January 1994

The QoS Broker

Klara Nahrstedt
University of Pennsylvania

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
http://repository.upenn.edu/cis_reports/245


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/245
For more information, please contact repository@pobox.upenn.edu.
The QoS Broker

Abstract
Many networked multimedia applications are delay-sensitive, and hence desire services with guarantees of resource availability and timeliness. For networks such as those based on Asynchronous Transfer Mode (ATM), these services are specified through Quality of Service (QoS) parameters. Delivering end-to-end QoS implies complex resource management at the end-points (e.g., computer workstation hosts), as well as in the underlying network. In this paper, we describe a model for an end-point entity, which we have designed and implemented, called the QoS Broker. The broker orchestrates resources at the end-points, cooperating with resource management in the underlying ATM network. The broker, as an intermediary, hides implementation details from applications and resource managers. We motivate the concept and particulars of our design, including services such as translation, admission and negotiation which the broker uses to properly configure the system to application needs. We treat the QoS negotiation as a 'deal' between the user ("buyer") and the network ("seller") for the setup of a customized connection. The key concept is that the broker is an active intermediary which isolates cooperating entities from operational details of other entities.

Comments
The QoS Broker

MS-CIS-94-13
DISTRIBUTED SYSTEMS LAB 77

Klara Nahrstedt
Jonathan Smith

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

March 1994
The QoS Broker

Klara Nahrstedt
Jonathan M. Smith*

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

Abstract

Many networked multimedia applications are delay-sensitive, and hence desire services with guarantees of resource availability and timeliness. For networks such as those based on Asynchronous Transfer Mode (ATM), these services are specified through Quality of Service (QoS) parameters. Delivering end-to-end QoS implies complex resource management at the end-points (e.g., computer workstation hosts), as well as in the underlying network.

In this paper, we describe a model for an end-point entity, which we have designed and implemented, called the QoS Broker. The broker orchestrates resources at the end-points, cooperating with resource management in the underlying ATM network. The broker, as an intermediary, hides implementation details from applications and resource managers. We motivate the concept and particulars of our design, including services such as translation, admission and negotiation which the broker uses to properly configure the system to application needs. We treat the QoS negotiation as a ‘deal’ between the user (“buyer”) and the network (“seller”) for the setup of a customized connection.

The key concept is that the broker is an active intermediary which isolates cooperating entities from operational details of other entities.

1 Introduction

The design and implementation of multimedia communications systems requires an architecture in which system components, such as protocols, devices and schedules, are configured and coordinated as a system. In particular, since many such systems are delay-sensitive, services that they employ must provide guarantees. This is well-understood in the context of real-time communications between hosts, but less well-understood in the context of applications, which require resources beyond communication services. These other resources, such as processing capacity, must be managed in concert with networking requirements in order to deliver guaranteed behavior to applications.

In the design and implementation of a system for teleoperation operating over an ATM network (described in Section 3.1), we were faced with the problem of supporting applications which have

*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.
strict requirements yet wish to isolate themselves from details of process scheduling or bandwidth allocation. It became clear that a new model was needed for specifying application requirements, and translating those requirements into negotiated resource allocations. Further, this new model would result in a new and different way to structure multimedia systems.

Before describing the QoS broker, it is worth examining some other structuring models, both to borrow good ideas from and to analyze where they fall short of our needs (Figure 1a, Figure 1b).

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 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. 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 the features of advanced displays. Unfortunately, this led to considerable redundancy, non-standard terminal handling code, and problems with portability. The termcap library and database address 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 [15].

We choose to follow this model of servicing applications using extensive information for customization to specific operating environments. The following 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 resources, unlike a terminal which is PASSIVE. This means, in practice, that the managed resources may signal changes of state, implying that communication must be bidirectional. It also means that the information used for decision-making cannot be entirely contained in a static database.

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. It also means that delay-sensitive applications (as most multimedia applications are) must be protected from much of the dynamics caused by sharing.

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

While satisfying these goals completely is unlikely, we have defined and implemented a prototype architecture which offers considerable progress towards these goals. 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 between these hosts (e.g., at the network switches or intermediate network nodes).

The QoS Broker as shown in Figure 1c is an entity which is responsible for the orchestration of resources at the hosts and cooperates with a network resource orchestration. The QoS Broker is a decision making protocol engine which employs resource databases and a variety of different services to achieve a balance (‘deal’) among application requirements and multimedia I/O devices, as well as network and operating system resources.
We describe the QoS Broker concept, design and services in the next section, Section 2. Section 3 gives an overview of the experimental setup, our implementation, and limitations of the prototype broker caused by the current implementation platform. Section 4 concludes with a discussion of system features needed to implement a fully functional QoS Broker.

1.2 Related Work

Most related research has been on resource management in networks, where resources such as bandwidth, and buffer space for queues are allocated and controlled. 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 [5], [11], [13]. Protocols for reservation (RSVP) and administration (RCAP) of network resources are described in [4], [6]. Network resource management in network protocols is done, for example, in ST-II [9]. An important component of resource management systems is translation of resource specification between consecutive layers of the network protocol hierarchy. An example is translation between AAL resource parameterization and ATM resource parameterization, as presented in [7].

Resource management at the end-point, e.g., of CPU and disk space used to support local multimedia services [3], is a necessary part of end-to-end design. Several real-time extensions of different operating systems have been introduced, such as Mach (NeXT), AIX (IBM RS/6000), Solaris (SPARC 10), IRIX (SGI) to improve support for ‘delay sensitive’ multimedia applications.

Thus these systems have in fact partially orchestrated the necessary elements. For example, the Lancaster system [8] orchestrates the behavior of the network components, and the computer music system of Anderson [14] orchestrates the relationships of users and devices, and Schooler’s system[12] orchestrates some parts of end-to-end communication.

The problem is that there is a little orchestration between the 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 discovered experimentally[1], the behaviors of these components are to a large degree interdependent in a networked multimedia system. This suggests that the orchestration of these resources be integrated in a single entity.

2 QoS Broker Concept

The QoS Broker entity is an end-point management entity which orchestrates resources for multimedia networked applications. To provide applications with end-to-end guarantees, network resource
management alone is not sufficient. This is particularly true when end-points, where applications operate, 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.

### 2.1 End-Point Model

Our end-point model, shown in Figure 2, includes two major subsystems: an *application subsystem* and a *transport subsystem*. The application subsystem is assumed to be embedded in the OS *user space* protection domain, while for reasons of protection, resource scheduling, and access to network interface hardware, the transport subsystem is embedded in the OS *system space*. Figure 2 relates the application, network and OS. All three components work with resources in their own terms, and thus the functional and modular division involves some information-hiding as well.

The **application's resources** are represented through local or remotely located I/O devices ("media") comprising a multimedia system. The resources are described (or *parameterized*) through **application QoS parameters**. The parameters consist of *media quality* descriptions for the specific *media characteristics* of each device, as well as its *transmission characteristics* requirements for end-to-end delivery, and any *media relations* among media (e.g., synchronization).

The **network resources** are bandwidth, buffer space for packet/cell queuing, intermediate packet delay, end-to-end delay and jitter. They are parameterized through the **network QoS parameters**. The parameters are divided into *throughput pledges, traffic characteristics, and performance requirements*.

The **OS resources** are processing times required for tasks (both application and OS proxy tasks), secondary storage, and memory buffer requirements. These are parameterized through **system parameters**.

---

Figure 2: *End-Point Model with Layer and Resource Specification*
All parameters are stored in profiles (databases), to which the broker has access. The broker uses them to support its decision-making process (brokerage process). Parameters are specified in a form consistent with their use (details are described in [2]). For example, delay bound parameters are specified as a range of acceptable values \( j\text{expected value, upper bound value}i \). Jitter in system behavior can be accommodated by specifying task processing times as a pair \( j\text{average processing time, worst acceptable processing time}i \). Hence the task processing time will be accepted if it is in the interval bounded by the pair of values.

### 2.2 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 applications with QoS requirements and the “sellers” are resource managers which can potentially deliver the requirements.

In a distributed system, the broker entity has two main components: the broker-buyer and the broker-seller (Figure 3).

The buyer at the sending side wants to establish a customized connection for the user. The broker-buyer includes following activities (as shown in Figure 3): (1) orchestrates the local resources for outgoing connections at the sender side, (2) relies on the information about available network resources in the intermediate elements (such as switches) provided by network resource management, and (3) relies on the information from the broker-seller at the receiver side in order to set up customized connection.

The seller is located at the receiving side. It orchestrates the remote end-point resources for
incoming connections, 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 3.

The communication between the brokers draws on other specialized services (e.g., admission) and protocols (e.g., negotiation). Figure 4, which is necessarily a bit "busy", gives a more detailed illustration of communication (i.e., the process shown in Figure 3) between the broker entities and the underlying network in the context of our end-point model.

Returning to our end-point 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 signaling 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 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 [2].

2.3 QoS Broker Protocol

The broker protocol incorporates three types of communication: layer-to-layer, peer-to-peer, and layer-to-OS communication. Communication 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. The application-to-transport communication is provided by the QoS translator. Peer-to-peer communication is done by negotiation service between the peers. For instance, the negotiation of application QoS provides communication among the distributed end-point application entities with respect to their application QoS. Analogously, the negotiation of network QoS employs the 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.

The signaling among the protocol services includes answers: ‘accept’ when expected resources can be reserved on the way to the remote side and allocated on the way back to the initiator side, ‘reject’ when required resources cannot be provided, and ‘modify’ when resources have to be relaxed, but are still in bounds of acceptance.

**QoS Broker Protocol - Buyer**

The broker-buyer protocol (Figure 5) is initiated by the input of application QoS requirements. 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, to meet the requirements. The admission functionality currently implies that application protocol task processing times and buffer space requirements are known a priori and present in the system.
parameter profile before admissions decisions are made. Admission service at the application level does 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) end-to-end delay test to see if tasks to move data out of the user space can meet the specified end-to-end delay upper bound.

If local resources are reserved, then negotiation with the remote broker-seller occurs. Negotiation concerns the receiver’s ability to accommodate incoming multimedia quality. This ability would depend on the receiver having available appropriate output devices, and temporal and spatial resources.

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 initiation results in several steps, which the broker-buyer in the transport subsystem has to perform:

First, the application QoS requirements are translated into network QoS requirements using the QoS translator. An example of a translation\(^2\) at the application/transport interface (one medium onto one connection) is shown in Figure 6. As we alluded to 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 need to be transported with different network QoS. The application QoS profile includes the entire multimedia networked application description (input and

\(^1\)Schedulability test should be part of OS resource management. In current OS, this function is inadequate. Hence it is part of the admission service in the broker. The derivation of the schedulability tests for periodic multimedia streams in a workstation environment can be found in [2].

\(^2\)The 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).
output media), hence the translator has to pick input medium by medium and map it into outgoing transport connections with their specific network QoS parameters. The broker stores the network QoS parameters in the network QoS profile.

Second, after translation, the admission for the transport subsystem is invoked. The broker maps the network QoS parameters to spatial and temporal resources needed by the transport tasks to move data. This admission service tests not only the resources of the transport/ network protocol but also does an admission test for all end-point resources, including application resource requirements. They must be jointly managed because they share resources such as the processor.

Third, if the admission for the host is successful, negotiation of network QoS parameters per connection is initiated by the broker. This negotiation relies (1) on the translation between network QoS in terms of transport packets and underlying ATM cells at the boundary between AAL and ATM layers, and (2) on the network resource management in the ATM layers across the ATM network.

Forth, the broker waits for the answers from the network resource management and collects all answers about requested customized connections. After having all responses, they are translated back to the application QoS, so that the user gets a complex picture which media at what quality will be transmitted.

**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 of the protocol) 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 admission service in the transport subsystem. 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 in order to allocate/relax/free the application resources, depending on the outcome.

3 QoS Broker Implementation

We have implemented an experimental prototype of the broker in order to get an initial testbed for our ideas and to refine the notion after testing with applications.

3.1 Experimental Setup

Our test application is telerobotics. This telerobotics/teleoperation application is non-trivial and has challenges distinct from teleconferencing. Our test configuration is shown in Figure 7.

Figure 7: Telerobotics System Configuration

*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 samples/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 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 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 strict guarantees and (2) dynamics of the telerobotics application requirements (removal/addition/change in quality of media/connections). These issues imply an establishment of customized connections, using the broker and its services, and the dynamic management of resources invoked by the degradation of QoS requirements from the user.

3.2 Implementation of QoS Broker under AIX and Restrictions induced by OS

The first version of broker, as well as our experimental transport protocol stack excepting the ATM device driver, is implemented as a user process using procedure interfaces. Table 1 shows the application service procedures and the QoS translator procedure. The set of service procedures

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetAppQoS</td>
<td>Sets Application QoS Parameters in Application QoS Profile</td>
</tr>
<tr>
<td>GetAppQoS</td>
<td>Retrieves Application QoS Parameters from Application QoS Profile</td>
</tr>
<tr>
<td>SetAppTaskParam</td>
<td>Sets OS Parameters for Application Task in System Parameter Profile</td>
</tr>
<tr>
<td>GetAppTaskParam</td>
<td>Retrieves Task Parameters from System Parameter Profile</td>
</tr>
<tr>
<td>SetAppSysParam</td>
<td>Sets System Parameter Description of Application in System Parameter Profile</td>
</tr>
<tr>
<td>GetAppSysParam</td>
<td>Retrieves System Description of Application from System Parameter Profile</td>
</tr>
<tr>
<td>AdmitAppQoS</td>
<td>Admits Application QoS Parameters</td>
</tr>
<tr>
<td>NegotiateAppQoS</td>
<td>Negotiates Application QoS between Application Sender and Receiver</td>
</tr>
<tr>
<td>QoSTranslator</td>
<td>Communicates between Application and Transport Layers</td>
</tr>
</tbody>
</table>

Table 1: Service Procedures used by the QoS Broker Entity

for the broker’s transport part is equivalent to the set of the application service procedures. 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 API interface is \( \text{Broker}(\text{Application Requirements, entity}) \), where application requirements are passed in form of application QoS parameter profile identifier. The \( \text{entity} \) parameter specifies the side of the brokerage process - buyer or seller. The negotiation protocol of application QoS parameters is split from the negotiation of network QoS - a TCP/IP connection is used by the broker for negotiation of application QoS; for negotiation of network QoS an ATM signaling channel is used. This entire functionality requires only a few milliseconds.

The current broker requires a \textit{preprocessing phase}. This means that the system parameters of tasks processing times have to be filled into the system parameter profile during the installation phase of the protocol stack. This also implies that the application and transport protocol tasks must be known in advance. This is not an optimal implementation solution, but currently AIX does not support guaranteed services. 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.

Our preliminary measurements and analysis with AIX, using video and robotics data showed [1], that using the so-called “\textit{real-time}” priorities does not really help to control protocol task behavior when used for implementation of rate-monotone or deadline-based scheduling, unless severe restrictions are made, such as (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 3.1., are not jeopardized by the restrictions, for more general purpose uses, the broker and guarantees will require far better support from the OS.

3.3 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 Sunshine ATM switch. For the lightly loaded ATM LAN in our experimental environment, the network resources are always available and 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, end-to-end delay, buffer space for queues to schedule the packets over ATM host interface VCIs.

4 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.

It can incorporate, for example, the integrated layering approach in the control-management plane, proposed by Clark at el. [10]. It has interaction mechanisms in it to make ‘contracts’ with an OS as well as with network resource management. When ‘contract’ oriented operating systems and network subsystems are available, the broker uses them.

Our current broker implementation has some restrictions due to our experimental platform’s inability to make true scheduling guarantees. In particular, the support for guarantees is weak due to the use of priorities, as priorities do not preclude admission of processes which share the priority, and thus affect resource allocations. However, using our telerobotics experience, we expect to validate our prototype and expand it to meet the needs of a real application.

References


