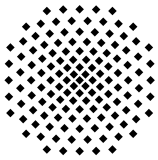
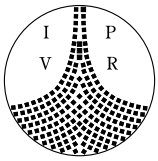


Universität Stuttgart
Fakultät Informatik



A Negotiation and Resource Reservation Protocol (NRP) for Distributed Multimedia Applications

Gabriel Dermler, Walter Fiederer, Ingo Barth, Kurt Rothermel

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

A Negotiation and Resource Reservation Protocol (NRP) for Distributed Multimedia Applications

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

Fakultätsbericht 11/1995
Technical Report
Dezember 1995

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

Abstract

Distributed multimedia applications require negotiation of quality of service (QoS) and resource reservation for distributed application parts and communication links. Negotiation of QoS is a balancing process between the QoS specified by a client, the resource availabilities of the distributed system and the functional capabilities of the distributed application. Negotiation requires application level QoS descriptions and an end-to-end view spanning the whole distributed application. NRP is an application level protocol meeting these requirements. NRP performs negotiation based on client specified QoS value ranges and given resource availability on endsystems and communication links. NRP is independent of application level QoS semantics. It allows QoS negotiation and resource reservation in three phases and supports a wide range of distributed application topologies.

1 Introduction

The promise of networked multimedia is to provide a multitude of attractive applications with considerable benefits to users in various areas. Multimedia retrieval services for video-on-demand, teleshopping or news services as well as teleconferencing are among the first applications demonstrating this value. Other applications such as distributed games, multimedia based virtual meeting spaces are examples for applications currently pursued as prototypes.

Networked multimedia applications offer a service oriented view to provide QoS. An application client specifies QoS requests 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 the transport level as currently pursued for various transport system designs (e.g. [VHN92], [MMR93]). However, a number of requirements exist at the application level making negotiation a different and potentially much more complex task than at the transport level. It is the goal of this paper both to elaborate these requirements and to develop a negotiation protocol coping with them.

The paper is structured as follows. In Section 2, we introduce an abstract model for describing multimedia applications as topologies of distributed components. Referring to this model we motivate in Section 3 requirements covering appropriate abstraction levels for negotiation and the negotiation complexity arising from heterogeneous receivers and different application topologies. In Section 4 we describe our negotiation and resource reservation (NRP) protocol. We start by indicating the scope of NRP and presenting it in the context of an example application topology. Then we present an architecture anchoring NRP with respect to a distributed application using NRP. The architecture is used to explain the three phases of NRP in detail.

Section 5 serves to discuss several features of NRP. First, we motivate the design of NRP phases. Then we indicate dimensions of expanding NRP. For one, we show problems which have to be overcome, if arbitrary application topologies are to be supported. Then we briefly indicate, how end-to-end parame-

ters such as delay and jitter can be included into NRP. In Section 6, we relate NRP to previous work. The paper concludes in Section 7 with a short summary and an outlook on future work.

2 Application Model

In this section we briefly 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* we refer to [RBH94].

CINEMA allows a client[1] 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 components, ports are used that deliver data units to the component (input port) or take 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 for a video conferencing scenario. The topology is composed of two video source components connected to a mixing component which provides a data stream multicasted to two sink components. Prior to each sink component a converter component is included.

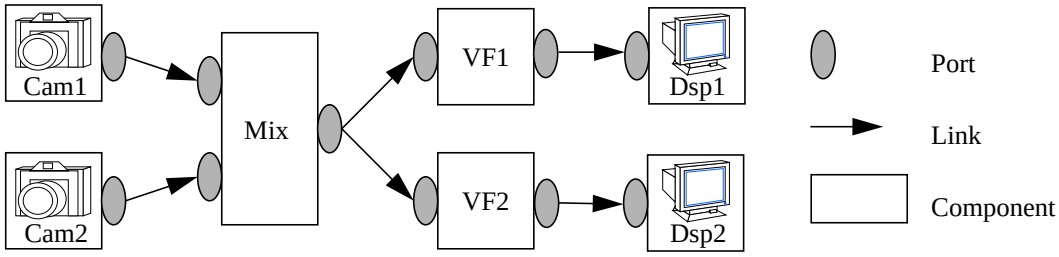


Figure 1 : Video Conferencing Scenario with Mixer and Multicast

Before using an application, a client has to indicate the desired QoS to *CINEMA*. *CINEMA* offers the concept of a session as the unit of application instantiation for client requested QoS. To define the quality of presentation at sinks a client specifies the QoS expected at sink component input ports. For instance, for the

[1] A client is a program entity using the services offered by *CINEMA*.

application of Figure 1, a client would specify the QoS expected at the ports of Dsp1 and Dsp2. The QoS can be specified independently for each sink component port. In consequence, the *CINEMA* session concept allows for "heterogeneous" QoS requests at sink components[1] reflecting in this way varying QoS requests of independent (human) endusers supported by a client (see below).

3 Negotiation Requirements

Abstraction levels

QoS negotiation for multimedia applications can be considered at various abstraction levels. At the highest level, endusers express their QoS requests to the presentation realized by an application at its sink components. Various QoS abstractions can be used at this level. It can be expected that **enduser oriented QoS** descriptions will prevail, e.g. "high, medium, low quality video".

At the lowest level, QoS descriptions serve to indicate resource demands. Consequently, QoS description is resource oriented and independent of media specific characteristics. For instance, in order to be able to set up a connection for video data transport, **generic transport level QoS** parameters (e.g. TSDU size and rate) are used which can be handled inside the transport system to reserve resources along the connection.

When negotiating QoS for an application topology, there is a need to transfer QoS information between components, in order to make sure that two interconnected components send (generate) and receive (consume) the same QoS. The nature of this QoS description is neither enduser nor resource oriented. The description has to reflect the ability of a component to support QoS at its (input and output) ports. Given that a component's internal processing is specific for the media (e.g. audio, video) handled, passing of information across an application topology has to concern corresponding **media specific QoS descriptions**.

For instance, a video processing component may describe its QoS support in terms of parameters such as frame size, frame rate and encoding scheme, since it was designed to process data units formatted correspondingly. An audio component may use similar parameters, i.e. audio sample size and rate. Media specific differences may be given in form of different parameters, at least in form of different implied

[1] The term "heterogeneous receivers" is often used in the literature to indicate this property (e.g. [ZDE+93]).

dimensions. 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.

The protocol presented in this paper relates to negotiation at the level of application topologies and media specific QoS as described last. It is independent of enduser oriented QoS abstractions. Offering such abstractions through appropriate user interfaces is left entirely to the *CINEMA* client initiating an application session. Similarly, the protocol is independent of lower level protocols used for resource reservation at the transport level (see section 4). Consequently, requirements elaborated below are focused on application level negotiation only.

QoS ranges

Technological developments show that a multitude of media specific qualities can be made available (e.g. CIF, QCIF, VGA, HDTV video frame sizes; 8, 11, 22, 44 kHz audio sampling rates; etc.). Given this fact, many application components will be designed to support **various QoS**. As a rather simple example, one may envisage a video-on-demand server component which is able to provide videos with various picture sizes. In consequence, a client may compile QoS requests as **QoS value ranges** indicating best desired and worst accepted QoS for a sink component.

Using ranges in QoS requests avoids drawbacks arising from the limited knowledge a client can have about resource availability in communication links and endsystems. Consider the case of a video server component connected to a remote user's sink component. The interconnecting communication link is subject to resource availability changing in time if the link is shared among many possible connections. If no QoS ranges were used, negotiation would amount to a yes/no decision indicating session initiation success or failure. In case the client would be satisfied with lower QoS selections, it would have to retry session instantiation with successively lower QoS values. Using QoS ranges avoids this overhead, since it delivers the best possible selection at a given time of resource availability.[1]

Negotiation with and without resource reservation

Application level negotiation is required both in the case of unlimited and limited resources (on endsystems and/or in communication links). Assume the simple scenario of a video source component multicasting a video stream to a set of sink components (Figure 2a). As mentioned earlier, QoS requests for

[1] QoS models at the transport level already support ranges (e.g. XTPX [MMR93]).

these sink components may be selected independently. In this case, negotiation has to ensure that the source component provides a QoS which is acceptable to all sink components and lies within the QoS range supported functionally by the source component. To perform these operations negotiation is required, although no resource availability is considered. Rather, QoS dependencies are taken into account imposed by the application topology.

In case of **limited resources**, resource reservation has to be performed in addition in order to assess the QoS which can be provided at a given time and to grant stable QoS. For instance, in case the communication link prior to sink 1 (S1) in Figure 2a would enforce a QoS reduction to 20 due to resource scarcity, this reduction would have to be taken into account when performing negotiation.

Variable filters

The above example showed QoS dependencies that exist across an application topology. In case the intersection of requested QoS ranges yielded an empty set, session initiation would fail. The same would happen, if the intersection delivered a QoS range for which no sufficient resources were found, even though resources would have been available to satisfy QoS for each of the sink components individually. Figure 2a shows such a case: the QoS range intersection yields a range of QoSR(60 .. 70), but the communication link to S1 can support QoS only up to 20, a value which could be supported both by the source component and S1, but is contradicted by the QoS request of S2: QoSR(60 .. 100).

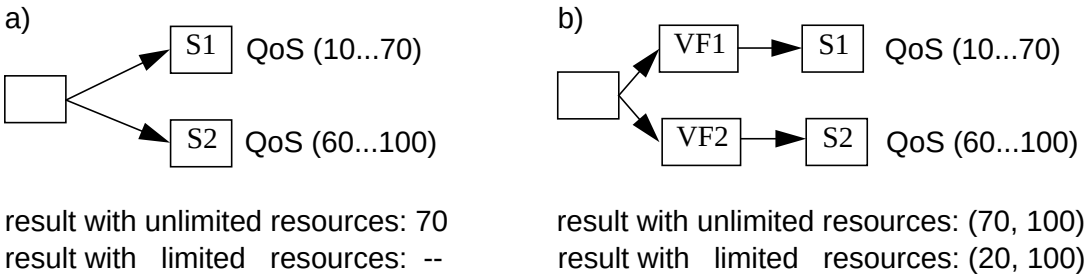


Figure 2 : Use of Variable Filters

A better solution would strive to isolate the impact of QoS dependencies and reductions. For this purpose, we consider the concept of special components, termed **variable filters**. In effect, a variable filter decouples QoS provided at its input port from QoS delivered at its output port. In Figure 2b, prior to each sink component such a variable filter is inserted. Should, for instance, QoS requests for S1 and S2 yield an empty intersection, the variable filters VF1 and VF2 can be controlled during negotiation to generate

individual QoS for both sink components. Similarly, a QoS reduction enforced by the link preceding for instance S1 can be prevented from topology-wide dissemination (e.g. to S2) by controlling VF1 during negotiation to reduce QoS received from the source component appropriately. These examples show that variable filters are reasonable both for the case of unlimited and limited resources.

It is important to see that QoS reduction in such filters is not parametrized in advance by the client, but is controlled by and at the time of the QoS negotiation process. The reason is that the process of finding filter parameters in complex topologies may be a complex and time-consuming task, since it depends on the state (of resource availability and QoS dependencies) of the whole distributed application. Basically, this task would amount to a time-consuming trial-and-error approach for client defined filter parametrization. These drawbacks can be overcome by using variable filters.

Topology complexity

The examples of Figure 2 exhibited rather simple structures. Other applications may include in addition mixing components, i.e. components with more than one input port. Such components are used to process data units of several data streams to produce new data units. For instance, audio mixers are employed to generate a single audio stream out of different audio streams. Similar mixers exist for video streams, for instance to generate virtual aggregations of objects monitored at distributed sources or to place one video into another (“picture-in-picture”). Introducing mixing components into application topologies makes negotiation even more complex, given that they allow to construct topologies of most various structures (e.g. Figure 1) and introduce new QoS dependencies between their input ports (e.g. same video frame size for two video streams received).

Requirements to NRP

Above arguments laid the ground for requirements to application level negotiation for networked multimedia applications: support for media specific QoS parameters, support for QoS value ranges, support for complex component topologies and support for control of QoS reductions in variable filters. Meeting these requirements was the driving force for developing the **Negotiation and Resource Reservation Protocol** (NRP) presented in this paper. Its name reflects the targeted goals. NRP was designed to provide for QoS negotiation both for cases where QoS dependencies have to be resolved without underlying resource limitations and for cases where resource limitations (at least in some topology parts) have to be taken into account by resource reservation.

4 Negotiation and Resource Reservation Protocol

4.1 Assumptions and Scope

NRP is initiated when a client issues a session set-up command. NRP uses client QoS requests which are associated with input ports of sink components. For each such port the attached QoS defines the QoS requested for the stream feeding the port. All component ports are typed, i.e. to each component port a streamtype object exists defining the (media specific) QoS parameter set governing QoS description at that port. This parameter set reflects the characteristics of the data which are provided to a component (at input ports) or delivered by a component (at output ports). For instance, a video mixer component may use at all its ports a QoS description scheme (i.e. streamtype object) consisting of parameters (frame rate, frame size, color depth).

Two component ports which are interconnected via a link have to support the same QoS parameter set in order to be compatible. Compatibility is checked in *CINEMA* prior to initiating NRP. For this paper, we assume for simplicity that at each component port the same parameter set is supported. Besides the parameter set, at each component port for each parameter a value range is specified indicating built-in functional support (e.g. support for video frame sizes between 100x100 and 600x600). This is static information which exists independently of a session and serves NRP during session set-up to take into account functional design limitations of components.

NRP was designed to be applicable when using various component types in an application: source and sink components, fixed filters, mixing components and variable filters. This classification is required to distinguish the way components interlink QoS between their input and output ports. The most simple components in this respect are source and sink components. They are supposed to have exactly one port, one output respectively one input port.

A fixed filter is supposed to have one input and one output port and to map QoS between them in a fixed way. Many components will for instance leave the parameter “picture size” unchanged when performing image processing (e.g. image smoothing). In contrast, other fixed filters may realize a fixed reduction for instance from full picture size to half picture size.

A mixing component may have two or more input ports and one output port. All input ports are supposed to have a fixed relationship between their QoS values. An audio mixer may enforce in this way that the same audio quality is mixed, while a video mixer may enforce that pictures of the same size are mixed. The QoS provided at the output port is supposed to be coupled in a fixed way to the QoS of one of the input ports (implicitly to the QoS of the other input ports as well).

Finally, a variable filter is supposed to have one input and one output port with a variable mapping between input and output QoS. For instance, given a range (min, max) for picture size at its input port the variable mapping only requires that the output QoS be below max. In other words, a variable filter can be controlled to supply a mapping between a maximum value at the input port and any QoS value below. This property was motivated Section 3.

We restricted component type definition in order to keep the description of NRP simple. Without proof, we state that much more general classes can be supported. As a plausibility argument take the possibility to model more component types with more complex properties out of components defined above. For instance, a multicaster component (with one input port and several output ports) can be modelled as a sequence of a fixed filter connected via a multicast link to a set of fixed filters. A source with several output ports can be modelled similarly. More complex components can be constructed: a mixer component with variable relations between input port parameters can be modelled as consisting of a mixer component and a variable filter connected to each of the mixer input ports.

NRP was designed to be applicable to a so-called two-zone model of component topologies. In this model, the topology is requested to be composed of two zones: a “mixing” zone containing source components, fixed and variable filters and mixing components interconnected in arbitrary sequence, but only via unicast links, and a second “multicasting” zone containing all but mixing components, interconnected via unicast and/or multicast links. The definition covers common topologies including pure multicast topologies (with arbitrarily many multicast links), pure mixing topologies (with arbitrarily many mixing stages) and as shown in Figure 1 combinations of such topologies. Here , the camera components and the mixer belong to the first zone and the variable filters and the display components belong to the second zone. In section 5, we discuss briefly topologies beyond the two-zone model.

Before presenting a first example of how NRP works, we summarize capabilities of NRP and restrictions assumed to keep discussions and explanations simple. NRP supports negotiation for various component types which can be modelled out of source and sink components, fixed and variable filters, as well as mixing components. NRP will be explained in the context of these four basic component types. NRP is applicable to the introduced two-zone model and will be explained for this full model. NRP handles value sets of QoS for a media specific parameter, the NRP description will resort to QoS value ranges given as (min, max) value sets. Finally, NRP supports QoS negotiation for multiple media specific parameters for a specific medium, for instance for a video stream both picture size and rate may have to and can be considered during negotiation. For simplicity, NRP description will relate to one parameter only (see Section 5 for a brief discussion of multiple parameters).

4.2 NRP Example

The following example outlines how NRP is performed. Figure 1 depicts a video conferencing scenario with two participants. At each participant site a camera component and a display component are installed. The two video streams produced by the camera components are mixed by a mixer component. The resulting video stream is delivered to two variable filters which can provide individual video qualities for the display components.

For the example we use the frame size of the video streams as an appropriate media specific parameter to be negotiated by NRP, while assuming that the video frame rate is fixed in advance to, let's say, 25 frames/s. In order to keep the protocol description simple, we bypass the links and assume that all links can reserve resources as required with one exception: the link between the variable filter VF1 and the display component Dsp1 shall enforce a QoS reduction due to resource scarcity. We assume that the camera component 1 (Cam1) can produce a video frame size between 480x360 and 320x240. The second camera component (Cam2) shall support 640x480 downto 320x240. The client requests shall correspond to the functional capabilities of the camera components, i.e. at display 1 (Dsp1) a 480x360 presentation is preferred and at display 2 (Dsp2) a 640x480 presentation. Both clients will also accept a lower quality presentation (downto 320x240) if a better quality cannot be supported (for instance as a consequence of resource scarcity). The mixer (Mix) requires a fixed equality relationship between incoming frame sizes.

NRP proceeds in three phases and starts at the sources (both aspects will be discussed in Section 5). In each phase so-called Application FlowSpecs (AFS) are propagated between sources and sinks throughout the topology. An AFS contains the media specific QoS parameter(s) (here: frame size) with associated value (ranges). The first phase starts at the sources and serves to reserve resources based on QoS values supplied by sources. AFS arriving at sinks are matched (i.e. intersected) against the value ranges specified by the application client. The second and third phases serve to calculate the final QoS values which have to be fixed for each component port as result of negotiation. The third phase in addition relaxes resource reservation according to the final QoS values. The second phase runs backward, i.e. from sinks to sources, while the third phase starts at sources and ends at sinks. NRP finishes after completion of the third phase. The three phases for the introduced scenario are as follows.

The first phase starts with reservations at the sources, where reservation is assumed to be possible for the whole QoS ranges of QoSR(480x360 ... 320x240) at Cam1 and QoSR(640x480 ... 320x240) at Cam2. These ranges are propagated in two AFS to the mixer, where input port matching yields a possible range of QoS(480x360 ... 320x240). For the mixer's output port the unchanged range QoSR(480x360 ... 320x240) is derived. The mixer is assumed to be able to reserve resources for the most resource consuming value of the whole range, i.e. 480x360. Both variable filters VF1 and VF2 can reserve for the ranges QoSR(480x360 ... 320x240) at their input and output ports. Assume that the link between VF1 and Dsp1 could reserve only for QoSR(320x240). At display components Dsp1 and Dsp2, the arriving ranges are matched against the ranges specified by the application client, which are as mentioned above (320x240) for Dsp1 and (640x480 ... 320x240) for Dsp2. Thus, Dsp1 is invoked with QoSR(320x240) for reservation, while Dsp2 is invoked with range QoSR(480x360 ... 320x240) for reservation. Both reservations are assumed to be successful. The first NRP phase ends here.

The second phase proceeds from the sinks towards the sources. The AFS at the sinks contain at the beginning QoSR(320x240) for Dsp1 and QoSR(480x360 ... 320x240) for Dsp2. VF2 translates the AFS received from Dsp2 into an AFS for its input port containing (480x360 ... 320x240). Similarly, VF1 generates an AFS containing QoSR(480x360 ... 320x240). Note that the latter translation is in line with the ability of variable filter VF1 to reduce QoS from any of the values (480x360 ... 320x240) to 320x240, the value received from Dsp1. On the other side, the translation from (320x240) to (480x360 ... 320x240) is necessary for the following reason. VF1 cannot know whether the frame size at its input port affects

other components as well. Were VF1 to translate QoSR(320x240) unchanged to its input port, one can see from Figure 1 that the multicast property of the mixer's outgoing link would enforce a matching of received AFS to 320x240. Proceeding this way, NRP would unnecessarily restrict the frame size towards VF2 and Dsp2 to 320x240.

In contrast, our approach ensures that an AFS received during the second phase indicates the full range which *can* be supported by the topology situated downstream from a communication output port. In the figure above, the mixer receives for its output port QoSR(480x360 ... 320x240) which can be supported along both downstream branches. In conclusion, during the second phase, each AFS has to contain the QoS range supportable in downstream direction. However, this QoSR does not suffice.

Consider the case, where both VF1 and VF2 would indicate a supportable range of QoSR(640x480 .. 320x240), although none of them requests a frame size higher than 480x360 at their respective output port (since both Dsp1 and Dsp2 do not require more than 480x360). It is obvious, that the QoSR alone cannot give the mixer component any hint that only 480x360 is required. We remove this deficit by requiring the AFS in the second phase to include a second part, a so-called DRV (downstream request value) indicating the maximum requested quality in downstream direction from a component's output port. Thus, the AFS delivered by VF1 consists of (QoSR(480x360 ... 320x240), DRV(320x240)) and the AFS delivered by VF2 consists of (QoSR(480x360 ... 320x240), DRV(480x360)). The two AFS are propagated to the output port of the mixer. Assume for the time being that it is the mixer who matches the two AFS into one AFS(QoSR(480x360 ... 320x240), DRV 480x360)) and passes this AFS unchanged to components Cam1 and Cam2 through its input ports. Upon arrival of the AFS at source components Cam1 respectively Cam2 the second NRP phase is done. Note that matching of the two AFS at the mixer's output port is done differently for the QoSR and DRV parts: the two QoSR were intersected, while the DRV values were maximised (within the bounds of QoSR).

In the third phase each component relaxes resource reservation according to propagated DRV values. Source Cam1 fixes its output frame size to the DRV value received during the second phase, i.e. to 480x360. In addition, Cam1 propagates in its AFS the fixed DRV value to the mixer. Cam2 proceeds similarly, i.e. fixes its output frame size to 480x360 (relaxes resource reservation !) and propagates this DRV to the mixer. During the third phase only the DRV field is considered. Upon receipt of the two AFS from Cam1 and Cam2, the mixer fixes its input and output frame sizes to 480x360 and relays the

DRV(480x360) to both VF1 and VF2. VF1 knows that at its output port only a DRV of 320x240 was received during the second phase (i.e. this value is retained in the second phase). Based on this knowledge and the arriving DRV of 480x360 at its input port, VF1 is able to deduce that a resizing from 480x360 to 320x240 is required. It fixes its internal behaviour accordingly. In contrast, VF2 has to deliver 480x360 at its output port, so it does not have to do any resizing of the incoming 480x360 frame size. As a result, VF2 maintains resources acquired during phase 1. VF1 relays DRV(320x240) to Dsp1, VF2 relays DRV(480x360) to Dsp2. In both cases, no resource relaxation has to take place. Upon the AFS arrival at sink components Dsp1 and Dsp2, the third phase of NRP is finished and a resource reservation session was set up.

The AFS structure introduced here contained an entry for the media specific parameter considered. As motivated above, the three NRP phases taken as a whole required two fields in this entry: a QoSR field and a DRV field. The former was used during the first two phases, though with different interpretation. In the first phase, QoSR denoted the values which could be supported (functionally and by resource reservation) in upstream direction, while in the second phase QoSR denoted the values which could be supported in downstream direction (given upstream reservations of the first phase). The DRV value was required due to the existence of variable filters to perform related calculations correctly.

4.3 Implementation Architecture

NRP is performed by a set of protocol agents (PA). Figure 3 shows a realization with one PA per component (C) and link (L), where PAs are interconnected according to the component topology of the application.[1]

An important feature of NRP is that it allows components and links to take a local view to negotiation. They only have to support a few methods (see below) which are invoked by NRP (PAs) during negotiation. A component or link is only required to interact with its associated PA, but not with any other component or link. On the other side, the sole task of NRP[2] is to provide for correct AFS passing between components and links. This passing is done towards components and links via component and link

[1] An alternative realization is to have one PA per node, which interacts with the local components of the flow graph. The order in which the components are to be called is derived from the topology.

[2] Besides negotiation triggering, exceptional and regular negotiation termination.

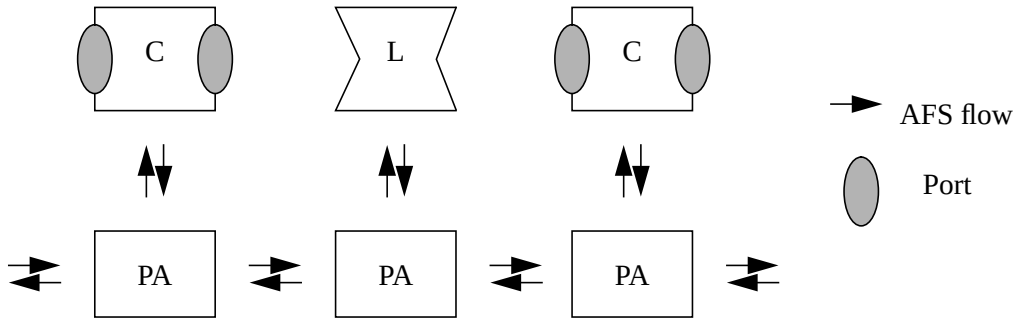


Figure 3 : Implementation Architecture

method invocations. In addition, interconnected NRP PAs exchange AFS according to the direction of the protocol phase (e.g. in source-to-sink direction during the first phase). Keeping in mind that NRP includes client QoS requests into negotiation, NRP can be said to be the negotiation glue between client, components and links encapsulating all topology-oriented AFS distribution.

The architecture of NRP requires each component to provide a generic interface for negotiation. This interface consists of a set of methods where each method corresponds to one of the three phases introduced in the previous section:

- *Reserve* (invoked during phase 1)
- *PropagateBack* (invoked during phase 2)
- *Relax* (invoked during phase 3)

In addition a fourth method *Free* has to be offered to indicate to a component that a reservation session is to be terminated. Except for *PropagateBack* the same methods are provided by a link. The reason is, that links are not invoked during the second NRP phase (see Phase 2, below). Except for *Free*, each of these method invocations incorporates a (set of) AFS and yields a (set of) new AFS calculated by the component or link.

In the first and third NRP phase, each component is given one AFS for each of its input ports, while returning as result one AFS for its output port (note, we have assumed earlier one output port to exist for every component type). In the second phase, each component is invoked with one AFS for its output port and returns one AFS for each of its input ports.

Upon invocation, each component has to ensure that AFS calculated as response to the invocation are in line with (a) resource availability encountered internally when requesting resources (e.g. CPU) and (b) AFS received in the invocation. The former has to be taken into account during the first phase, the latter during all phases. In addition to these requirements, variable filters are expected to have a certain amount of “memory” conserving AFS information across NRP phases.

Links are invoked by NRP for resource reservation required for data transfer between component ports. Links encapsulate internally the transfer mechanism (e.g. transport system) used. Links, like components, are invoked by NRP by passing them (media specific) AFS. Mappings required to derive lower level QoS representations (e.g. transport level parameter values) are encapsulated in the link objects.

Links may be of local or remote type, depending on whether they interconnect component ports on the same or different machines. A local link may for instance encapsulate a shared buffer mechanism for data transfer. To each local link one NRP PA is associated. The PA has the same view to the link as to a component with one input and one output port. Thus, it invokes it with one AFS and receives one AFS as result.

A remote link has to encapsulate a transport system for data transfer. In order to offer a simple view to NRP, a remote link is split into a source and a destination link object (DLO). In case of a multicast link, a set of DLOs are used. Each SLO and DLO is associated with one PA. In consequence, each such PA views its SLO and DLO as if it were a local link, i.e. a component with one input and output port. In [BDFR95] we indicate that this approach allows for arbitrary transport connection set-up schemes (e.g. sender or receiver initiated).

In the first and third NRP phase, each link object is given one AFS and returns one AFS as result. As mentioned, no link invokations take place during the second NRP phase. As for components, each link object has to ensure that AFS calculated in response to PA invokations are in line with the received AFS and resource availability encountered internally, for instance when setting up a transport connection or reserving shared buffers.

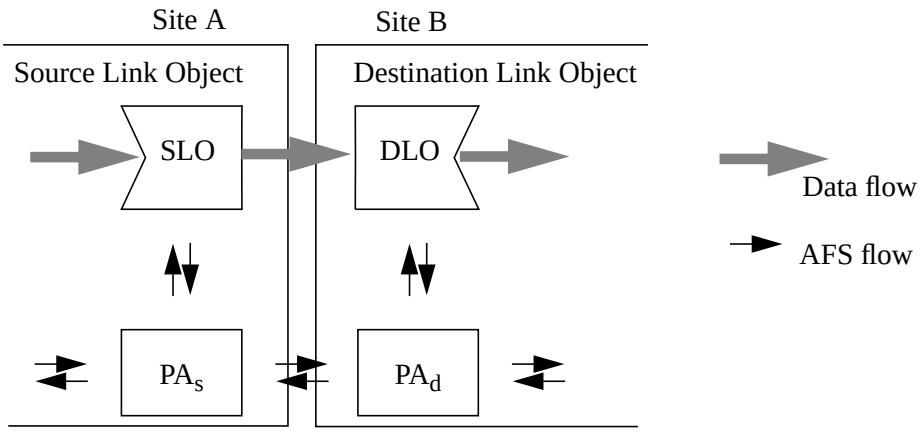


Figure 4 : A Remote Link

Phase 1 (Reserve)

In the first phase a component uses the AFS received for each of its input ports to calculate the AFS which it can provide for its output port. In case resources do not suffice for the whole QoS range of the received AFS, the component reduces the range accordingly. The reduction is applied to the received AFS and, as a consequence, also to the output port's AFS.

A *fixed filter* calculates the output port AFS by using its fixed relationship between input and output QoS. If, for instance, the filter is to reduce a received video frame size to 50%, the output AFS will be calculated by "halving" the values of the received AFS, for instance from received QoSR(320x240 ... 640x480) to QoSR(160x120 ... 320x240). In case the filter cannot support better qualities due to resource shortage or functional limitations, it reduces the received AFS correspondingly, for instance to QoSR(320x240 ... 480x360), which of course results in a corresponding reduction of the output port AFS as well.

A *mixer* proceeds in a similar way like a fixed filter, however, prior to calculating the output port AFS, the mixer has to match all AFS (i.e. the QoSR contained in the AFS) in order to preserve identity of QoS received at its ports. The matched AFS is used for calculating the output port's AFS.

A *variable filter* acts in the first phase in a similar way like a fixed filter with the fixed relationship between input and output port QoS fixed to identity. Two differences exist. The filter sets the lower bound of the calculated QoSR to 0 (or a functionally predefined lower bound), since a variable filter is assumed

to be able to convert to any lower quality. In addition, the filter is assumed to store the received (and possibly reduced) AFS for later use.

Source and sink components deviate in their behaviour from the above description, since they only have one output respectively input port. Each source component is invoked with an empty AFS, since these invocations mark the beginning of the first NRP phase. A *source component* reserves resources according to its best ability and returns an AFS with a QoS range QoSR indicating this ability.

A *sink component* is invoked with one AFS for its input port. The component reserves resources based on this AFS and calculates a new AFS for its input port containing a possibly shrunk QoS range. Besides this range QoSR, the sink component includes a DRV value into the response AFS, since upon returning this AFS the second NRP phase is started requiring DRV values for calculation (as motivated in 4.2). The DRV value is set to the maximal value of QoSR. A returned AFS has the form: AFS(QoSR, DRV) (see Section 4.2).

A *link object* is invoked with one AFS. Based on this AFS, it translates the media specific QoS onto transport system specific QoS for the encapsulated transport system (e.g. XTPX [MMR93]). It invokes the transport system with values for connection establishment and resource reservation. Values accepted by the transport system are recalculated into the AFS representation, which is handed back to the associated PA. We refer to [BDFR95] for a method of mapping between media specific and transport QoS values.

Since a remote link consists of a source link object and a destination link object, the above actions are performed only by the link object which establishes the connection. The other link object returns the received AFS unchanged back. In case of a multicast link, the PA associated to the source link object passes a copy of the AFS to each PA associated to the destination link objects.

Phase 2 (PropagateBack)

A component is invoked in this phase with one AFS for its output port and generates one AFS for each of its input ports. Sink components are not invoked (respectively their invocation for phase 1 and 2 coincide).

A *fixed filter* calculates the new AFS similar to phase 1, except that it applies its internal mapping (e.g. reduction of frame size to 50%) in inverse direction and both to QoSR and DRV. Pursuing the example from above, the filter calculates from a received AFS(QoSR(100x100... 200x200), DRV(200)) the following AFS for its input port: AFS(QoSR(200x200 ... 400x400), DRV(400)). A mixing component acts in the same way like a fixed filter, except that the calculated AFS is returned for each of the input ports (i.e. for n input ports, n AFS are returned).

A *variable filter* acts differently. As motivated in Section 4.2, it calculates the new AFS using the following semantics for QoSR and DRV: QoSR denotes all quality values which can be supported in downstream direction and DRV denoted the value truly requested in downstream direction. To preserve these semantics, a variable filter calculates for the input port:

(a) a QoSR retaining all values from the QoSR stored in phase 1 for the input port, except for those which are below the QoSR received for the output port. Such exclusions may be necessary, since client requests may have excluded some of the lower QoS values which were indicated as supportable by the variable filter after phase 1. All other values still make sense (i.e. are supportable) for providing at least one QoS value indicated as possible in downstream direction from the filter's output port (i.e. a value contained in the received QoSR).

(b) a DRV value which is set to the DRV value received, if this value is within the newly calculated QoSR. This value may not be part of this new QoSR, in case the received DRV is below the QoSR stored in phase 1. Recall that in phase 1, a variable filter generated for its output port a QoSR with a minimal value of 0, since downstream of its output port all lower qualities are supportable. Traversing the filter in backward direction enforces the transformation of such a "lower" DRV value which can be supported at the filter's input port. Hence, if the received DRV is not part of the newly calculated QoSR, a DRV is taken for the input port which is equal to the lowest value of the new QoSR.

Besides calculating the AFS for the input port, a variable filter stores the AFS received at its output port (more exactly, the DRV received). As an example for a variable filter calculating an AFS for its input port, refer to the example of Section 4.2.

As in phase 1, the behaviour of a source component deviates from the general description. A *source component* uses a received AFS to extract the DRV value and to install it as the final QoS value. Doing so,

it may install a value which is below the value for which resource reservation was done in phase 1. In this case, it would relax reservation in accordance to the received DRV value. Finally, it returns the AFS (unchanged) for usage in the third phase.

Link objects are not invoked during the second phase but their associated PAs have to perform some actions. A destination link object PA passes its AFS to the connected upstream source link object PA. In case of a multicast link, the source link object PA receives one AFS from each connected destination link object PA. Since these AFS may be distinct, the source link object PA merges these AFS into exactly one AFS. The rule for this matching is given as:

matched AFS = (matched QoSR, matched DRV),

with matched QoSR being the Intersection of received QoSR ranges and matched DRV being the maximal value of received DRV values which is also contained in matched QoSR (see 4.2 for motivation).

The reason why the matching is anchored outside components (unlike in the mentioned example) is that NRP is supposed to invoke a component with exactly one AFS per port, thus making it independent of the number of receivers attached to a possible multicast link following that port. On the other side, this matching can be done outside of components, since it involves only usual set related operations: range intersections and selection of a best value out of a set of values. The latter property is ensured due to the well-ordering requirement to which possible values of a media specific parameter are subject.

Phase 3 (Relax)

A component in this phase is invoked with one AFS for each input port and generates an AFS for each output port. Source components are not invoked (respectively the invocations for phases 2 and 3 coincide). In this phase, only the DRV values of the AFS are considered. All components use received AFS to relax resources, in case their initial reservation (of phase 1) corresponded to a higher quality.

A *fixed filter* uses the received DRV value to relax resources. It responds with an AFS containing the DRV value received. A *mixer* acts similarly. Note that in this phase a mixer is automatically provided with consistent DRV values at all its input ports, since upstream from any mixer no DRV value adjustments take place in the second phase which affect the mixer's input ports.

A *variable filter* acts again differently. It first retrieves the DRV value stored in phase 2 for its output port and compares it to the received value. In case the received value is greater or equal to the stored one, the filter installs at its input port the received DRV value as its final quality, and at the output port the DRV value stored in phase 2. Otherwise, the filter installs for both input and output port the same, namely the DRV value received. Doing so, the filter fixes its, previously variable, internal function for QoS reduction. Resource reservation may be relaxed according to the installed qualities. The filter generates an AFS containing the DRV value installed for its output port.

A *sink component* uses the received DRV value to relax reservation like a fixed filter. It does not generate an AFS, instead it merely indicates that resource relaxation was completed. When all sink components have indicated their completion the third phase of NRP was finished and a resource reservation session was set up.

In case of a *link object*, the method Relax is handled internally exactly in the same way as the Reserve method with one exception: the link invokes the transport system not for reservation but for resource relaxation.

5 Discussion

The design of the NRP phases is closely linked to the existence of variable filters. To explain this relationship, we first consider below protocol design alternatives for two-zone topologies containing no variable filters. We observe that any topology has two distinguished locations which could be used for starting NRP: sink components and source components.

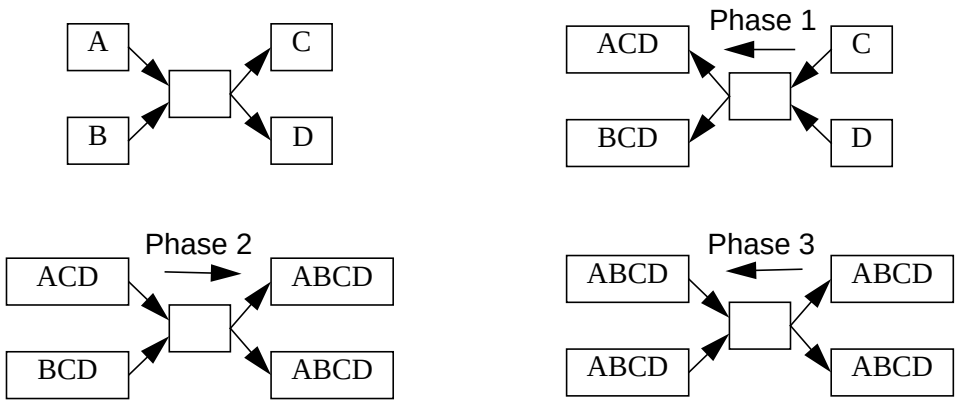


Figure 5 : QoS Information Spreading

Starting NRP at the sink components has the advantage that from the beginning client QoS requests can be taken into account. The protocol would require three phases to disseminate QoS information throughout the topology (Figure 5). The first phase would spread sink QoS information (C, D) towards source components, while the second phase would spread source QoS information (A, B) towards sink components. The third phase would ensure that source QoS information is exchanged between any two source component pairs. Resource reservation would have to be done in the first protocol phase in order to ensure that QoS information spread in the first and later phases relies on “sure” (i.e. already reserved) resources.

Since two-zone topologies are symmetric with respect to source and sink components (i.e. for each two-zone topology there is a “turned” topology where sources become sinks and vice-versa), the above protocol could equally well be started at source components. However, in this case resource reservation in the first phase would have to be done without knowledge of client QoS requests, i.e. based on QoS capabilities of the source components. In consequence, a protocol for two-zone topologies without variable filters would be preferably defined to start at the sink components. Furthermore, it would have to include a minimum of three phases.

Introducing variable filters into the topology disturbs the symmetry mentioned above, since variable filters reduce QoS in one (towards sinks) direction only. It is obvious, that any protocol coping with such filters in addition, requires at least the number of phases required by the protocols described above. The question is whether the number of three phases can be maintained.

The NRP protocol presented in 4.2 relied on three phases only. As argued above, resource reservation has to be done in the first phase, if three phases shall suffice. Starting NRP at sink components would have meant to reserve resources for variable filters in the first phase when traversing them in backward direction (i.e. towards sources). However, such a reservation is impossible since, while for variable filter’s output port a desired QoS request (in form of an AFS) may exist, there is no corresponding QoS request for its input port (recall that a variable filter can translate any input QoS into any lower output QoS). In consequence, starting NRP at sink components would have required more than three phases, more exactly (but without proof here) five phases. In order to avoid these additional phases, we decided NRP to start at source components.

Topologies beyond the mentioned two zone model are not supported by NRP. One of the reasons is the possible occurrence of QoS dependencies across mixing/multicasting cascades. In a simple form, such a cascade may consist of a component multicasting to two mixers each of which receive additional data through another link (see Figure 6). If both mixers require a fixed equality relation for qualities between their input ports, NRP would have to settle on one value for all depicted ports. Assuming for instance that at port 1 the smallest possible media specific value is provided, NRP would have to spread this information from port 1 to port 3. Since NRP employs AFS forward respectively backward propagation phases directed from sources to sinks respectively vice-versa, the cascade example would require at least 4 phases. More generally, the number of required phases for a forward/backward protocol depends on the “breadth” of a cascade. Topologies with such characteristics are current topic of research beyond NRP.

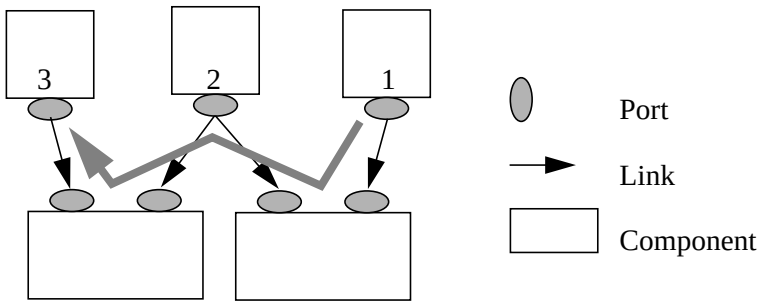


Figure 6 : A Cascade Topology

The description of the negotiation protocol considered only one media specific parameter. We indicate here that cases with more media specific parameters can be solved by including an additional scheme of priorities among them. The scheme indicates to each component which of the parameters would have to be reduced first, in case negotiation indicates necessity of QoS reduction. The priority scheme would have to be defined by the application client, in case priority selection is an application specific decision. For instance, the client may specify that for a video stream picture size is more important than picture rate, so whenever a choice is possible, the rate should be reduced before reducing the size.

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.

Delay is only considered during the second and third phase of NRP. At the end of the first phase, the client specified target values for each sink port are inserted into the AFS associated with that port. During the second phase, components and ports do not perform any operations on these values, i.e.required handling is assigned to NRP. NRP stores for each port of a component the target value specified for sinks situated downstream from the component. In case a component port feeds a multicast link, the target values received from downstream sinks are merged into one value by taking their minimum.

The third phase of NRP is used to accumulate the contributions of components and links to the end-to-end delay values. Each component and link provides a corresponding contribution delay value in its response to a *relax* invocation. NRP accumulates these values in the field carried in the AFS. When reaching an input or output port during AFS propagation, the target value stored for that port is compared to the current value in the AFS. If the latter exceeds the former, resource reservation failed to meet requested end-to-end targets. Note that NRP accumulates end-to-end delay values according to relaxed (i.e. final) reservations.

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

In the past, several protocols have been designed to support resource reservation between two endsystems across network links. SRP [AHS90], ST-II [Topo90] and RSVP [ZDE+93] were designed to support reservation for multicast communication between a source and many sinks (SRP, ST-II) or between many source/many sinks (RSVP) by assuming that no processing has to take place between sources and sinks. Reservation was based on transport parameters only and media specific parameters were not regarded.

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 (network) level parameters ([KrLi94], [BDF+94], [SMHW95]). [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. [KrLi94] uses a scheme to perform subsequent negotiation and resource reservation inside the network. Application level negotiation in the sense described in this paper is not considered and neither of these approaches considers topologies beyond client/server.

[NaSm95] introduces a QoS Broker which negotiates end-to-end QoS at application level for a telerobotics application. Selection of media specific QoS is done between application components and is coupled with resource reservation both on endsystems and communication links. However, this approach is restricted to client/server settings only and cannot be applied to more general application scenarios as described in this paper.

7 Summary

In this paper we presented a protocol performing QoS negotiation at the application level. We described this level as the level where distributed application topologies are visible and where QoS information is exchanged between components and links in order to resolve QoS dependencies. We motivated the media specific nature of QoS descriptions exchanged and defined an appropriate interface to a client expressing QoS requests to an application.

We showed that QoS dependencies at the application level are imposed by predefined functional capabilities of components, the application topology and resource availability for components and links. In

consequence, we argued that negotiation is required even if unlimited resources are assumed requiring no resource reservation.

We introduced NRP as a protocol being able to cope with such requirements. NRP was designed to be applicable to application topologies based on a two zone model and consisting of a number of component types including in particular variable filters. NRP supports the transfer of media specific parameters in application flowspecs, but does not depend on the semantics of these parameters.

We presented an implementation architecture to explain the role of NRP and the behaviour expected from components and link objects. We discussed why a minimum of three phases is required and why NRP starts at source components to reach this mimimal number.

Besides the protocol description itself, we view as main results of this contribution the definition of the problem of QoS negotiation at the application level and of aspects which have to be considered in this context. Our current activities are focused towards covering application topologies beyond the two-zone model and including delay and jitter as negotiation parameters into NRP.

8 References

[AHS90] D. Anderson, et al. SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet. *Technical Report No. UCB/CSD 90/562*, Computer Science Division (EECS) University of California, Berkeley, CA, 2 1990.

[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.

[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.

[BDFR95] I. Barth, G. Dermler, W. Fiederer, K. Rothermel. A Framework for Negotiable Quality of Service in Distributed Multimedia Systems. *Technical Report submitted for publication.*

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

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

[KrLi94] A. Krishnamurthy, T.D.C. Little. Connection-Oriented Service Renegotiation for Scalable Video Delivery. In *International Conference on Multimedia Computing and Systems*, p. 502–507, 5 1994.

[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. *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.