# The QoS Broker

Klara Nahrstedt and Jonathan M. Smith*

Distributed Systems Laboratory
University of Pennsylvania, Philadelphia, PA 19104-6389

**Abstract**

Many networked multimedia applications are delay-sensitive, and require services with guarantees of resource availability and timeliness. For networks such as those based on Asynchronous Transfer Mode (ATM), these service requirements are specified through Quality of Service (QoS) parameters. QoS guarantees are needed at multiple layers in an end-to-end protocol architecture. Delivering end-to-end QoS requires an architecture for resource management at the system end-points (e.g., computer workstation hosts), as well as in the underlying network.

We describe a model for an end-point entity called the *QoS Broker*. The broker orchestrates resources at the end-points, coordinating resource management across layer boundaries. As an intermediary, it hides implementation details from applications and per-layer resource managers. Services such as translation, admission and negotiation are used by the broker to properly configure the system to application needs. Configuration is achieved via QoS negotiation resulting in one or more connections through the communications system. The negotiation involves all components of the communication system needed for the setup.

An important property of the broker is its role as an *active* intermediary which insulates cooperating entities from operational details of other entities. The broker manages communication among the entities to create the desired system configuration. It can be viewed as a software engineering technique for distributed multimedia system architectures.

We have implemented an experimental prototype of a QoS Broker on IBM RISC System/6000 hosts connected by an ATM LAN. This prototype was validated using a telerobotics application. This application has very strict timing constraints. It helped us to identify limitations of our system and requirements for system support, and serves as a testbed for our architecture.

## 1 Introduction

General multimedia communications systems (of which teleconferencing systems are a subset) require an architecture in which the system elements, such as protocol stacks, devices and schedules, are configured and coordinated as a *system*. Since many such systems are delay-sensitive, services that they employ must provide timing guarantees. The problem of providing real-time communications between hosts is becoming better understood, but networked applications require a diversity

*Research support for this work came from Bellcore (through Project DAWN), IBM, Hewlett-Packard, and from the Corporation for National Research Initiatives (CNRI), which is funded by the National Science Foundation and the Defense Advanced Research Projects Agency under cooperative agreement # NCR-8919038.

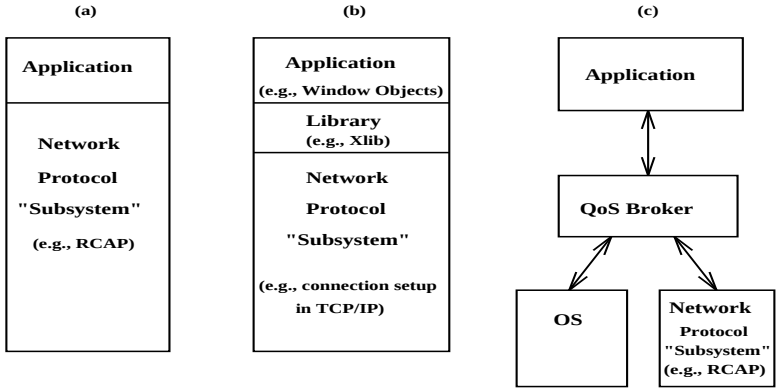


Figure 1: *Organizational models for resource management*

of resources, of which communication is only one. The other resources, such as processing capacity, must be managed in concert with networking to deliver guaranteed behavior to applications.

In the design of a communication system for a teleoperation application which uses an ATM network, we were faced with the problem of supporting applications which have strict timing requirements, yet wish to isolate themselves from details of process scheduling or bandwidth allocation. A new architectural model was needed for specifying application requirements, and translating those requirements into negotiated resource allocations. This model, which we call the QoS Broker, results in a new way to structure multimedia systems.

Before describing the QoS Broker, we examine some other structuring models (e.g., those shown in Figures 1a and 1b), to introduce ideas, and to point out their deficiencies.

## 1.1 Problem Description

Our experience with multimedia applications, as well as other interactive applications, has shown that the module structure of the system is an important factor in its ease of programming, extensibility, and eventual acceptance. This structuring is as much an aesthetic challenge as it is a technical challenge, as the right balance between flexibility and structuring is not always clear.

Similar structuring problems have been addressed before. The UNIX *termcap* terminal interface library is worth examining as a case study. Previous to the use of termcap, specific terminal control sequences ("escape sequences") were embedded in interactive applications in order to use features of advanced displays. Unfortunately, this led to considerable redundancy, non-standard terminal handling code, and problems with portability. The termcap library and database provide a solution

to this problem.

When a terminal type is defined, say "`vt100`" for the DEC VT100 display terminal, an entry is defined in the termcap database, which is found in a well-known location, such as `/etc/termcap`. The current terminal type in use is set using the UNIX shell's environment variables, e.g., `TERM=vt100`. When the termcap library routines are invoked, the library routines (such as `clear()`, to clear the display) use the terminal type information to generate the proper escape sequence. Thus, the database and library serve to customize a general interactive application to a specific terminal type. Using the termcap scheme allows an application to be written independent of the terminal type. There is a limited capability to query the database for terminal features of interest to the application. A similar structure has been employed in the X Window System.

This model can be viewed as providing an application interface using extensive information for customization to specific operating environments. Some characteristics of the networked multimedia environment are different from terminal-handling and hence must be addressed by any proposed solution:

1. The network and operating system are ACTIVE (first class) resources, unlike a terminal which is PASSIVE. The managed resources may signal changes of state, implying that communication must be bidirectional. The information used for decision-making cannot be entirely contained in a static database, although such databases can support event service.

2. In addition to dynamics, the network and operating system are SHARED resources, unlike a terminal which is DEDICATED to a single user. Thus, in addition to managing a device, the management of other entities must take place. Delay-sensitive applications (as most multimedia applications are) must be protected from the dynamics induced by sharing.

3. Finally, multiple devices must be managed, which adds considerable complexity. This stems from the fact that each device has different requirements and allowable tradeoffs, which may be in conflict for a given application. Management must model this complex system and devise a strategy which MUTUALLY satisfies device and other requirements to the satisfaction of the application.

We have defined and implemented a prototype architecture which addresses these issues. Resource orchestration plays a central role in the architecture. In a distributed environment, its

functions include resource orchestration at individual hosts (the end-points of the distributed networked system), and resource orchestration among these hosts (e.g., at the network switches or intermediate network nodes).

The *QoS Broker*, shown in its role as an intermediary entity (Figure 1c), orchestrates resources at the hosts, and cooperates with network resource orchestration. It accomplishes this with resource databases and a variety of different services to achieve a balance (a "deal") among application requirements and multimedia I/O devices, as well as network and operating system resources.

We describe the QoS Broker concept in the next section, Section 2. Section 3 presents the broker architecture, as well as protocols and services used by the broker. Section 4 gives an overview of our implementation, the experimental setup of the telerobotics application, and limitations of the prototype broker stemming from the current implementation platform. Section 5 concludes the paper, identifying some problems which must be solved for a comprehensive realization of a QoS broker.

### 1.2 Related Work

The majority of work on QoS has focused on resource management in *networks*, where resources, such as bandwidth and buffer space for queues, are allocated and controlled. This allocation and management is in contrast to the dynamic sharing of resources achieved with TCP/IP. Network resource management (when guarantees are required) uses an *admission* service to determine if resources are available for a request, e.g., at the intermediate network switches. Such admission control mechanisms are presented in [11], [13]. Protocols for reservation (RSVP) [5] and administration (RCAP) [6] of network resources have been developed.

Network resource management is performed, for example, in the protocol ST-II[9], intended for multimedia applications. An important component of resource management systems is translation of resource specifications between consecutive layers of the network protocol. For example, in [7] a translation between ATM Adaptation Layer (AAL) and ATM resource description is presented.

In order to support local multimedia services, management of system resources, such as the CPU and disks, is a necessary part of end-to-end design and requires real-time services [4]. Several real-time extensions of common operating systems have been introduced, e.g., for Mach, AIX, Solaris, and IRIX. One important goal of such extensions has been to improve support for 'delay sensitive' multimedia applications.

Some systems have *partially* orchestrated multiple elements. For example, the Lancaster system [8] orchestrates the behavior of the network components. The computer music system of Anderson [14] orchestrates the relationships of users and devices. Schooler and Casner's system[12] orchestrates some parts of end-to-end communication.

There has been little orchestration between the operating system (OS) and network resources and their management structure. There is even less orchestration among all three types of resource (multimedia devices, OS resources, and network resources). Yet, as we have shown through experiments [1], the behaviors of these components are to a large degree interdependent in a networked multimedia system when examined from an application perspective. This suggests that the orchestration of these resources be integrated in a single entity for multimedia applications.

# 2 QoS Broker Concept

To provide applications with end-to-end guarantees, network resource management alone is not sufficient. This is particularly true when end-points become more sophisticated, e.g., workstations with rich device support, multiprocessing and multiple users. This suggests a need to balance resources among application, network and OS at the end-points, and between end-points and the network. We begin with a model for the communication components at the end-points, and specify the role for resource management entity.

## 2.1 End-Point Communication Model

Our end-point communication model, shown in Figure 2, includes two major communication components: an *application subsystem* and a *transport subsystem*. The application subsystem provides functions such as: (1) multimedia call management, (2) I/O device management to move data from/to multimedia devices, and (3) synchronizing the received media and its delivery to the application. The transport subsystem provides connection management, error correction, rate (bandwidth) control and movement of data to and from the computer/network host interface. The application subsystem is assumed to be embedded in a less-privileged OS *user space* protection domain. For reasons of data security, resource scheduling, and access to network interface hardware, the transport subsystem is embedded in a more protected OS *system space*, as shown in Figure 2. All three components work with resources through component-specific interfaces, and thus the functional and modular division involves some information-hiding as well.

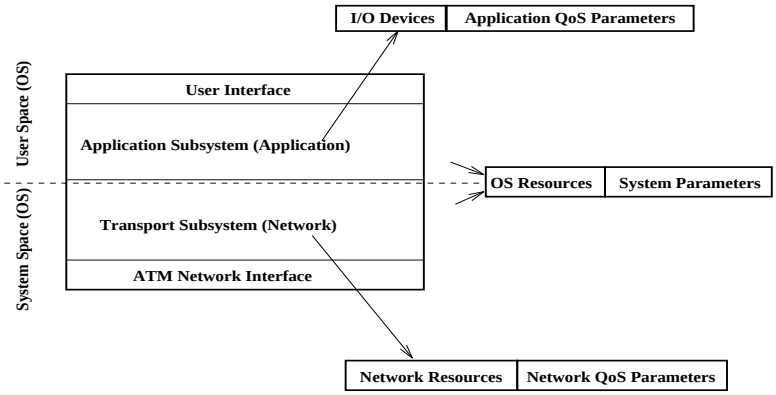


Figure 2: *End-Point Communication Model with Layer and Resource Specification*

The *application's resources* are represented as I/O devices (local or remote) comprising the interfaces to a multimedia system. This representation allows an analysis of the communications flows through which the devices are coupled together and orchestrated. The resources are described (i.e., *parameterized*) through *application QoS parameters*. The parameters consist of *media quality* descriptions for the specific *media characteristics* of each device, such as sample size (e.g., height, width, and color specification in a video stream), and the media sample rate and priority/criticality. The parameterization also includes an application-oriented specification of the *transmission characteristics* requirements for end-to-end delivery (e.g., end-to-end delay bounds) and communication topology (e.g., peer-to-peer, peer-to-group) and any *media relations* among media (e.g., synchronization). The application QoS parameters are specified for all involved devices, both input and output.

The *network resources* are bandwidth, buffer space for packet/cell queuing, and time slices. We find it convenient to specify the time slices by their side-effects on data, e.g., by the round-trip delay and interarrival delay (jitter), since the underlying network may have a variety of means to meet these specifications. Network resources are specified as *network QoS parameters*. The parameters are specified in three domains of interest. In the first domain are basic parameters such as packet size and packet rate. In the second, we have environment-sensitive parameters such as interarrival delay, round-trip delay and packet loss rates. The third domain of interest is the specification of the overall communication requirements, using parameters such as packet ordering, communication topology, cost and priorities.

The *OS resources* are processing times required for tasks, secondary storage, and memory buffer

requirements. The OS resources are needed at the application subsystem level and at the transport subsystem level tasks to handle input/output of media and sending/receiving connections. All OS resources are parameterized through *system parameters*.

All parameters are stored in *profiles* (databases). Parameters are specified in a form consistent with their use (details are in [3]). For example, delay parameters are specified as a range: <*expected value, worst value*>. Jitter can be accommodated by specifying task processing times as a triple < *best processing time, average processing time, worst acceptable processing time*>. Hence, the task processing time will be accepted if it is in the interval bounded by the pair of values <*best value, worst value* >.

## 2.2 Role of the QoS Broker in a Communication System

The QoS Broker entity is an end-point resource manager which: (1) orchestrates resources, needed for tasks in the application and transport subsystems, at the end-point, (2) negotiates with network resource management, and (3) negotiates with the remote QoS Broker. To orchestrate resources and support the decision-making process (the brokerage process), the broker must have access to all profiles. Conceptually, the profiles could be used by the Broker to automate the choice of tradeoffs possible in a system. For example, the burstiness in the network could be accommodated by the operating system with an elastic buffer *if* the additional end-to-end delay resulting from inserting the buffer was acceptable based on the profiles. Jitter bounds could be met, within acceptable bounds, by (1) finer granularity clocking in the real-time OS; (2) tighter jitter bounds on the network; or (3) careful schedule coordination by the real-time OS to ensure that process scheduling and packet arrival closely coincide. The profiles define the space in which decisions can be made, as well as preferences within that space.

Hence, the QoS Broker is an additional component in the communication architecture (as shown in Figure 3), which prepares the communication system for guaranteed transmission of multimedia streams. The broker presumes that transmission tasks in both subsystems are parameterized and behave according to the deal the broker negotiated. The negotiated deal is encoded in a QoS profile.

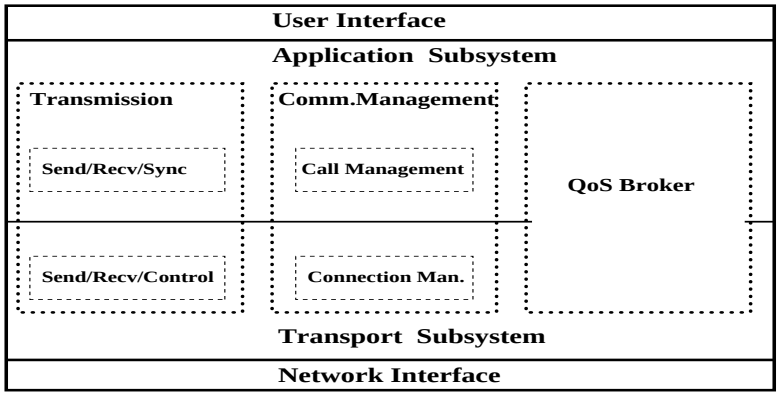


Figure 3: *QoS Broker viewed in the End-Point Communication Model*

# 3 QoS Broker Design

In human affairs, brokers are engaged as specialists in a particular type of negotiation. They serve as intermediaries, using specialized knowledge, and ideally work towards obtaining a mutually desirable outcome between some set of buyers and some set of sellers. This often involves converting detailed requirements of individual parties into terms usable for an agreement between parties. We have tried to emulate this situation in our QoS Broker, where the "buyers" are the end-point sites which want to 'buy' resources from other remote sites of the networked multimedia system and the "sellers" are the remote sites which have their resources for 'sale' during the brokerage process. One could view the buyer as an 'initiator' of a brokerage process.

In our design the buyer and seller roles are filled by protocol entities called *the broker-buyer* and *the broker-seller*. The *buyer* wants to negotiate resources for a customized connection, hence the broker-buyer includes the following activities (as shown in Figure 4): **(1)** orchestrates the local resources, **(2)** gathers information about available network resources in the intermediate elements (such as switches) provided by network resource management, and **(3)** gathers the resource information from the broker-seller at the remote side. The *seller* is located at the remote side. It orchestrates the remote end-point resources, and lets the broker-buyer know what resources it can offer for establishing the specified (customized) connections. The broker-seller's activity is shown as **(4)** in Figure 4.

The brokerage is integrated into a distributed environment as follows: broker communications are split into two parts: *sender* and *receiver*. The broker-sender is responsible for the resources needed for outgoing calls/connections and the broker-receiver is responsible for the resources of

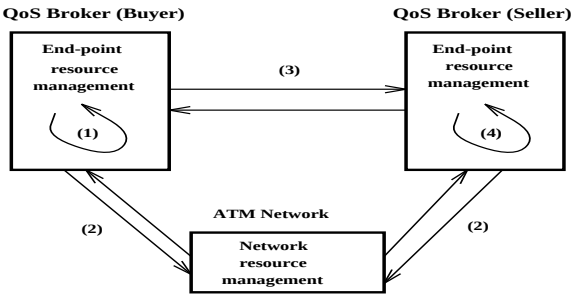


Figure 4: *QoS Broker Design*

incoming connections/calls. Two *operational modes* can be derived: *sender-initiated brokerage* and *receiver-initiated brokerage*.

The first mode occurs when the buyer is the sender requesting resources from the receiver (a seller whose resources are for sale) to establish a unidirectional multimedia call with end-to-end guarantees from the sender to the receiver. This is a *sender-initiated brokerage*. An example of such a situation is a conference set up of audio/visual connections from the speaker to the listeners. Another example is a teleoperation application such as remote camera control. In this case the sender (user) initiates a resource brokerage for opening an unidirectional connection from the sender to the receiver (camera) in order to send commands to control the position of a camera.

The second mode is represented by the situation when the buyer is the receiver who wants to negotiate resources with the sender (seller whose resources are for sale) to establish a unidirectional multimedia call with end-to-end guarantees from the sender to the receiver. From the communication point of view this is a *receiver-initiated brokerage*. Examples are video-on-demand or pay-per-view cable-TV applications where the viewer wants to watch (receive) a movie from the source originator (e.g., video database). Here, the receiver must initiate the unidirectional connection setup from the sender to the receiver.

An expansion of the QoS Broker where both concepts (brokerage and communication) are integrated is shown in Figure 5.

The broker includes two main entities: broker-sender and broker-receiver. The *broker-sender* entity manages the resources (application QoS, system and network parameters) for input/outgoing connections and the *broker-receiver* manages the resources for output/incoming connections. Inside of each of these entities reside the *buyer* and *seller* protocols which are activated depending on which side (sender or receiver) initiates the resource brokerage. The actual brokerage is carried out

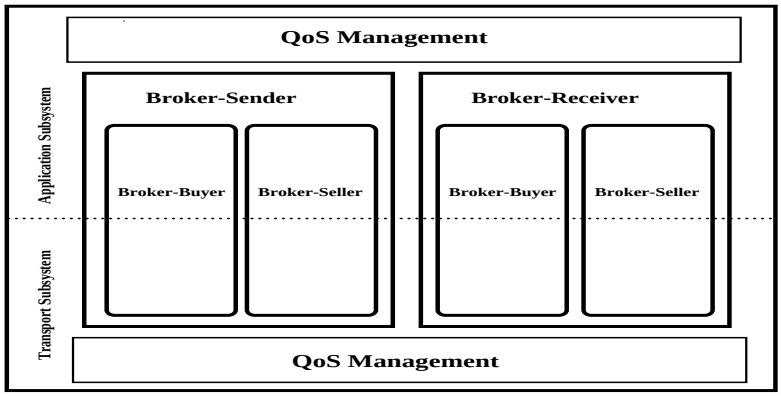


Figure 5: *QoS Broker Architecture*

between the buyer at one side and the seller at the remote side as shown in Figure 4. Further, the broker has a *QoS management* entity which oversees the handling of profiles at the end-point for I/O and outgoing and incoming connections. Returning to our end-point communication model of Figure 2, the buyer and seller have two parts, an *application part* and a *transport part*. The application part orchestrates resources in the user space, such as interactions between processes required by the multimedia devices and applications structure. The transport part manages resources shared by lower layers of protocol stacks. The cooperation between the application and transport parts is done through layer-to-layer communication in the broker protocol. Resource synchronization is done through the system parameter database. Interaction between the transport part of the broker and network resource management ensures global orchestration.

In each mode, the communication between the remote brokers draws on other specialized services (e.g., admission) and protocols (e.g., negotiation). Figure 6, which is necessarily a bit "busy", gives a more detailed illustration of communication (i.e., the process shown in Figure 4) between the broker entities and the underlying network in the context of our end-point model in sender-initiated brokerage mode.

In the next few subsections we detail protocols used by the broker-buyer and broker-seller, and briefly describe some new services. More detailed discussion of the services can be found in [3].

### 3.1 QoS Broker Protocol

Since it crosses layer boundaries, the broker protocol incorporates several types of communication: *layer-to-layer*, *layer-to-OS*, *peer-to-peer*, *peer-to-group*, and *group-to-peer* communication. Commu-

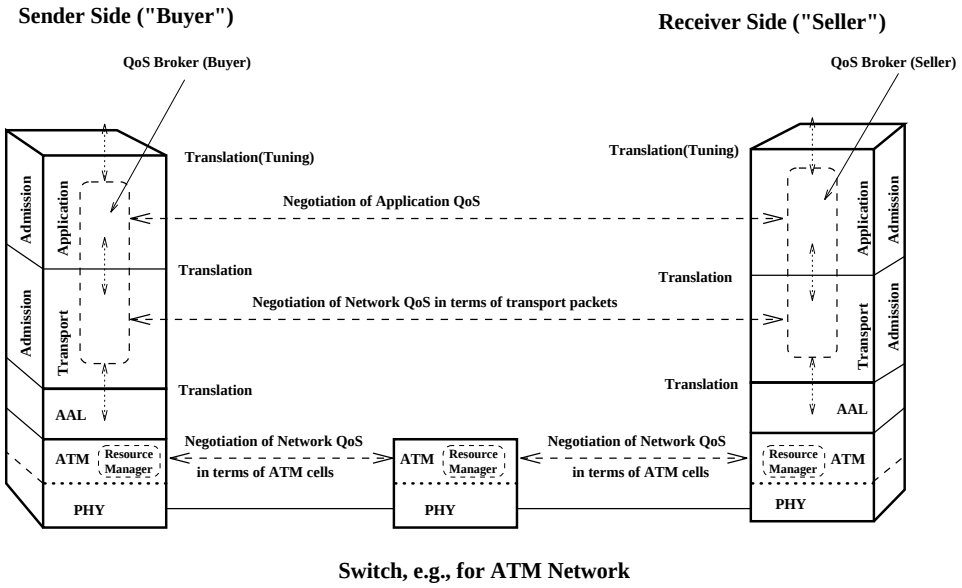


Figure 6: *QoS Broker Communication inside and between End-Points (ATM example)*

nication types are performed by different services. For example, layer-to-layer communication such as the human-to-application communication is supported by the *tuning service*, described further below. The application-to-transport communication is provided by the *QoS translator*. Peer-to-peer, peer-to-group and group-to-peer communication setup resource requirements are distributed by the *negotiation* service among the remote sites. For instance, the *negotiation of application QoS* provides communication among the distributed end-point application entities with respect to their application QoS. Likewise, the *negotiation of network QoS* employs communication among the distributed end-point transport entities. The layer-to-OS communication uses an *admission service* for the decision-making process for OS resources required of the particular layer.

*Signaling* among protocol services results in one of three responses: *'accept'* when expected resources can be reserved on the way to the remote side and allocated on the return, *'reject'* when required resources cannot be provided or the broker-buyer signals timeout because something happened in the network (e.g., the QoS/reservation request is spliced into an existing multicast tree and may never reach the seller – this case is handled as if no network resources were available), and *'modify'* when resource requirements must be relaxed, but are still within bounds.

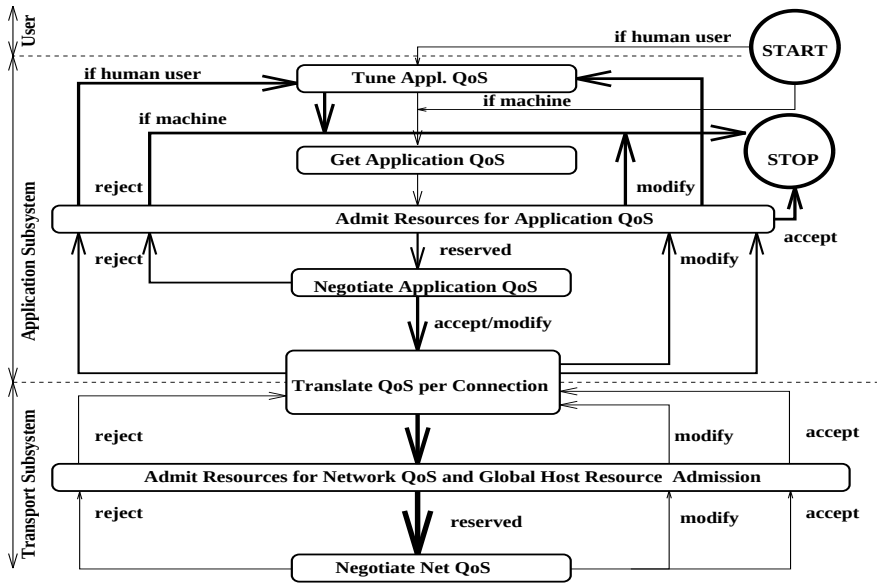


Figure 7: *QoS Broker Protocol - Buyer*

### 3.1.1 QoS Broker Protocol - Buyer

The *broker-buyer protocol* (for which a state diagram is given in Figure 7) is initiated by the input of application QoS requirements.

The input of application QoS requirements can be performed either by the *tuning service* when audio/visual services are specified, or by textual input. The tuning service interactively aids the user in selecting QoS characteristics by e.g., viewing a test video with image rate, image height, image width, color, etc. specified through manipulation of a slider (audio might use a test sound passage with specified amplitude, etc.).

An example graphical user interface (GUI) is shown in Figure 8. This diagram illustrates a video stream being displayed and a set of "sliders" which control the quality of the video display by adjusting the playback parameters such as frame rate and picture size. The positions of the sliders encode selections of application QoS which result in the required user QoS. The user selections (e.g., image height, image color) are in this fashion translated into application QoS parameters such as frame size, frame rate and others. For instance, the image height, width, and color (256 colors can be encoded in 8 bits/pixel, etc.) influence the frame size as follows: $frame\ size = width\ (pixels) \times height\ (pixels) \times color\ resolution\ (bits/pixel)$ if the image is not compressed.

The translation between user QoS and application QoS is non-trivial and it is still an open

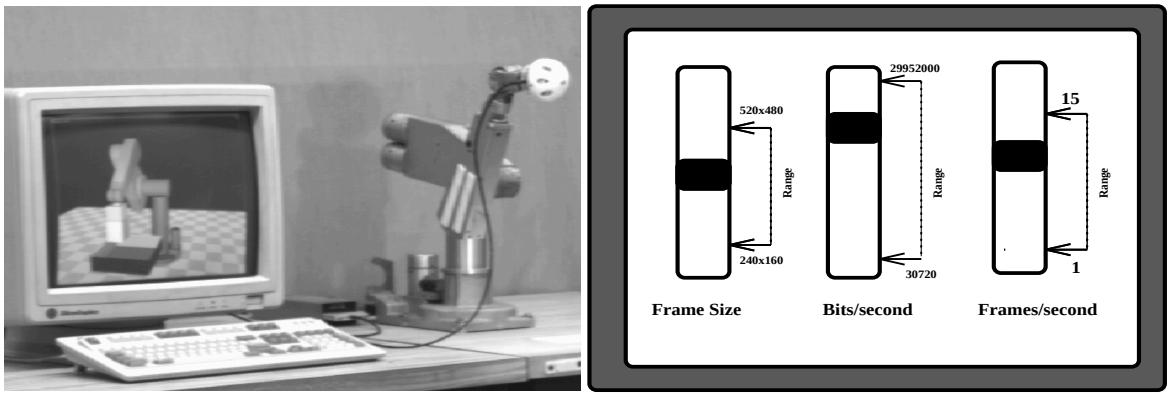


Figure 8: *Graphical User Interface*

research issue, largely because the perceptual issues are not completely understood. We expect that future research will focus on the interplay between Computer-Human Interface (CHI) research and the support mechanisms for multimedia display and interaction. The GUI-based tuning service prototype, while simplified relative to the requirements of a full-scale system, allows us to focus our energy on the system architecture, and in particular the broker concept discussed in this paper.

The application QoS requirements are mapped into resource requirements for the local OS. The broker negotiates with the OS, using an *admission service* implemented at the application level. The OS is assumed to have real-time capabilities such as real-time priorities, fine granularity timers, and locking mechanisms for specified memory buffers. Without these real-time capabilities, a prediction of a deterministic behavior during the negotiation is impossible to achieve. As we discuss below in Section 4.3.1, such guarantees are difficult even where some facilities are available from the operating system.

The admission service currently assumes that application protocol task processing times and buffer space requirements are known *a priori*, and available in the system parameter profile before admissions decisions are made. Admission service at the application level performs two tests against temporal resources: (1) a *local schedulability test* [1] to see if the tasks can manage I/O from media devices within the required time bounds; (2) an *end-to-end delay test* to see if tasks can meet the specified end-to-end delay upper bound.

[1] Schedulability tests should be part of OS resource management. In current OS this function is inadequate. For now, it is part of the admission service in the broker. Details of the schedulability tests for periodic multimedia streams in a workstation environment can be found in [3].

Once local resources are reserved, then *negotiation at the application level* with the remote broker-seller occurs. We separate this negotiation from the negotiation of network QoS for two main reasons: (1) holding network resources is expensive, and (2) network resources are shared. The negotiation at the application level concerns the seller's ability to accommodate the requested multimedia characteristics; this depends on the seller having available appropriate I/O devices, processing capacity and storage space. Until this is determined, appropriate bandwidth allocation cannot be made. Furthermore, this negotiation can also be used to exchange additional application information, for example, participant identity verification in closed conferences, or an image of a robot work space to prepare for teleoperation. Application QoS and other application specific information can be exchanged several times over non-real time connections before any request for guaranteed network resources is made.

If the answer is 'accept', or 'modify', the broker-buyer in the application subsystem initiates the request for network QoSs and their resource reservation/allocation, which correspond to the multimedia application QoS. This results in several steps, which the broker-buyer in the transport subsystem must perform:

First, the application QoS requirements are translated into network QoS requirements using the *QoS translator*. As we pointed out in the introduction, translation between application QoS and network QoS is *bidirectional*. The translation is done in one or more steps, depending on the number of media which need to be transported with different network QoS. The *application QoS profile* (introduced in Section 2.1) includes the entire multimedia networked application description (input and output), hence the translator must choose input medium by medium and map it into outgoing transport connections with appropriate network QoS parameters. The broker stores the network QoS parameters in the *network QoS profile*. Translation includes mapping (1) between one medium and one connection, which means that all samples of the medium are transported through the specified connection; (2) between two or more media of the same quality and one connection, which means that the media share one common network connection; (3) between one medium and two or more connections, which means that the medium has samples of different importance, which are mapped onto different connection (e.g., MPEG compression creates video medium with I-frames, which are the most important and have different media quality than less important P-frames and B-frames). As with the other translations, this is a difficult problem in the most general case, as the traffic is not guaranteed to be smooth. Our use of uncompressed

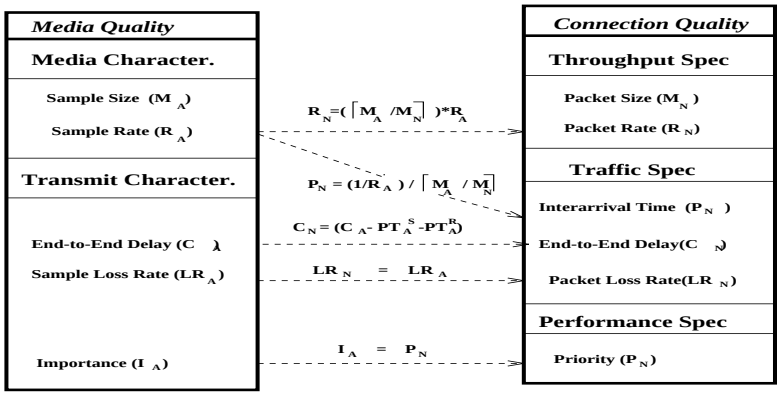


Figure 9: *QoS Translation Relations (Example for Strong Periodic Streams)*

video in our prototype implementation (discussed in Section 4) allowed us to avoid some of these problems. However, the correlation and burst properties of a more realistic video stream make a robust and accurate translation a challenging research problem. The straightforward method for handling this situation is to accommodate the imprecision in translation by conservative resource allocations, e.g., by leaving some headroom. As further study refines our capability to translate, the resource allocations can be made less conservative.

Consider a concrete example. The broker-buyer wants to negotiate for a unidirectional connection for robotics data to the broker-seller. The user specifies application QoS parameters such as: sample rate $R_A$ of 100 samples/second, sample size $M_A$ of 64 bytes, loss rate $LR_A$ of 1 sample/minute, and maximum end-to-end delay $C_A$ of 20 milliseconds. The QoS translator translates the provided parameters into network QoS parameters using the translation relations in Figure 9. For instance, assuming a network packet size $M_N$ of 48 bytes, the translation results in a packet rate $R_N$ of 200 packets/second, an interarrival time $P_N$ of at most 5 ms between two packets belonging to the same sample and at least 5 ms between two packets of two consecutive samples, and a loss rate $LR_N$ of 1 packet/minute (when one part of a sample gets lost the whole sample needs to be dropped). Further, the QoS translator retrieves the processing times of application tasks for transforming robotics data (read application sample $PT_A^S = 0.148$ ms, write application sample $PT_A^R = 0.802$ ms). These actual times were measured in advance, and exchanged during application QoS negotiation. The required end-to-end delay $C_N$ for a sample at the transport level is 19.05 ms.

Second, after translation, *admission* for the transport subsystem is invoked. The broker maps

the network QoS parameters to spatial and temporal resources needed by the transport tasks. This translation includes mapping from the <*packet priority, packet loss rate*> pair to the task priority which handles the priority queue, dependencies between packet size and task duration, and the < *end-to-end delay, packet rate, packet size* > triple and task period. Further, the admission service at the transport level tests both the network resources such as end-to-end delay and bandwidth, and schedulability for all end-point resources. They must be jointly managed because they share resources such as the processor.

Third, if the admission at the host is successful, negotiation of per connection network QoS parameters is initiated by the broker. This negotiation relies on: (1) *translation* between network QoS in terms of transport packets and underlying units such as ATM cells, and (2) *network resource management* in the network.

Finally, the broker waits for the replies from the network resource management and broker-seller's transport level. These responses are translated back to the application QoS, so that the user understands which media at what quality will be transmitted.

#### 3.1.2 QoS Broker Protocol - Seller

The *broker-seller's protocol* is similar to the broker-buyer's protocol. The differences are in the order of using the services:

First, the broker-seller waits for the negotiation request from the remote broker-buyer before performing admission. The negotiation request contains the sending application's QoS input parameters. The broker-seller compares these parameters against its own application QoS output parameters. If they match, the admission service in the application subsystem is invoked. The admission answer is sent as a negotiation response.

Second, after positive negotiation of application QoS, the broker-seller waits for network signaling in the transport subsystem (no translation or global admission is necessary at this point) until the network management signals the broker-seller on behalf of the broker-buyer about the availability of network resources.

Third, a signal from the network resource management initiates the global admission service in the transport subsystem part of the broker-seller. The answer from this admission service is sent back to the network resource management as well as to the application part of the broker-seller, using reverse translation (from network QoS to application QoS parameters) in order to

allocate/relax/free the application resources, depending on the availability of resources.

### 3.2 Dynamic Change of State

In our system, we view the communication between entities managed by the QoS Broker (OS scheduling, network bandwidth allocation, etc.) as logically component-to-component. Here, component-to-component does not refer to layering, but rather to the ability to mutually signal a change of state. For the QoS Broker, as an example, consider the situation where the user requests a change in presentation (e.g., a larger video window). This change in presentation propagates to the application, which will require, at the least, more memory and more processing cycles to manage the larger bitmap. These changes are then propagated to the OS scheduler, which must allocate more of each CPU quantum to the process (if possible), and to the network, which must allocate more communications bandwidth if the video source is remote. As the locus for negotiation and management of system resources, the QoS Broker interprets the requirements as they cross boundaries, and facilitates the negotiation process.

Now, as it is less familiar, and a bit trickier, let us consider *renegotiation* when it is triggered at a point other than a user process. Consider, for example, an application which supports a window of live video, which is delivered through a network with intelligent call setup, e.g., a switched ATM LAN. As the network becomes overloaded, it becomes increasingly difficult to support connections with limited statistical multiplexing potential, such as those with hard bandwidth allocations. Thus, it would be desirable for the network to request a reduction in the guaranteed bandwidth provided to an application.

This would be done in the following manner. The port controller associated with the application's ATM virtual circuit would send a message to the host networking subsystem (typically embedded in the operating system). This message would signal the QoS Broker that a renegotiation has become necessary, and the Broker initiates communication with the necessary system components. While the details will vary depending on the programming environment, an easy to imagine scenario is one using the UNIX `signal()` facility to initiate exception-handling by the application, which would note that a broker-initiated renegotiation was to be undertaken and a change of application QoS had to be performed. Further, this change would trigger a change of user QoS which implies that either a bidirectional mapping between application QoS and user QoS exists, or a range of application QoS parameters exists which satisfies the user-specified QoS.

In systems where interrupting the application would be undesirable, other mechanisms could be used (e.g., application-polled message queues), or the Broker could be programmed to seek solutions without interrupting the application, e.g., employing a more aggressive data compression scheme, or automatically resizing the video window. Thus, the bidirectional capability of the QoS Broker can be exploited for high overall system performance.

### 3.3 The QoS Broker in a Group Brokerage Situation

The QoS Broker can also work in a group brokerage situation if the application requests a peer-to-group communication in the sender-initiated mode or group-to-peer communication in the receiver-initiated mode. The type of group communication is specified in a media quality parameter – *communication topology*, and translated into traffic characteristics of network QoS parameters.

#### 3.3.1 Peer-to-Group Communication

The application specifies in the input application QoS parameters the communication topology as the peer-to-group communication type. Further, it provides information for addresses such as from whom the brokerage request is, to whom the request should be sent (the group address), and response addresses to get the result of the brokerage from the network and the sellers. The broker-buyer at the transport level maps these addresses to network addresses (e.g., VCI) and waits for responses at these addresses from the network and the group of sellers. The next step of the broker-buyer is to multicast the network QoS parameters to the group. Here, the broker relies on the multicast capabilities of the network protocols, i.e., the broker itself does not have multicast capability, nor do the end-point transmission protocols at the end-point (note that IP multicast may allow a rethinking of this assumption). The sellers at different remote sites proceed according to the broker-seller protocol and send their brokerage replies over the specified response address. The broker-buyer in the peer-to-group situation analyzes the received QoS parameters and chooses the highest quality the network and any of the receivers can support. It then responds to the group with QoS parameters it will send to the group. At this point, the sellers who can't service the specified quality must adjust their own capabilities at the network interface. If, for example, 10 frames/second video will arrive at the remote end-point, but the end-point can support only 5 frames/second, then the receiving protocol will drop every other frame and provide only 5 frames/second to the application. The broker-seller needs to adjust all resources to this

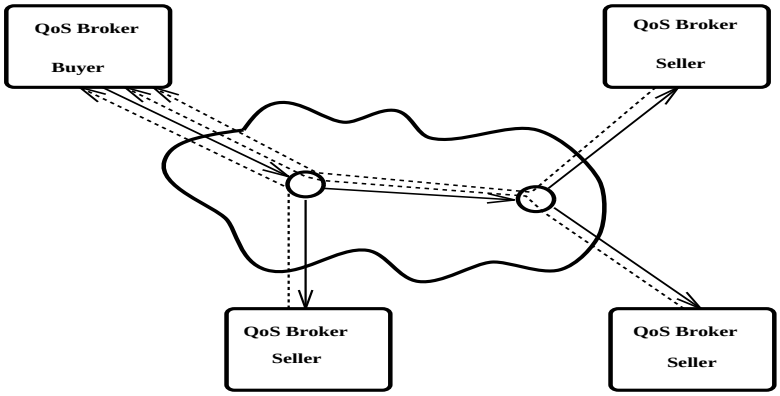


Figure 10: *Distribution of Negotiation Messages in Group Brokerage Situation*

specification. Hence, it is the responsibility of the *receiver* to adjust to the quality it can handle. The negotiation paths in this situation are shown in Figure 10.

The advantage of the negotiation with every seller is that each seller can report to the buyer its own deal and the broker-buyer can make a better decision on what kind of quality and guarantees can be supported. Remember, the broker wants to negotiate end-to-end guarantees. The disadvantage is that the number of connections for negotiations increases proportionally with the number of the group, if for every negotiation two unidirectional connections are used. However, there are many possible optimizations to decrease the connection space. For example, one possibility would be to provide *group reporting channels* in the network which could be setup during the multicasting of the first negotiation message from the broker-buyer. Sites could share these for negotiation and renegotiation.

### 3.3.2 Group-to-Peer Communication

The application specifies in the output application QoS parameters the communication topology as group-to-peer communication type, which means that the application would like to receive multimedia streams from multiple remote sites (e.g., receive and display several TV programs in separate windows on the screen). The application needs to specify to the broker-buyer information for addresses such as from whom the brokerage request is, to whom the request should be (group address), and response addresses. The specification is essentially the same as in peer-to-group communication. Similarly, the distribution of negotiation messages can be done as in Figure 10. The difference is in the decision making process when the responses arrive. In this case, the broker-

buyer needs to plan to allocate resources for several incoming connections according to the different QoS parameters from the broker-sellers.

### 3.4 QoS Broker and Underlying Network Resource Management

As described above, the QoS Broker is an end-point resource management entity based on QoS specification which helps to negotiate QoS parameters at application and transport level, and admit and reserve/allocate end-point resources. We assumed there is adequate network resource management, with network admission and reservation support at intermediate nodes to make a global decision about end-to-end guarantees. We provide some examples of existing network protocols and reservation schemes which could be used as the underlying support for QoS Broker.

If the buyer is in sender-initiated brokerage mode, then the underlying reservation and QoS supporting establishment protocol must be sender-oriented. It means that the network reservation protocol starts to test for admission and reserve network resources from the sender, along the path to the the receiver end-point(s). When the negotiation response from the seller(s) comes into the network, the network compares what kind of resources the end-point can support and it allocates reserved resource, relaxes or releases the resources. Many elements of such a network reservation establishment protocol are implemented in the Tenet Real-Time Channel Administration Protocol (RCAP) [6].

If the buyer is in receiver-initiated brokerage mode, then it relies on a receiver-oriented admission and reservation scheme. The resources are tested for admission and reserved on the path from the receiver to the sender and allocated, relaxed, or released on the way from the sender to the receiver. Some elements of such a network reservation establishment protocol can be found in the design of the Resource Reservation Protocol (RSVP) [5].

## 4 QoS Broker Implementation

We have implemented an experimental prototype of the broker. This gave us an initial testbed for our ideas and allows refinement after testing with our applications.

### 4.1 Implementation of QoS Broker

The first version of the broker, as well as our experimental application and transport protocol stack above the ATM device driver, is implemented as user processes using procedure interfaces as

| Name | Function |
|---|---|
| `TuneAppQoS` | Communicates Application QoS between User and Application Layers |
| `SetAppQoS` | Sets Application QoS Parameters in Application QoS Profile |
| `GetAppQoS` | Retrieves Application QoS Parameters from Application QoS Profile |
| `SetAppTaskParam` | Sets individual parameters for Application Task in System Parameter Profile |
| `GetAppTaskParam` | Retrieves Task Parameters from System Parameter Profile |
| `SetAppSysParam` | Sets System Parameter Description of Application in System Parameter Profile |
| `GetAppSysParam` | Retrieves System Description of Application from System Parameter Profile |
| `AdmitAppQoS` | Admits Application QoS Parameters |
| `NegotiateAppQoS` | Negotiates Application QoS between Application Sender and Receiver |
| `QoSTranslator` | Communicates between Application and Transport Layers |

Table 1: *Service Procedures used by the QoS Broker Entity*

stubs. Table 1 shows the application services and the QoS translator procedures used by the QoS Broker. The set of service procedures for the broker's transport part (e.g., `SetNetQoS, admitNetQoS, NegotiateNetQoS`) is equivalent to the set of the application service procedures (`SetAppQoS, admitAppQoS, NegotiateAppQoS`) shown in Table 1 above. The services such as admission, negotiation, and translation, which the QoS Broker uses, form a *service kernel* [3]. The transport protocol and the transport portion of the broker will be kernel-resident in our next implementation.

In the current implementation the broker's Application Programmer Interface (API) interface is:

```
QoSBroker(QoS parameters, Additional parameters, Notification, Side, Direction)
```

The five brokerage parameters have the following roles:

- *QoS parameters*

  The `QoS parameters` field represents either input or output application QoS parameters. The QoS parameters are stored in profiles, hence the QoS parameter field includes the file descriptor of the application profile.

- *Additional parameters*

  `Additional parameters` include additional application-specific request/response information (non-QoS parameters) which are exchanged among the remote sites, for example, conference participants images in a closed conference for identity verification, or an image of a work

space in the case of teleoperation.

- *Notification*

  `Notification` includes the response about the QoS parameters acceptance. The parameters can be: *accept*, *modify* or *reject*.

- *Side*

  `Side` specifies if the buyer (BUYER) or the seller (SELLER) should be invoked, i.e., the role in the brokerage protocol.

- *Direction*

  `Direction` specifies to the QoS Broker if the sender (SENDER) or the receiver (RECEIVER) should be invoked.

  If the `direction` is *SENDER*, and `side` is *BUYER*, input application QoS parameters are entered from the application to the QoS Broker - Buyer, the buyer protocol starts and the returned value is accepted, modified or rejected input application QoS parameters.

  If the direction is *SENDER* and the side parameter is *SELLER*, it means that the input application QoS parameters were passed to the broker, but the seller waits for contact from the buyer about the output application QoS parameters. The returned value is accepted, modified or rejected input application QoS parameters.

  If the `direction` is *RECEIVER*, and `side` is *SELLER*, output application QoS are passed to the broker, and the seller waits for contact from the buyer according to the broker-seller protocol. Returned values are accepted, modified, or rejected output QoS values.

  If the `direction` is *RECEIVER* and `side` is *BUYER*, the output application QoS are passed and the buyer protocol starts working with output application QoS parameters. The returned values are accepted, modified or rejected output application QoS parameters.

The *negotiation protocol* of application QoS parameters is split from the negotiation of network QoS - a network control connection is used by the broker for negotiation of application QoS; for negotiation of network QoS parameters another control connection is used. This entire interaction requires only a few milliseconds in our current system.

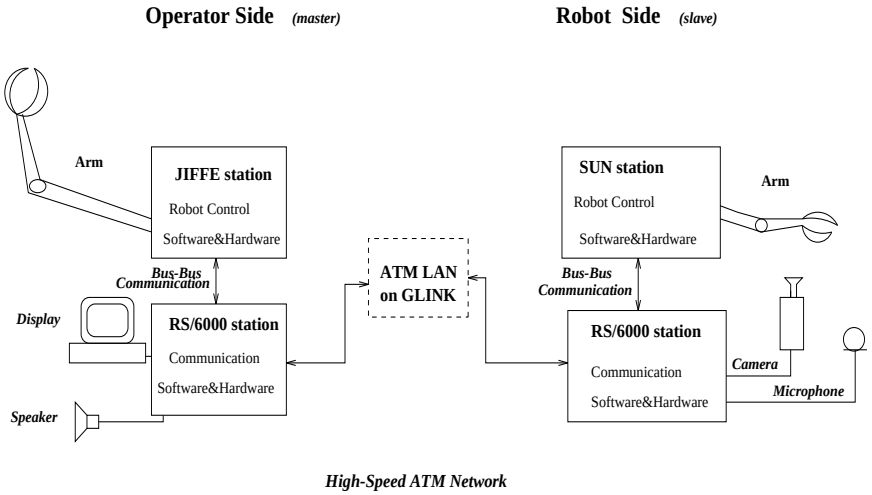


Figure 11: *Telerobotics System Configuration*

The current broker requires a *preprocessing phase*. This means that the set of tasks, needed in the application and transport subsystem protocol stack, and their processing times have to be filled into the system parameter profile during the installation phase of the protocol stack. This is not an optimal implementation solution, but current operating systems do not support guaranteed services, especially, they do not provide resolution of scheduling conflicts based on QoS. Hence, the application cannot explicitly control the CPU and it is not possible to predict and constrain how much of the processor will be allocated to application and transport protocol tasks.

## 4.2 Experimental Setup for a Telerobotics Application

We verify our prototype of the QoS Broker with a *teleoperation* application. Our teleoperation application involves remote control of a robot (telerobotics). Telerobotics/teleoperation application is non-trivial and has challenges distinct from teleconferencing. Our test configuration is shown in Figure 11. *Teleoperation* allows a remote operator to exert force or to impart motion to a slave manipulator. The operator experiences the force and resulting motion of the slave manipulator, known as "kinesthetic feedback". An operator is also provided with visual feedback and possibly audio feedback as well.

The media, *tactile and sensory data*, *audio* and *video*, have greatly differing network QoS parameter requirements. For example, the sensory data has high reliability requirements (1 packet/minute can be lost, and no two consecutive packets can be lost), and strict constraints on end-to-end delay (20ms), but relatively low throughput demands (1 sample is 64 bytes and the sample rate is 50 sam-

ples/sec). On the other hand, the video data have looser delay (200ms) and reliability requirements, but relatively high throughput requirements, even with compression. This application has dynamic changes in its requirements over its execution because the physical information changes, the robot arms are mobile, they may obscure a camera while moving. Changes in physical information may result in renegotiation of requirements among remote sites. The complex and dynamic requirements of the telerobotics application provide an ideal 'real-world' setting to test the brokerage concept.

The network solution employs point-to-point links to a high-speed ATM switch in an ATM LAN. The end-point workstations use a high-speed transmit/receive chipset from Hewlett-Packard, called the HDMP-1000, but more colloquially the G-link to refer its gigabit/second capability. Laser-equipped, it is fully capable of over 1 gigabit/second throughput, but we use it without lasers over 50 ft. of twisted pair, where it is successfully operating at 155 Mbit/second, the speed of an OC-3c SONET link. The G-link chip provides the model of a virtual ribbon cable using a high-speed serial link. This virtual ribbon cable serves to connect two ATM host interfaces, which send and receive ATM cells for their respective workstation hosts.

The broker, and other communication software and hardware support for video, audio and ATM host interface are implemented on IBM RISC System/6000 workstations using the AIX operating system. Robot sensory data are obtained from a SUN-4 with an SBus-to-MCA bus interconnection card at the slave side. On the master side a real-time processor (called "JIFFE") and a dedicated IBM PC provide robot control. Another bus connector card connects the JIFFE processor with the IBM RS/6000 workstation.

The objective of the broker research and resource orchestration at the end-points at this stage is to study (1) the provision of strict guarantees, and (2) dynamics of the telerobotics application requirements (removal/addition/change in quality of media/connections). This requires an establishment of customized connections, using the broker and its services, and the dynamic management of resources invoked due to QoS requirements from the user.

### 4.3 Implementation Choices for our Experimental Setup

Running a specific application above the QoS Broker, several selections have been made with respect to the QoS Broker. First, the QoS Broker for the teleoperation application uses the sender-initiated brokerage mode for sending data from master to slave and receiver-initiated brokerage mode for receiving data from slave to master because of the application behavior. Second, the additional

information in the QoS Broker is specified to serve the telerobotics application, i.e., in addition to the exchange of application QoS parameters, the operator sends a request to the robot to get a video image because the operator wants to view the working environment of the robot before real-time transmission starts. The robot sends the requested image in the response message of the negotiation. Our detailed application QoS negotiation in a robotics environment is described in [2]. Third, currently no group brokerage is supported because we do not have the group setup for the telerobotics application (although controlling a group of robots would be interesting).

#### 4.3.1 Implementation Restrictions induced by the AIX Operating System

Our preliminary measurements [1] and analysis with the AIX OS, using video and robotics data, showed that using the so-called '*real-time*' priorities are not sufficient to control protocol task behavior when used for implementation of rate-monotone or deadline-based scheduling, unless severe restrictions are made. These include (1) having only one user, (2) one multimedia application running on the RS/6000, (3) implementation of application/transport protocols in a single user process with real-time priority, and (4) rate-monotone and deadline based scheduling is done by the protocol stack. Only with these restrictions satisfied can rate-monotonic scheduling be mapped into the real-time priority scheme of AIX to provide (approximate) predictability for guaranteed services.

While our research objectives, outlined in section 4.2, are not jeopardized by the restrictions, for more general purpose uses, the broker and guarantees will require far better support from the OS.

#### 4.3.2 Implementation Restrictions induced by our ATM LAN Network

The broker does not yet rely on network resource management in the ATM layer, as this mechanism is not implemented in the host interface or in the ATM switch. For the lightly loaded ATM LAN in our experimental environment, the network resources are always available and successfully allocated. Therefore the response from the ATM LAN to the broker is assumed to be 'accept'. This trivialized the network management, but let us test the broker's distributed end-to-end entities (buyer/seller) to (1) orchestrate the local buyer resources, (2) orchestrate the remote seller resources, and (3) coordinate between them. The admission service in the transport subsystem does partial control of network resources, for example, available bandwidth in terms of transport packets not ATM cells,

| | **Robotics Data** | **Video Data** |
|---|---|---|
| **End-to-End Delay** | 2 ms | 80 ms |
| **Protocol Overhead RTAP/RTNP together** | Send: 0.4 ms Receive: 1.1 ms | Send: 6.9 ms Receive: 68.9 ms |
| **Error Rate** | 0.06 % | Quality Degraded over Time with no Recovery Possibility |
| **Transmission Interval** | 20 ms | 1 second |

Table 2: *Preliminary Results of Robotics and Video Data Transmission using QoS Broker*

end-to-end delay, buffer space for queues to schedule the packets over ATM host interface VCIs.

The other limitation is the absence of a practical way to experiment with the broker using other LANs. If using other LANs such as a Token Ring, the end-to-end links are shared links. In this case, the MAC layer needs to be included in resource reservation and allocation, which is difficult.

#### 4.3.3 Preliminary Results

We ran two distinct tests to establish customized connections for robotics data and video data using the QoS Broker, and to transmit them according to a 'deal' negotiated by the broker using our Real-Time Application Protocol (RTAP) at the application subsystem and Real-Time Network Protocol (RTNP) at the transport subsystem. RTAP and RTNP include parameterized protocol tasks for data movement, synchronization, forward error correction, and control.

In the first test, the broker got the robotics application QoS specification (sample size 64 bytes, sample rate 50 samples/second, loss rate 1 sample/minute, end-to-end delay bound 20 ms), performed admission, translation and negotiation, and stored the possible resources (parameters) in the corresponding profiles (application QoS profile/network QoS profile). Then RTAP/RTNP started to send/receive robotics data over the specified connection according to the parameters they retrieved from the corresponding profiles.

In the second test, the same activity as in the first test was performed for video data (sample size 48000 bytes = 240 $\times$ 200 pixels/frame, sample rate 1 frame/second, loss rate 1 frame/minute, end-to-end delay upper bound 200 ms). The results are shown in Table 2 and they represent the average values.

# 5 Conclusion

The QoS Broker is a new end-point design for resource orchestration, drawing on successful models in human affairs. The design provides a specialized manager to establish resource guarantees, using detailed databases and negotiation among managers of required resources. We have by no means completely worked out the details of bidirectional QoS translation, admission tests and negotiation among system elements, but expect to incorporate effective solutions as they are discovered. The major contribution we have made is to organize these translations, admission tests and negotiations into a system architecture which we believe provides significant advantages over existing architectures for managing QoS. We have validated the architecture by building an experimental prototype, which has served both to help us select appropriate techniques as well as identify limitations in existing systems.

Treating the system components (e.g., the end-stations and the network infrastructure) as peers has implications which may prove useful in many future systems. While we used telerobotics as a driving application both in our work and in the discussion in this paper, the QoS Brokerage idea seems to have wide application. One example is computer modems, where the line quality may be a dynamic, requiring signaling of bandwidth capacity to an application as the line improves or degrades. A second example is using one or more idle workstations to support an application; when the workstation owner begins typing at the system console, the allocation of system resources may have to change dramatically. Finally, consider ATM over wireless media. It is advantageous to use the same software model for both wired and wireless media above the ATM layer. But in this case, the link quality may change as the mobile application element moves from an indoor wireless LAN to wireless microcells, and on to ATM over cellular. In each of these three examples, the resource management concept of the broker allows the application and the system components to cooperate in support of the user.

The QoS Broker concept is general enough to be useful across many implementation technologies. It can incorporate, for example, the integrated layering approach in the control-management plane, proposed by Clark, et al. [10]. Further, the broker has interaction mechanisms in it to make 'contracts' with an OS as well as with network resource management. When operating systems and network subsystems for which contract protocols exist are available, the broker uses them. We expect this availability to become more common as multimedia manipulation and other advanced

uses of machines becomes more common and the best-effort resource-sharing models of common operating systems are stressed to failure.

We expect to continue building on our telerobotics experience. This continues to validate our prototype as it is extended to meet more complex applications requirements, e.g., for distributed two-arm manipulation of an object. A result of dealing with greater complexity in the applications is that we may uncover more general translation schemes than we have discussed in this paper.

## 6 Acknowledgment

We gratefully acknowledge the rich discussions we have had with colleagues in formulating these ideas, particularly David Farber and Ralf Steinmetz. Brendan Traw has been a key contributor of ATM technology, and Ruzena Bajcsy of Penn's General Robotics and Sensory Perception (GRASP) laboratory inspired the investigation of telerobotics. We would also like to acknowledge many valuable ideas and suggestions from the anonymous reviewers.

## References

[1] Klara Nahrstedt, Jonathan Smith, "Experimental Study of End-to-End QoS", submitted to ACM/Springer Multimedia Systems, also available as MS-CIS-94-08, Technical Report, University of Pennsylvania, February 1994

[2] Klara Nahrstedt, Jonathan Smith, "QoS Negotiation in a Robotics Environment", *Workshop on Distributed Multimedia Applications and Quality of Service Verification*, Montreal, Canada, June 1994, pp. 153-161

[3] Klara Nahrstedt, Jonathan Smith, "A Service Kernel for Multimedia Endstations", in *Multimedia: Advanced Teleservices and High-Speed Communication Architectures*, Lecture Notes in Computer Science No. 868, Ralf Steinmetz (ed.), Springer Verlag, 1994, pp. 8-22

[4] Michael B. Jones, "Adaptive Real-Time Resource Management Supporting Modular Composition of Digital Multimedia Services", *4th International Workshop on Network and Operating System Support for Digital Audio and Video*, Lancaster, England, November 3-5, 1993, published in Lecture Notes in Computer Science Vol. 846, Springer-Verlag, 1994, pp. 11-18

[5] Lixia Zhang, Bob Braden, Deborah Estrin, Shai Herzog, Sugih Jamin, "RSVP: A new Resource ReSerVation Protocol", *IEEE Network*, September 1993, pp. 8-18

[6] Bruce Mah, "A Mechanism for the Administration of Real-Time Channels", *Master Thesis*, University of California at Berkeley and International Computer Science Institute, March, 1993

[7] J. Jung, D. Seret, "Translation of QoS Parameters into ATM Performance Parameters in B-ISDN", *IEEE INFOCOM'93*, Volume II, San Francisco, CA, March 1993, pp. 748-755

[8] Andrew Campbell, Geoffrey Coulson, Francisco Garcia, David Hutchison, "A Continuous Media Transport and orchestration Service", *SIGCOMM'92*, ACM, Baltimore, MD, August 1992, pp. 99-110

[9] Frank Oliver Hoffmann, Luca Delgrossi, "Detailed Tour of ST-II for the Heidelberg Transport System", *Technical Report 43.9302*, IBM European Networking Center, Heidelberg, Germany, 1993

[10] David D. Clark, Scott Shenker, Lixia Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", *ACM SIGCOMM'92* , Baltimore, MD, August 1992, pp. 14-26

[11] Lawrence A. Crutcher and A. Gill Waters, "Connection Management for an ATM Network", *IEEE Network*, November 1992, pp. 42-55

[12] E. Schooler, S. Casner, "An Architecture for Multimedia Connection management", *Multimedia'92*, Monterey, CA, April 1992, pp. 271-274

[13] Domenico Ferrari, D. C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, April 1990, pp. 368-379

[14] David P. Anderson, Ron Kuivila "A System for Computer Music Performance", *ACM Transactions on Computer Systems*, Vol. 8, No. 1, February 1990, pp. 56-82