Applications. *Technical Report submitted for publication*.

[BDF+94] L. Besse et al. Towards an Architecture for Distributed Multimedia Applications Support. In *International Conference on Multimedia Computing and Systems*, p. 164–172, 5 1994.

[BFM+94] A. Banerjea et al. The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences. *University of California at Berkeley and The International Computer Science Institute, TR-94-059, available via http://tenet.berkeley.edu/tenet-papers.html*, 11 1994.

[Gall91] Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM*, 34(4):46–58, 4 1991.

[IMA93] HP Company and IBM Corporation and SunSoft Inc. *Multimedia System Services, Version 1.0, available via ftp from ibminet.awdpa.ibm.com*, 7 1993.

[JPEG93] ISO IETC JTC 1. Information Technology - Digital Compression and Coding of Continuous-Tone Still Images. *International Standard ISO/IEC IS 10918*, 1993.

[KHSM95] T. Käppner et al. Eine verteilte Entwicklungs- und Laufzeitumgebung für multimediale Anwendungen. In *Proceedings of KiVS'95*, p. 76–86, 1995.

[MKSD90] J. Magee et al. An Overview of the REX Software Architecture. *2nd IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, 10 1990.

[MMR93] B. Metzler et al. Specification of the Broadband Transport Protocol XTPX. *Technical University of Berlin, available via http://www.prz.tu-berlin.de/docs/html/prot/xtpx.html*, 2 1993.

[NaSm95] K. Nahrstedt, J. Smith. The QOS Broker. *IEEE Multimedia, Vol. 2, No. 1*, p. 53-67, Spring 1995.

[RBH94] K. Rothermel, I. Barth, T. Helbig. In *Architecture and Protocols for High-Speed Networks*, Chapter CINEMA - An Architecture for Distributed Multimedia Applications, p. 253–271. Kluwer Academic Publishers, 1994.

[SMHW95] A. Schill et al. A Quality of Service Abstraction Tool for Advanced Distributed Applications. In *International Conference on Open Distributed Processing*, 2 1995.

[Topo90] C. Topolcic. Experimental Internet Stream Protocol, Version 2 (ST-II). *RFC 1190*, 10 1990.

[VHN92] C. Vogt et al. HeiRAT: The Heidelberg Resource and Administration Technique, Design Philosophy and Goals. *Technical Report No. 43.9213*, ENC European Networking Center, 1992.

[ZDE$^+$93] L. Zhang et al. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, p. 8–18, 9 1993.

Universität Stuttgart
Fakultät Informatik

A Framework for Negotiable Quality of Service in Distributed Multimedia Systems

Gabriel Dermler, Walter Fiederer, Ingo Barth, Kurt Rothermel

CR-Klassifikation: C.2.4, D.4.7, H.5.1

**A Framework for
Negotiable Quality of Service in
Distributed Multimedia Systems**

G. Dermler, W. Fiederer, I. Barth, K. Rothermel

Fakultätsbericht 10/1995
Technical Report
Dezember 1995

Fakultät Informatik
Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20 - 22
D-70565 Stuttgart

Abstract

Distributed multimedia applications offer to clients degrees of freedom for selecting Quality of Service (QoS). This paper describes a framework for application level negotiation as a means to regulate client requested QoS under QoS constraints imposed on an application. A QoS architecture is motivated interrelating levels for media specific and transport level QoS handling. The necessity of corresponding protocol levels is explained. The coupling between these levels is demonstrated for a concrete protocol developed for an example application. The embedding of end-to-end delay into the protocol is discussed.

1 Introduction

Distributed multimedia applications are employed to generate, process and comsume (e.g. present) continuous (e.g. audio, video) data streams across distributed locations. An application client has a service oriented view towards QoS provided by the application. It specifies QoS with respect to consumed data (e.g. presented video frame size and rate) and expects the application to agree on a specific QoS selection. Providing such an agreement requires a process orchestrating client QoS requests and QoS constraints imposed on the application (for instance) by restricted resource availability. We refer to this process as application level QoS negotiation.

So far, description of negotiation is in analogy to negotiation at transport level as currently pursued for various transport system designs (e.g. [VHN92], [MMR93]). However, a number of features make QoS selection at the application level a different and potentially much more complex task. It is the goal of this paper to elaborate these features and to interrelate them in an overall framework allowing the development of negotiation protocols for distributed multimedia applications. In particular, such a protocol is developed for an example application demonstrating in this way the feasibility of introduced concepts.

The paper is structured as follows. In Section 2, we introduce an abstract model for constructing distributed multimedia applications. We use this model in Section 3 to motivate and position two distinct QoS levels required during negotiation and describe the relative position of application clients issuing QoS requests. The relationship between the QoS levels is completed by appropriate mapping functions.

Section 4 motivates the necessity of two protocol levels referring to the existence of different QoS levels and application topologies. It describes the role of each level by introducing a negotiation protocol in the context of an application example, in particular highlighting the coupling required between the protocol levels. Section 5 indicates how issues concerning end-to-end parameters such as delay can be integrated into the introduced protocols. Section 6 summarizes important conclusions and gives a status report on current implementation work.

2 Application Model

In this section we introduce a concept for constructing distributed multimedia applications. Similar concepts are pursued by various research groups ([KHSM95], [BCA$^+$92], [MKSD90]) including the group defining IMA MMS [IMA93]. We describe here the terminology used for our *Cinema* development platform. For a detailed description of *Cinema* refer to [RBH94].

Cinema allows a client to compose an application out of components and links. Components encapsulate processing of multimedia data, e.g. for generating, presenting or manipulating data. To provide a uniform data access point for the components, ports are used that deliver data units to the component (input port) or take the data units from the component (output port). A client constructs an application by specifying a topology of components interconnected via links. A link provides an abstraction from underlying communication mechanisms which may be used to perform the transport of data units. Figure 1 shows an example topology composed of two video components generating two video streams which are mixed by a video mixer and displayed by a 3D monitor component.

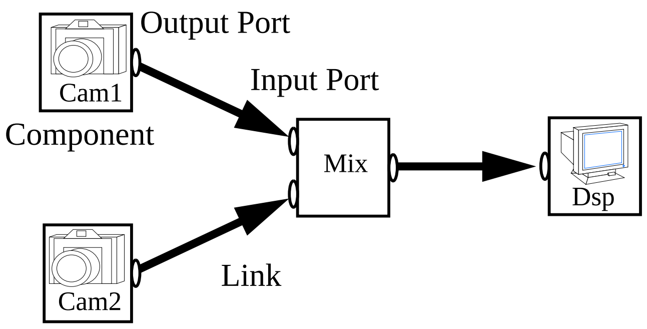

Figure 1: Example of an application topology

Before using an application, a client has to indicate desired QoS to *Cinema*. For this, *Cinema* offers the concept of a session. A session is the unit of resource reservation allowing the client to specify the application to be instantiated and the QoS expected with respect to output generated by sink components. The goal of negotiation is then to settle on the best possible QoS complying with client requests under two kinds of restrictions: limited functional capabilities of components (e.g.: design enforcing processing of video frames only up to a maximal size) and limited resource availability for links and components.

3 QoS Architecture

The necessity of different QoS levels arises from the type of data encountered at component ports. Such ports are employed to feed components for processing and to obtain processed data for distribution to subsequent components. Internal processing of a component requires data in a formatted form. For instance, a video component may expect data as a sequence of frames with a certain picture size and picture rate. For each existing component port, a description has to exist in order to indicate externally the characteristics of data expected. Such information is a prerequisite to negotiation, for instance, checking whether two component ports can be interlinked in a compatible fashion.

Characteristics of processed and communicated data are media specific. For video, description parameters frequently include video picture size, picture rate, compression scheme and ratio. For audio, similar parameters may be used including sample rate, sample size and encoding scheme. Media specific differences are being given at least in form of different implied dimensions and value ranges. In terms of abstraction levels, such parameters are media specific since they have no unambiguous transport level representation. For instance, a pair of (video frame size, frame rate) values can be mapped in various ways onto a pair of (packet size, packet rate) values.

From the above, it is obvious that any distributed multimedia system has to include two levels of QoS: (a) a “higher” media specific level describing the communication load requested by interconnected components and (b) a “lower” level describing the communication load requested at the service interface of transport systems. Figure 2 shows the positioning of QoS levels in *Cinema*. At the level of components and links, media specific parameters are exchanged in so-called application flowspecs (AFS), while transport level parameters are employed only

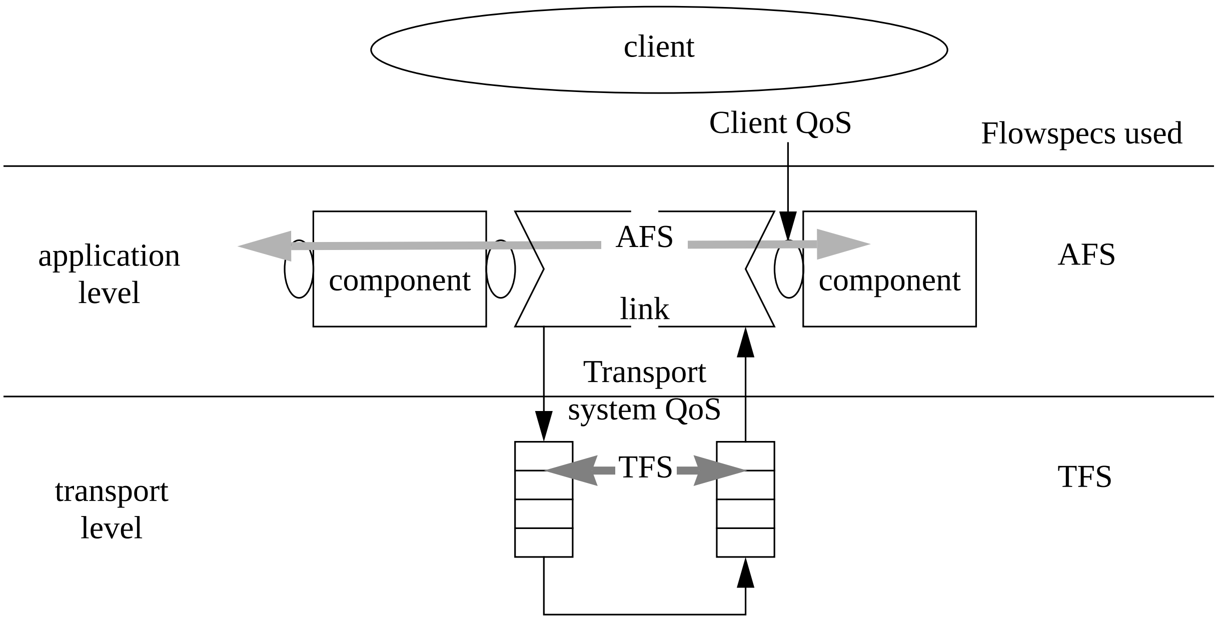

Figure 2: QoS Architecture

inside links within transport flowspecs (TFS). The figure shows that a client specifies its QoS requests with respect to the input ports of sink components, i.e. in terms of media specific parameters expected there. The following discussion motivates the architecture.

Cinema offers a basic level for QoS specification. The client itself may be given as a program converting media specific parameters into higher level abstractions for (human) end users (e.g. “high”, “mid” and “low quality” video). Client requests are related to sink component ports, assuming that the client has knowledge about sensed output QoS and corresponding media specific QoS required as input to sink components.

As mentioned earlier, component design involves the selection of a media specific parameter set to be used by a component at a certain port. Since we do not expect such parameter sets to be defined too often, we make a specific set of parameters available in form of streamtype objects including, besides the media specific parameter list, the possible value ranges for each parameter. In consequence, a component designer has to associate with each component port the streamtype to be used making all related information available at the port.

Components are designed to be reusable in various application topologies.[1] In particular, component design is kept independent of data transport characteristics. Differences exist here since current transport systems expect and handle transport level QoS differently. Link objects abstract from such peculiarities and offer a uniform external view. As shown in Figure 2, a link object receives media specific parameters at its interface from which it derives the QoS representation needed by the encapsulated transport system. But in order to reduce the number of implemented link types, link implementation is to be kept independent of any media specific characteristics. These requirements lead to the following structuring of the mapping from media specific to transport system parameters.

A first mapping, termed DownQM, is provided by streamtype objects at component ports.[2] DownQM maps media specific parameter values (e.g. video frame size and rate) onto a so-called uniform communication load representation (UCL) defined by *Cinema*. UCL is positioned at the transport level, its purpose is to abstract from (QoS) interface peculiarities of a specific transport system. Given DownQM, a link implementation has to include only a function mapping a UCL description onto the QoS representation expected by the specific transport system. A link object handles media specific parameters since it has access to both described functions. It can use its internal mapping and can call DownQM since it has access to component ports connected to the link. Given this structuring, a new link implementation is required only if it encapsulates a new transport system (e.g. for radio communication).

Mapping to UCL

The selection of UCL parameters was based on parameter selections pursued by current multimedia transport systems. It was guided by the requirement to describe application generated load both for compressed and uncompressed media streams as shaped for instance by MPEG [Gall91] or JPEG [JPEG93] compression. Optimal selection of parameters for load description

[1] Two interconnected component ports have to be associated with the same streamtype.
[2] Another function, NextQM, is also provided by a streamtype object (see Section 4 for motivation).

is a continuing research issue. In particular, a future standardized transport QoS description may be used for UCL, should one emerge. The following UCL parameters are currently defined:

- peak data unit size (*peak*)
- average data unit size (*avg*)
- data unit rate (*rate*)
- average interval (*iv*)

For uncompressed streams the use of these parameters is straightforward. For audio, a UCL data unit would encompass the amount of audio processed by a component at once (e.g. 40 ms of Audio. *Peak* and *avg* would be both set to a corresponding number of bytes, while *rate* would be fixed to 25 data units/s. For uncompressed video, a UCL data unit would encompass the amount of video processed at once, typically a picture size, while *rate* would be set to the video frame rate. However, different mapping schemes could be defined.[1].

c. quality	192 x 144	256 x 192	384 x 288	512 x 384	768 x 576
10	4.1	6.0	8.6	16.0	29.2
25	4.7	6.6	9.4	17.5	31.8
50	5.9	8.4	12.2	22.1	40.0
70	7.8	10.4	15.4	28.6	58.1
85	9.8	15.2	24.1	45.0	80.0
95	16.6	26.7	44.8	69.8	131.6

Table 1: UCL data unit sizes in KB

For JPEG compressed video, we have to include the compression effect. We took an empiric approach to measure the size of a UCL data unit resulting from a picture of a given size, color depth (fixed to 24 bit) and compression quality (given as a number between 1 to 100). Table 1 contains some measurement results. Calculating UCL values then amounts to:

[1] Recall that DownQM encapsulates this mapping.

peak = *avg* = table entry for desired picture size and compression quality

rate = f_rate

The mapping may be defined differently, for instance by using *peak* and *avg* to reflect variations of compressed picture sizes over a longer time interval. Our UCL description scheme does allow for this, though further experimentation is required for deriving corresponding (*peak*, *avg*) value pairs as table entries. This latter approach is currently taken for MPEG compressed streams, where we measure (*peak*, *avg*) pairs. *Peak* denotes the maximum compressed picture size encountered (typically an I coded frame), while *avg* is an average value for an average interval which encompasses at least one picture sequence starting from an I frame up to the next I frame (e.g. IBBPBBPBB).

Mapping to a multimedia transport system

In order to be efficient, such a mapping should be supported by a transport system capable of transferring both constant and variable bit rate streams. In case the latter is not possible, either (inefficient) worst case reservation or (time consuming) traffic shaping has to be performed. Below, we describe the mapping of UCL parameters onto parameters defined for the Tenet Protocol Suite [BFM$^+$94]. Tenet allows QoS representation of a variable bit rate data stream using the following parameters: minimum inter-message time, average inter-message time, averaging interval and maximum message size. In contrast to UCL description, which is application processing oriented, this load description is network oriented. Tenet expects a fixed packet size and a variable packet rate, whereas UCL offers a variable data unit size, but a fixed rate corresponding to a more ‘natural’ description of periodic data streams.

The mapping from UCL to Tenet parameters is performed as follows:

maximum message size = *seg*

minimum inter-message time = $1/(\lceil peak/seg \rceil * rate)$[1]

[1] assuming corresponding temporal smoothing in the link

average inter-message time = $1/(\lceil avg/seg \rceil * rate)$

averaging interval = iv

where *seg* is currently a value derived experimentally and fixed for the transport system in advance in order to avoid reservation inefficiency (i.e. overbooking of resources).

Without going into detail we indicate that similar parameters are offered by other transport systems, e.g. XTPX [MMR93] or HeiTS [VHN92] (which incorporates ST-II [Topo90] as reservation protocol). Another reservation protocol, RSVP [ZDE$^+$93], defines no communication load parameters so far.

4 Negotiation Protocols and Protocol Coupling

The last section introduced two abstraction levels for QoS. Current QoS oriented protocols are defined for the transport or network level. Their primary purpose is to provide for resource reservation along connections between computing endsystems. Protocols have been developed to cover point-to-point and point-to-multipoint connections (Tenet, XTPX, ST-II), while RSVP realizes a concept for interconnecting m sources with n sinks.

Except for RSVP, these protocols imply that data sent at source sides is delivered unchanged (i.e. unprocessed) at receiving sides. RSVP allows for a restricted form of QoS scaling for receivers by employing filters at the network level. In all cases, QoS representation adheres to the transport or network level only. None of the protocols offers support for negotiation between transport connection users beyond transparent user data transfer.

Distributed multimedia applications feature further requirements. First, if resource reservation were to be performed in a complete end-to-end manner, it has to include all application components and intermediate transport connections influencing a data stream. For applications implying processing hops between source and sink components (Figure 1), neither of the mentioned protocols is applicable.

Second, even if no resource reservation is to be performed (or only in topology parts), QoS negotiation is still required to regulate interworking between interconnected components and links. The reason is that QoS support by components may already be limited by their implementation and that any limited QoS support influences interconnected components. Finally, as motivated in Section 3, QoS negotiation at the application level cannot be carried out in terms of transport level parameters.

The example of Figure 1 shall illustrate these requirements. The possibility of negotiation shall be given with each of the involved components being able to support various picture sizes. The components shall be such that it has to be ensured that all components handle the same picture size. For instance, the mixer component shall require equality of picture sizes at its two input ports. A limited resource availability of any component or intermediate link affects QoS to be provided by all other components or links. In order to regulate QoS according to such dependencies a protocol is required encompassing the whole application topology.

These requirements motivate the approach taken by *Cinema* to define, in analogy to QoS levels, two protocols aiming at QoS negotiation and resource reservation at different levels. At the transport level, existing protocols are taken as they are provided currently by external sources. At the application level, a new protocol type is required. Below, we develop such a protocol (termed negotiation and resource reservation protocol NRP) for the example application of Figure 1, in particular to indicate required couplings between protocol levels. We refer the reader to [BDFR95] for a protocol design covering more complex application topologies.

Negotiation Example

We assume that client QoS requests and negotiation relates to one media specific parameter, namely video picture size (assuming for instance, that the frame rate is fixed in advance to 25 frames/s). Note that the description could easily be extended to the case of multiple parameters, if priorities are specified (by the application client) indicating which parameter is to be reduced first, in case that QoS reductions become necessary.

As mentioned in Section 3, negotiation is triggered by an application client when issuing QoS requests with respect to the input ports at sink components. We assume the client requests for Dsp the best possible frame size from an acceptable range of (256x192 ... 512x384) (in pixels x pixels). At this point, NRP is triggered.

NRP is carried out by a corresponding protocol engine (PE) in three phases during which (in phase 1) the AFS available at the sink port is propagated towards the source components for resource reservation, (in phase 2) AFS from source ports are propagated back to the sink to prepare resource relaxation, and (in phase 3) resource reservations are relaxed while propagating the AFS towards sources again. We describe the phases below, first skipping the descriptions concerning negotiation inside links.

Phase 1 starts with the PE using the client QoS request to furnish initial AFS for the input port of Dsp: AFS(256x192 ... 512x384). Dsp is invoked with the AFS and responds to the PE with an unchanged AFS, thus indicating that it could reserve resources for the most demanding frame size. Next, the PE invokes the link between the mixer and the display. Both sending and receiving side of the link are invoked as motivated later. Assuming that the link is able to support all requested frame sizes, the link returns an unchanged AFS. The PE passes the AFS to the mixer by invoking it. Assuming that the mixer can support all frame sizes, two unchanged AFS are returned to the PE, one for each of the two input ports of the mixer.

The PE invokes the two links leading to the mixer input ports. We assume the link from Cam1 to the mixer can reserve for the best of requested frame sizes and returns an unchanged AFS. The PE invokes Cam1 with this AFS and receives it back unchanged. For the link from Cam2 the sequence of actions is similar. However, we assume that the link can support only a reduced frame size range and returns a reduced AFS(256x192 ... 384x288) to the PE. The PE invokes

Cam2 with this AFS and receives it back unchanged assuming that Cam2 can reserve correspondingly.

The first NRP phase ends upon reaching all source components. The second phases serves merely to match media specific QoS between mixer input ports. In consequence, links are bypassed during this phase. The phase starts with the AFS obtained from Cam1 and Cam2 being propagated by the PE to the mixer. The mixer matches the two AFS QoS ranges into a single range (256x192 ... 384x288) and returns it in its AFS (for the output port) to the PE. The PE delivers the AFS to the sink component Dsp. Upon invocation, Dsp installs the final video frame size to 384x288 and relaxes resource reservation accordingly. This last step marks already the begin of the third phase. The rest of this phase is similar to the first phase, except that the PE invokes resource relaxation methods of components and links instead of reservation methods. Upon reaching all source components, NRP is completed and the reservation session initiated by the client is set up.

Role of Links - Protocol Coupling

We explain here the effect of link invocations identified in the example above. The discussion shall highlight issues resulting from coupling QoS levels as well as their corresponding protocols. Three aspects are addressed: (a) the way how a link interacts with the PE and the transport system for connection set-up, (b) how a link does the mapping between media specific and transport level parameters and (c) how a link is to react in case of resource shortages in a transport system.

Interaction between the (NRP) PE and link objects was designed to offer to the PE a uniform view towards components and link objects. For this, link objects are made visible by decomposing them into receiver and sender side link objects denoted as L_s resp. L_r. Each of these objects is treated by the PE like a component with one output and one input port. Figure 3 depicts the sequence of actions for invoking links for resource reservation. Since the NRP phase proceeds from sinks to sources, the PE invokes first the receiver side link object L_r (arrow 1) receiving back a result AFS (2) just as when interacting with a component. Next, the PE invokes the sending side of the link L_s (3) from which it also receives a result AFS (7). Besides ensuring this uniform view of the PE, the approach detaches the PE from any specific connection establishment schemes used inside the link.

To demonstrate this feature, we assume that inside the link of Figure 3 a transport system is employed for which connection set-up initiation and completion has to be done at the sending side (e.g. HeiTS). When invoked by the PE (1), L_r responds with an unchanged AFS to the PE, since the receiving side does not initiate connection set-up (the PE is unaware of this). In contrast, L_s upon invocation may respond with a modified AFS (8) due to QoS reductions enforced inside the link.

Internally (to the link), L_s invokes the transport system for connection set-up (4), while L_r acknowledges the connection setup indication (5 and 6) without interacting further with the PE. This behaviour is sufficient, since L_r does not have to support negotiation between interconnected components using the link (given that the PE already does this). The set-up confirmation (7) is used by L_s to determine the response AFS for the PE (8).

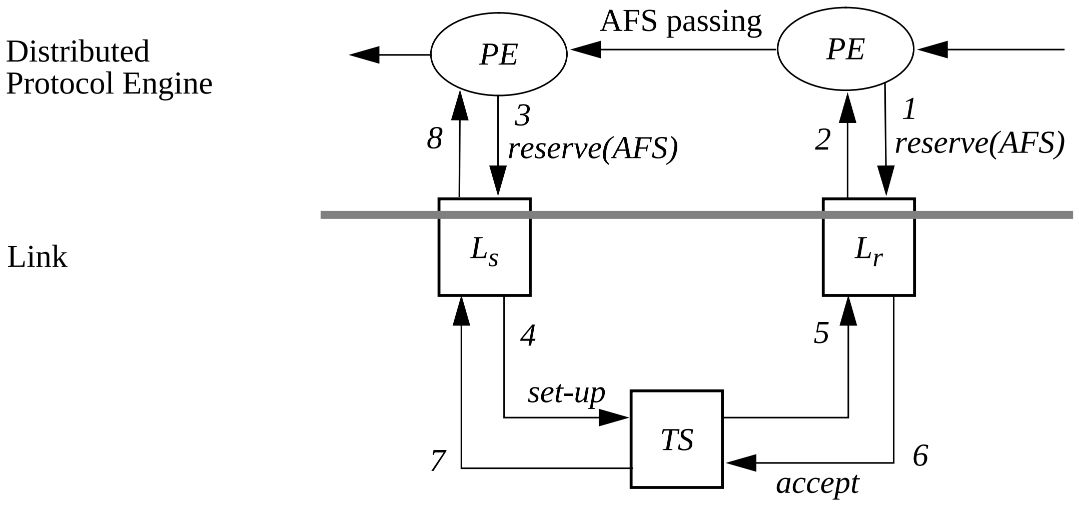

Figure 3: Protocol Coupling Example

L_S receives and delivers media specific parameters from/to the PE. In order to provide a transport flowspec (TFS) to the transport system, it first invokes the DownQM function (introduced in section 3) available at the port[1] to which L_S is connected. DownQM maps the QoS range of the AFS onto the UCL representation for the best media specific selection of this range. Given this UCL representation, L_S applies its internal mapping function to obtain the TFS. In case the transport system reserves successfully according to this TFS, the L_S returns the AFS unchanged to the PE.

In case resource reservation was not successful, L_S may be faced with two situations. In one case, the transport system indicates its QoS values (e.g. packet size and rate) for which reservation was possible. L_S uses these values to compare them with transport values derived for successively worse media specific value selections. For this, L_S invokes at its connected port a second mapping, NextQM, delivering two results: a shrunk QoS range excluding the highest possible media specific selection and UCL values corresponding to the new best media specific selection of the range. From the latter, L_S can again derive a reduced TFS. Repeating this process allows L_S to find the best possible media selection implying a TFS which is in line with the initial resource reservation of the transport system.[2] L_S returns the possibly shrunk media specific QoS range in its AFS response to the PE.

A second case is given for a transport system which only indicates resource reservation failure. In such a case, L_S can still calculate successively reduced TFS as described above. In addition, for each reduced TFS L_S has to invoke the transport system for connection set-up. This additional overhead is unavoidable here.

The interactions between the PE, L_S, L_r and the transport system are completely analogous for the relaxation phase of NRP. Beside different invocations (*relax* instead of *reserve*, *change*

[1] Recall that ports are associated with streamtype objects
[2] Given that the number of values for a media specific parameter is in most cases low, the implied overhead of this try-and-retry approach is low as well.

instead of *set-up*), a second difference is that relaxation never requires iterative use of QoS mappings.

5 End to end parameter handling

QoS requirements for multimedia applications concern, besides described media specific QoS, more generic parameters such as delay, jitter or loss-rate. These are usually referred to as end-to-end parameters, since they accumulate contributions of components and links between data stream source and sinks in order to yield QoS to the client. We currently work on concepts for including such generic QoS into NRP.

For delay, it is easy to see that NRP PE invocations can be extended to include a delay parameter filled in by links and components to indicate their contribution to delay. The PE would sum up these contributions to derive end-to-end values and compare them with client specified limits. For our negotiation example, this could be done during an additional fourth phase by summing up delay from sources towards sinks (Dsp). Accumulated values would have to be below the client's limit for Dsp (implying that the same limit applies to both paths from Cam1 and Cam2), otherwise the negotiation protocol would be stopped informing the client, that the delay cannot be kept.

The situation for jitter is similar, if component and link scheduling does not provide for jitter compensation. Otherwise, jitter values are accumulated prior to jitter compensating stages and compared with the tolerance limits of these stages. Considering data loss across many processing stages and communication links requires an adequate model describing their impact on data loss. Both number of stages and implied different abstraction levels make this a challenging task. The definition of such a model for *Cinema* is for further research.

6 Related Work

Media specific quality issues have been considered in restricted contexts. IMA MSS [IMA93] introduces the concept of media formats independent of QoS parameters. An IMA MSS format defines the encoding used for a specific medium. Format selection is done considering two adjacent component ports only, an end-to-end approach for applications consisting of many (more than two) processing component stages is not defined.

Several schemes have been developed for defining the mapping between media specific and transport level parameters ([KrLi94], [BDF+94], [SMHW95], [NaSm95]). [BDF+94] uses a management information base to map application specific QoS to transport system QoS. [SMHW95] uses a QoS manager tool to do the mapping. The QoS manager tool is called with the selected kind of media encoding and the desired QoS class and returns the transport level (XTPX-based) QoS data structure. [NaSm95] introduces a QoS Broker which negotiates end-to-end QoS at application level for client/server settings only. Application level negotiation in the sense described in this paper is not considered in either case and neither of these approaches considers topologies beyond client/server.

7 Conclusions

Provision of QoS by distributed multimedia applications is subject to QoS constraints. Application clients can specify QoS requests as QoS ranges in order to allow best possible QoS selection, while avoiding a time-consuming and possibly complex try-and-retry approach with fixed QoS-value requests. Application level negotiation differs from transport level negotiation. It requires a separation of QoS abstraction levels for media specific and transport level characteristics and definition of mappings between them. Application clients have to be offered a media specific QoS view.

Application level negotiation has to encompass all components and links, even if resource reservation is not performed or restricted to some application parts only. Application level protocol design can be kept independent from transport system design by providing link objects encapsulating transport system peculiarities. Coupling between application and transport level negotiation can be hidden from any of the two levels and confined to the link objects. This applies both to QoS mappings and connection set-up schemes.

Besides motivating introduced concepts, the paper demonstrated their feasibility for an example negotiation protocol. The QoS framework presented is currently introduced into our CINEMA system implementation. It prepares the implementation of an application level protocol which is applicable to a wide class of application topologies.

8 References

[BCA+92] G. Blair et al. An Integrated Platform and Computational Model for Open Distributed Multimedia Applications. In *3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, p. 209–222, 11 1992.

[BDFR95] I. Barth, G. Dermler, W. Fiederer, K. Rothermel. A Negotiation and Resource Reservation Protocol (NRP) for Configurable Multimedia Applications. *Technical Report submitted for publication*.

[BDF+94] L. Besse et al. Towards an Architecture for Distributed Multimedia Applications Support. In *International Conference on Multimedia Computing and Systems*, p. 164–172, 5 1994.

[BFM+94] A. Banerjea et al. The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences. *University of California at Berkeley and The International Computer Science Institute, TR-94-059, available via http://tenet.berkeley.edu/tenet-papers.html*, 11 1994.

[Gall91] Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM*, 34(4):46–58, 4 1991.

[IMA93] HP Company and IBM Corporation and SunSoft Inc. *Multimedia System Services, Version 1.0, available via ftp from ibminet.awdpa.ibm.com*, 7 1993.

[JPEG93] ISO IETC JTC 1. Information Technology - Digital Compression and Coding of Continuous-Tone Still Images. *International Standard ISO/IEC IS 10918*, 1993.

[KHSM95] T. Käppner et al. Eine verteilte Entwicklungs- und Laufzeitumgebung für multimediale Anwendungen. In *Proceedings of KiVS'95*, p. 76–86, 1995.

[MKSD90] J. Magee et al. An Overview of the REX Software Architecture. *2nd IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, 10 1990.

[MMR93] B. Metzler et al. Specification of the Broadband Transport Protocol XTPX. *Technical University of Berlin, available via http://www.prz.tu-berlin.de/docs/html/prot/xtpx.html*, 2 1993.

[NaSm95] K. Nahrstedt, J. Smith. The QOS Broker. *IEEE Multimedia*, *Vol. 2, No. 1*, p. 53-67, Spring 1995.

[RBH94] K. Rothermel, I. Barth, T. Helbig. In *Architecture and Protocols for High-Speed Networks*, Chapter CINEMA - An Architecture for Distributed Multimedia Applications, p. 253–271. Kluwer Academic Publishers, 1994.

[SMHW95] A. Schill et al. A Quality of Service Abstraction Tool for Advanced Distributed Applications. In *International Conference on Open Distributed Processing*, 2 1995.

[Topo90] C. Topolcic. Experimental Internet Stream Protocol, Version 2 (ST-II). *RFC 1190*, 10 1990.

[VHN92] C. Vogt et al. HeiRAT: The Heidelberg Resource and Administration Technique, Design Philosophy and Goals. *Technical Report No. 43.9213*, ENC European Networking Center, 1992.

[ZDE$^+$93] L. Zhang et al. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, p. 8–18, 9 1993.