Quality of Service (QoS) in Communication APIs

Werner Almesberger,[*] EPFL DI-LRC,
CH-1015 Lausanne, Switzerland, almesber@lrc.di.epfl.ch
Serge Sasyan, Hewlett-Packard France ENSD,
serge_sasyan@hp.com
Steven Wright, Fujitsu Network Communications,
s_wright@fujitsu-fnc.com

March 19, 1997

Keywords: QoS, API, ATM, RSVP, WinSock 2, XTI, Socket, DLPI, Arequipa

Abstract: This paper gives an overview of the mechanisms that currently specified APIs (for Native ATM and for RSVP) provide to applications for the purpose of controlling QoS parameters. It compares the abstractions they provide with respect to application needs and modern operating system practices. Furthermore, it discusses what extensions or modifications would be desirable in future APIs.

1 Introduction

After an introduction to general aspects of QoS and APIs, we describe in section 2 how several APIs for network architecture with QoS support provide control over QoS parameters, how they present signaling mechanisms to the application, and how they integrate connection establishment with reservation setup. In section 3, we examine if they meet application needs and how they integrate with advanced operating system concepts. Finally, section 4 proposes directions in which QoS-aware APIs should evolve in the future.

[*] Contact author.

1.1 What is QoS ?

Quality of Service (QoS) concerns are applicable at all layers of a protocol stack, and depending on the specific protocols, there may be an explicit transformation of the service guarantees as the user information transits the protocol stack. QoS concepts can be applied where there are performance aspects of interprocess communication, even when there is no external network. Most humans interacting with computers expect some sort of response within a reasonable time and this provides what one might call a user level QoS constraint on system performance. In general there may be multiple other dimensions to QoS e.g. reliability, security, loss, cost, etc.

User level QoS issues are becoming increasingly important as people rely more on computer interaction as a fundamental daily activity. One measure of this is in terms of the economic consequences of QoS failures. This is not just a question of customer relations for the service providers. Performance failures of consumer services may lead to legal or regulatory consequences. In the context of a Global Information Infrastructure, QoS must be parameterized and mechanisms for allocations and parameter degradation are required for deployment. For some applications, there are guidelines already available (see e.g. [1] for voice services), for others, the user level QoS requirements need to be clarified first.

QoS guarantees are most valuable when applied end-to-end. Unfortunately this is also the most complex case, since it requires common QoS concepts to be available across a variety of equipments, perhaps within different jurisdictions.

QoS concerns might arise wherever there are multiple independent requests for a constrained resource, (e.g. a multiplexer or scheduling function). This is not just a problem for network equipment – it applies also to the software applications that can introduce degradations into the end-to-end service. There have been some efforts in multi-threaded and distributed operating systems to provide support for QoS notions (see e.g. [2]) and also in terms of QoS support for higher layer constructs in the information infrastructure (e.g. Object Resource Brokers [3]). The widest consensus on QoS notions seems to have been reached on the communications aspects, here we have selected three APIs that are specifically relevant in that context, and so we are primarily concerned with QoS aspects such as loss and delay.

1.2 APIs in QoS Architecture

Because networks are becoming available which provide native support for QoS concepts, it is appropriate to consider related issues such as the mechanisms by which applications can best utilize these network based QoS concepts in support of user level QoS requirements. For the majority of software based network applications, the application developers make use of such network based QoS guarantees through some form of Application Programming Interface (API).

APIs with QoS concepts are required at various interfaces within a larger system or application wide QoS Architecture. Various QoS architectures and other abstractions have been proposed in the literature (see e.g. [4, 5, 6]). [7] proposes that QoS architecture concepts are required to span OSI layers 1-7 and also addresses aspects such as:

- orchestration of multiple information flows with related QoS bounds (e.g. relative jitter between component media streams of an integrated service),
- QoS negotiation, renegotiation and degradation indications,
- reservation of end system and network resources,
- policing of these resources,
- connection admission policies,
- resource control, monitoring and regulation for support of distributed integrated service (e.g. multimedia) systems

[8] arranges the elements in a pipeline in order to allocate delay budgets to the various components in the overall system. One of the interesting aspects of this specific implementation was that, although deterministic guarantees derived from queuing models (e.g. as provided by ATM switches) were desired, the host's I/O subsystem was not able to provide deterministic delay guarantees, and so only statistical guarantees could be provided end-to-end. QoS guarantees of a statistical nature (rather than deterministic) may be important until the software environment can also support deterministic guarantees.

2 API case studies

In this paper we consider several APIs which provide QoS guarantees for network services. Our intent here is not to provide an exhaustive list of APIs which provide some form of QoS support. In selecting APIs we looked for APIs which were clearly specified, and had some implementations available to provide the basis for further comparisons. Our preference has been to consider APIs which support QoS concepts which are well defined, e.g., ATM or the IETF's Integrated Services model. The APIs we compare in the later sections of the paper are:

- RSVP
- Native ATM
- Arequipa

For some cases, code examples of were given to illustrate the process of setting up a reservation. Those examples are heavily trimmed, do no error checking, and some parts of the code are only hinted at by comments.

2.1 RSVP

The Internet Engineering Task Force (IETF) has defined an architecture for providing integrated services on the Internet [9]. The protocol used to establish and maintain reservations is called the "Resource reSerVation Protocol" (RSVP, [10]).

There is currently only one published API for RSVP: the API defined by ISI and Sun Microsystems (SMI) [11] used in the RSVP implementations by ISI and by SMI.[1]

Since the services provided by and used with RSVP and the API are both still work in progress, the respective current versions may differ slightly from what is described in this paper.

RSVP carries traffic descriptions (called TSpecs), information about the network (called Adspecs), and the actual reservation parameters in flow specs (see [12] for details). Flows are selected using filter specs. The ISI/SMI RSVP API provides access to exactly the data conveyed in RSVP messages, i.e. for each flow, each sender can provide a TSpec and an Adspec, and each receiver can provide filter specs and flow specs. The numerical values are identical to the values contained in the RSVP messages. Additionally, placeholders for extensions such as explicit packet format templates and policy information are defined.

While there is no abstraction for the reservation information, the API hides protocol activities like continuous refreshes from the application. It does however generate asynchronous notifications in the event of changes in the reservation state, e.g. if a reservation succeeds, fails, or changes.

The API design follows the design of RSVP in that the mechanisms for reservation setup and for the actual flow setup are independent. Note that this can be used to set up reservations on behalf of other programs, which is nicely illustrated by the `tkrsvp` application (ISI).

Example:

```
#define TTL 10

struct sockaddr_in dest;
struct sockaddr_in host;
rapi_tspec_t tspec;
int rapi_errno;

/* set dest (gethostbyname, etc.) */
sid = rapi_session(&dest,0,0,NULL,NULL,
  &rapi_errno);
host.sin_family = AF_INET;
host.sin_addr.s_addr = htonl(INADDR_ANY);
host.sin_addr.s_port = 0;
tspec.form = RAPI_FLOWSTYPE_CSZ;
tspec.len = sizeof(qos_tspecx_t);
tspec.tspecbody_qosx.spec_type =
  QOS_GUARANTEED;
tspec.tspecbody_qosx.xtspec_r = 5300;
tspec.tspecbody_qosx.xtspec_b = 900;
tspec.tspecbody_qosx.xtspec_p = 14617;
tspec.tspecbody_qosx.xtspec_m = 60;
tspec.tspecbody_qosx.xtspec_M = 572;
rapi_sender(sid,0,&host,NULL,&tspec,NULL,
  NULL,TTL);
```

[1] `ftp://ftp.isi.edu/rsvp/release` and `ftp://playground.sun.com/pub/rsvp`

2.2 Native ATM

APIs for "native ATM" allow applications to establish ATM connections and to control the QoS parameters that are chosen for the connections. Native ATM APIs normally don't distinguish between the actual reservation and connection establishment, because a connection and the corresponding reservation are set up at the same time in the ATM network too. Furthermore, as of UNI 3.1 and UNI 4.0 ([13, 14]), connection parameters can't be modified once the connection is established. A connection's QoS therefore remains fixed for its entire lifetime.

2.2.1 Semantic description

The document [15] specifies the semantic definition of ATM-specific services that are available to software programs and hardware residing in devices on the user side of the ATM User-Network Interface. The ATM environment provides new services to the application developer. They allow enhancements in performance and specification of network characteristics. Such ATM-specific services are denoted by the term "Native ATM Services".

"Semantic" means that the document describes the services in a way that is independent of any programming language or operating system environment. Semantic specifications for generic services define various aspects of the interface to those underlying services. Here we will focus on the call establishment. This is the step where the Quality of Service necessary for the user application are defined. The current specification is based on version 3.1 of the UNI ([13]).

Call Establishment Via the `ATM_associate_endpoint` primitive, the application first acquires a local connection endpoint with which primitives for using native ATM services may be exchanged. The `endpoint_identifier` returned from this primitive is used in the future to identify the newly acquired connection endpoint, for outbound as well as inbound call establishment.

Below, the negotiation of connection attributes is described separately for outgoing and for incoming connections.

Outbound The application signals its intent to initiate an outgoing call, via the `ATM_prepare_outgoing_call` primitive. In response, the connection endpoint creates data structures that initially hold default values for various connection attributes. Examples of connection attributes include the AAL type, forward peak cell rate, and QoS class. The complete list of connection attributes is defined in Annex A of [15]. The application may examine any of these default connection attributes via the `ATM_query_connection_attributes` primitive. In addition, the application may optionally modify some of these attributes via the `ATM_set_connection_attributes` primitive.

Depending on the implementation, each of these attributes may or may not be settable by the application. Also, the value to which the application attempts to set a given connection attribute may be modified by the connection endpoint, due to local resource constraints or the local implementation of Native ATM Services.

When the application is satisfied with the connection attributes, as represented by the connection endpoint, then a call is placed across the ATM network. The application initiates this via the `ATM_connect_outgoing_call` primitive. Included as a parameter of this primitive is `destination_SAP`.

Native ATM Services places the call across the ATM network to the target ATM device. If a point-to-point call attempt is successful, then Native ATM Services signals the application via an `ATM_P2P_call_active` primitive.

Inbound The application signals its intent to respond to an incoming call, via the `ATM_prepare_incoming_call` primitive.

The application then issues the `ATM_wait_on_incoming_call` primitive to request that incoming calls be queued and presented to the application. When such an incoming call does arrive, the connection endpoint notifies the application via the `ATM_arrival_of_incoming_call` primitive.

The application must make a decision as to whether or not to accept the call. The application may examine the connection attributes of the newly arrived incoming call via the `ATM_query_connection_attributes` primitive. In addition, the application may modify some attributes (as specified in annex F of UNI 3.1) via the `ATM_set_connection_attributes` primitive.

If the application decides to accept the call, the application invokes the `ATM_accept_incoming_call` primitive. Otherwise, the application rejects the call via the `ATM_reject_incoming_call` primitive.

Even if call has been accepted, the application must wait for the ATM network to award the call. After an incoming point-to-point call has been awarded, the application is notified via the `ATM_P2P_call_active` primitive.

2.2.2 WinSock 2

The ATM Forum reviewed the WinSock 2 ATM annex [16, 17] and a mapping from this syntax to the Semantic description [18]. This API currently supports only AAL5 and user defined AAL types, although it had been announced that there would be future upgrades to support other AAL types (e.g. AAL1) as well as extensions to support UNI 4.0 signaling as the semantic specification is updated. The WinSock

2 API does not support management plane primitives, but does provide other mechanisms to access this information.

The QoS structure used in WinSock 2 is contained in the `ProviderSpecific.buf` field of the QoS Structure. Note that use of this ATM-specific structure is optional by WinSock 2 clients, but if provided, it takes precedence over any more generic FLOWSPEC structure. The protocol specific QoS Structure for ATM is a concatenation of the Q.2931 Information Elements.

The basic QoS mechanism in WinSock 2 descends from the flow specification ([19]). Flow specs describe a set of characteristics about a proposed unidirectional flow through the network. Flowspecs provide parameters to indicate what level of service is required and provide a feedback mechanism for applications to use in adapting to network conditions. An application may establish its QoS requirements at any time up to and including at the time a connection is established. If the connect operation fails because of limited resources an error indication is given, and the application may scale down its service request and try again or simply give up.

After every connection attempt, transport providers update the associated flow spec structures in order to indicate the existing network conditions. Applications expect to be able to use this information about current network conditions to guide their use of the network, including any subsequent QoS requests.

WinSock 2 also considers that, even after a flow is established, conditions in the network may change or one of the communicating parties may invoke a QoS renegotiation which results in a reduction (or increase) in the available service level. A notification mechanism is included which utilizes the usual WinSock notification techniques to indicate to the application that QoS levels have changed. If the updated level of service is not acceptable, the application may adjust itself to accommodate it, attempt to renegotiate QoS, or close the socket. Note that support for the functionality described in this paragraph is implementation-specific and is not provided by the current ATM extension.

The flow specs proposed for WinSock 2 divide QoS characteristics into the following general areas:

- Source traffic description;
- Latency (delay and delay variation bounds);
- Level of service guarantee (best effort, controlled load, predictive service, or (deterministic) guaranteed service);
- Cost (this is a placeholder);
- Provider-specific parameters, e.g. ATM QoS definitions

QoS templates can be established for well known media flows e.g. H.323, G.711.

It is up to each service provider to determine the appropriate values for each element in the QoS structure, as well as any protocol or media-dependent QoS extensions. A default flow spec (best effort) is associated with each eligible socket at the time it is created.

2.2.3 X/Open

Based on the semantic description [15], ATM extensions have been specified for the XTI, XSocket, and DLPI APIs defined by X/Open.

XTI The syntax instantiation of the semantic description [15] to XTI is defined in [20].

ATM connection attributes are represented as options.

These options may be negotiated for a connection initiated by the transport user. This negotiation may involve the `t_optmgmt()` function as well as the use of options with `t_accept()`, `t_connect()`, `t_listen()`, and `t_rcvconnect()`.

XSocket The syntax instantiation of the semantic description [15] to XSockets is defined in [21]. The XSocket extensions for native ATM are derived from the corresponding XTI extensions.

ATM connection attributes are represented as socket options.

These options may be negotiated for a connection initiated by the transport user. This negotiation utilizes the `getsockopt()` and `setsockopt()` functions.

DLPI The definition of an instantiation to Data Link Provider Interface of the Native ATM Services is a work item of the SAA/API work group of the ATM Forum, per X/Open request. The connection management aspects for this specific API syntax are considered below, although the details may be subject to change as the specification is refined by the ATM Forum and X/Open.

Most of the ATM connection attributes can be queried using `DL_INFO_REQ` and `DL_INFO_ACK`. Other connection attributes are either provided in DLPI primitives exchanged at connection establishment time by the DLS user, or they are returned at bind or connection time by the DLS provider.

Example:

```
ATL_TO_connect_req V_connect_req;
ATL_T_primitives   V_primitive;

memset(&V_connect_req, 0,
  sizeof(V_connect_req));
V_connect_req.dl_connect_req.dl_primitive =
  DL_CONNECT_REQ;
V_connect_req.dl_connect_req.dl_growth = 0;
/* Fill adress part (dl_dest_addr_*, dlsap) */
V_connect_req.dl_connect_req.dl_qos_length =
  sizeof(ADL_T_dl_qos);
V_connect_req.dl_connect_req.dl_qos_offset =
  offsetof(ATL_TO_connect_req, qos);
ADL_F_init_dl_qos(&V_connect_req.qos);
/* ... */
V_connect_req.qos.ATM_Traffic_Descriptor.
  tag = ADL_C_valid;
V_connect_req.qos.ATM_Traffic_Descriptor.
  fwd_pcr_clp1.tag = ADL_C_valid;
V_connect_req.qos.ATM_Traffic_Descriptor.
  fwd_pcr_clp1.value = I_peak_rate;
V_connect_req.qos.ATM_Traffic_Descriptor.
  bak_pcr_clp1.tag = ADL_C_valid;
V_connect_req.qos.ATM_Traffic_Descriptor.
  bak_pcr_clp1.value = I_peak_rate;
V_connect_req.qos.ATM_Traffic_Descriptor.
  best_effort = ADL_C_invalid;
/* ... */
ATL_FO_putmsg(I_dl_conn, &V_connect_req,
  sizeof(V_connect_req));
ATL_FO_getmsg(I_dl_conn, &V_primitive,
  sizeof(V_primitive));
/* V_primitive.primitive indicates success
   or failure */
```

2.3 Arequipa

Arequipa [22, 23] is a mechanism which aims to allow TCP/IP applications the use of QoS as supported by ATM networks. Arequipa has been developed at EPFL since 1995 and it has been implemented on Linux.

Arequipa allows applications to establish direct point-to-point end-to-end ATM connections with given QoS at the link level. These connections are used exclusively by the applications that requested them. After setup of the Arequipa connection (i.e. the ATM SVC that is used for Arequipa), the applications can use the standard TCP/IP protocol suite (and its APIs) to exchange data.

Arequipa requires that communicating hosts can reach each other over both the Internet and an ATM network. Its scope can be extended to include hosts without direct ATM connectivity by the use of application-level gateways.

For practical reasons, Arequipa is based on the native ATM API described in [24]. Arequipa uses a pre-defined SAP (Service Access Point) and – as far as they are invoked from user mode – it performs all operations in blocking mode. It also handles acceptance of incoming connections transparently for the application. Furthermore, future versions of Arequipa may not require a user-provided ATM address and query an address resolution service (e.g. NHRP [25] or MPOA [26]) instead.

Arequipa can therefore be characterized as providing full access to the QoS capabilities of the underlying native ATM stack, but as restricting the choice of endpoints. Figure 1 illustrates how applications use the Arequipa library to set up ATM connections and to use them for their TCP/IP traffic.

Example:

```
struct sockaddr_in addr_in;
struct sockaddr_atmsvc addr_svc;
struct atm_qos qos;
int s;

s = socket(PF_INET,SOCK_STREAM,0);
/* set addr_in (gethostbyname, etc.) */
addr_svc.sas_family = AF_ATMSVC;
text2atm(atm_host_name,(struct sockaddr *)
```

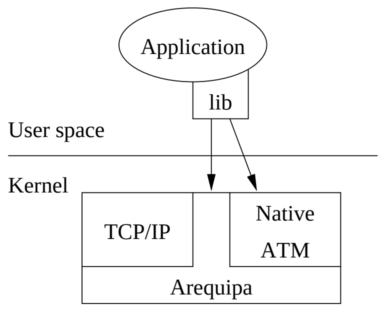

Figure 1: Interaction of applications with Arequipa

```
  &addr_svc,sizeof(addr_svc),T2A_SVC |
  T2A_NAME);
text2qos("cbr,aal5:pcr=100kbps",&qos,0);
arequipa_preset(s,&addr_svc,&qos);
connect(s,(struct sockaddr *) &addr_in,
  sizeof(addr_in));
/* send or receive data */
```

3 Evaluation

In the following sections, we examine how the QoS and other abstractions provided by the APIs meet application needs and how (if) they integrate with advanced operating system concepts.

3.1 Level of abstraction

All of the APIs we reviewed stay very close to the service interface the network provides: while they hide basic protocol mechanisms (e.g. retransmissions) from the applications, they (with the exception of Arequipa) are very careful not to miss a single opportunity to notify the application about any change in the overall reservation state. Furthermore, all of the APIs provide exactly the semantics and the syntax of the traffic characterization and reservation mechanisms of the respective architecture.

Access to most aspects of the protocol state has the disadvantage that existing interfaces need to be stretched very far, either by the introduction of many new primitives (e.g. RSVP, WinSock 2) or by overloading some functions (e.g. X/Open). Given that most applications only use a very small fraction of the available functionality, access to some simplified API is desirable. For RSVP, the tkrsvp package[2] offers a simplified API for Tk/Tcl programs.

The lack of abstraction in the QoS parameters raises several problems:

- API users frequently have difficulties characterizing the traffic their applications generate in the terminology and in the level of detail used by the network architecture.
- translation of QoS needs of the application to the interface provided by the API highly depends on the reservation architecture, i.e. ATM-based APIs expect all counts in cells, while RSVP APIs count in bytes and are also concerned with packet sizes and slack terms (see also [27] and [28]).

3.2 QoS dynamics

The life time of QoS settings can be aligned with different architectural concepts. The following alignments are common:

static, e.g. the configuration of an ATM PVC typically stays the same for hours or longer.

connection, e.g. with UNI 4.0, an ATM SVC keeps the same QoS parameters until the connection is close.

modification by signaling, e.g. modification of connection attributes after setup is specified in Q.2963.1 [29], but neither UNI 3.1 nor UNI 4.0 include that specification.

modification by traffic control, e.g. the ABR service specified in [30] gives the sender fine-grained control over the bandwidth it requests (within the limits negotiated at connection setup time)

An application which needs to change the quality of service over time, corresponding to different

[2]`ftp:/ftp.isi.edu//rsvp/release/app-tkrsvp.rel4.1a1.tar.Z`

phases of its operation, would benefit from one of the modification mechanisms. RSVP allows the modification of reservations for a flow at any time, so it provides a means to modify by signaling. (RSVP also supports reservations with different QoS parameters among leafs in a multicast tree.)

With ATM, neither Q.2963.1 nor ABR are generally available, so an application has to resort to either 1) allocate a quality of service which provides the maximum level of quality it needs at any time, or, 2) manage its session as a succession of connections, each with a different level of quality.

The first choice guarantees that the desired quality of service will always be obtained, but more resources than actually used are allocated (and therefore wasted), which is likely to result in higher cost for the user.

The second choice provides a tight resource management. It allows the resource provider to balance the unallocated resources and allows the application user to reduce its expenses, assuming higher level of quality of service is provided at a higher price, but at the cost of a more complex connection management, where connections are set up and tear down during session lifetime.

The possibility to dynamically re-negotiate QoS over time during the connection lifetime would avoid both, wasteful resource use and overly complex connection handling.

3.3 Advanced operating system concepts

QoS support in operating systems is considered primarily from the perspectives of real time support and concurrency. However, recent work in object based services may also be relevant as these become supported by the operating system. It has been proposed [3] that the following factors be considered in development of QoS support for real-time object based services:

1. policies and mechanisms for specifying end-to-end applications QoS requirements
2. optimized real-time operating system and network
3. optimized real-time communication protocols
4. optimized real-time request demultiplexing and dispatching
5. optimized memory management, optimized presentation layer

3.3.1 Real-time support

In considering real time programming, there is often a confusion in interpretation of what is required – the need for a fast implementation of the data transfer versus the need for a specification (explicit or implicit) of the expected delay which is both precise and accurate. For example, it may be easier to accommodate a data transfer mechanism that can provide a guaranteed delay of 50ms $\pm$ 5ms, than one that provides a great deal of variation (e.g average delay of 20ms but a range of 10ms to 100 ms).

The software implementations supporting the APIs typically attempt to minimize the delay between the I/O device and the API. However, the APIs we considered do not provide means to extend the guarantees to this part of the end-to-end path. [31] and [38] discuss more advanced architectural approaches.

3.3.2 Concurrency

For QoS-aware network applications, multi-tasking or multi-threading (i.e. concurrent execution of programs) are affected by the API in the following areas:

- processing of the application data, and
- control operations

Data processing can be parallelized if the concurrent threads can either work on distinct parts of the data stream or if elements of the data stream have no sequential inter-dependencies, so any thread can handle a new element whenever it is ready. The latter case is trivial, so all APIs we considered support it.

Automatic demultiplexing of data without involving the application is more difficult: While all APIs allow an application to open several flows and to perform operations on them, only RSVP can perform reservations for more than a single flow with one request. None of the APIs allow marking of data inside a single flow such that the operating system at the receiver can distribute the data directly to the thread that is processing it.

Most flexibility for parallelizing control operations is reached if no extra serialization of events is enforced by the API. With the exception of Arequipa, all the APIs we considered inflict serialization upon the applications only in negligible ways (e.g. the ATM APIs serialize incoming connections on the end point describing the service access point, but allow further processing to continue in parallel).

3.4 Legacy applications with implicit QoS requirements

In some cases, the data flow that is supported by the service quality guarantee does not need to pass through the same API that was used to specify the QoS level. One example of this approach is where legacy applications that are not QoS aware, but have implicit QoS requirements, are supported by their existing data plane API, but an additional API is used for management plane access to administratively control the QoS service provided to that application, see figure 2 (e.g. [32]).

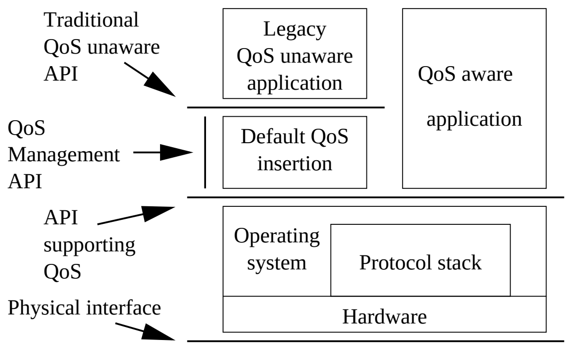

Figure 2: Management plane API supporting legacy applications with implicit QoS guarantees

4 Directions for future work

The diversity of API *syntaxes* can be explained by the different operating systems within which they are bound. The diversity of API *semantics* can be explained in part by the diversity of reservation protocols (e.g. soft state vs. hard state) and also by the level of abstraction of the interface in question. Three main directions for future work are seen:

- Unification of QoS semantics at the API for different architectures
- Extensions to the semantics of the APIs to cover data plane redirection
- Extensions of the QoS guarantees provided on external communications links to internal interprocess communications mechanisms
- Other software abstractions of QoS

4.1 Unification

Given that all the reservation services aim to support rather similar types of traffic (e.g. video encoded with MPEG-1, MPEG-2, or H.261), one could expect that a common set of parameters exists that the API (or an API layered on top of it) could translate to the set of parameters suitable for the reservation mechanism.

This is in sharp contrast to the current situation (see section 3.1), where APIs for distinct reservation architectures express QoS parameter in very different ways – even if they share the same operating system architecture.

The general QoS interface found in [16] and simplified RSVP API proposed in [33] are perhaps indicative for a trend towards the development of more abstract and unified APIs.

4.2 Data plane redirection

While most of the work on QoS Architectures to support multimedia services assume a general processor as an endpoint, it is also necessary to consider those applications where the data plane does not need to pass through the processor, but is destined for some hardware device – e.g. an MPEG codec. In many continuous media applications, the data stream from the network interface simply needs to be redirected towards the relevant I/O peripherals (e.g. the sound card or the display card). In this case, the data plane may have no appearance required at the API, and a management plane API may suffice. The ATM Forum SAA/API group is currently considering extensions of this type to the semantic API specification. The use of peripheral buses such as PCI [34] or PC-ATM [35] provide support for such approaches. Note that such buses can often be arranged in a hierarchical fashion which may complicate the data transfer. A simple switch abstraction may be the best mechanism to mask this potential complexity for the application while still supporting a QoS guarantee over this portion of the data transfer. Desk Area Networks [36] are one example of such an architectural approach.

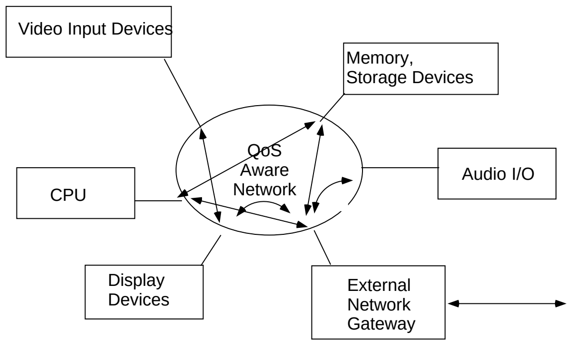

Figure 3: Data plane redirection

Conventional computing performance is measured in terms of processing power, however such performance measures are not necessarily appropriate for distributed applications such as those shown in figure 3, where not all the end points of a data flow are CPUs. Even I/O performance metrics (e.g. [37]) still assume a processor dependent component, which is not relevant to the network centric computing model of figure 3.

4.3 QoS for IPC

The APIs discussed in this paper provided mechanisms for QoS support over external interfaces. Mechanisms for the support of QoS over internal interfaces is an area of ongoing study topic for operating systems, particularly distributed operating systems and those intended to support real time multimedia applications (see e.g. [31, 38, 2]) This case is made difficult by the variety of different scheduling mechanism that need to be considered within the QoS architecture – not just the I/O devices, but also the buffer management, operating system processes etc. need to be considered.

4.4 Is an API always the answer?

APIs are not the only programming abstraction that can be used to provide support to applications that need QoS support. One could consider also the impact of QoS extensions in other programming abstractions such as the file system or elsewhere in the memory hierarchy. Most file systems rely primarily on semantics such as open, close, read, write which do not have any temporal basis. Extensions to file systems to support other semantics such as start/play, stop, pause, fast forward, rewind may also be required [39] to deal with continuous media types. The semantics of these operations may have temporal aspects that would need to be related to QoS support in other parts of the operating system.

One could also consider language constructs to support the use of QoS. QoS concepts are concerned with guarantees of performance. As such it might be reasonable to consider such guarantees in the light of language mechanisms to specify operational performance. One might reasonably consider QoS extensions to program specifications such as the package definitions in Ada or the Contracts of Eiffel.

5 Conclusion

We have briefly summarized how APIs for RSVP and ATM enable applications to control the QoS of their network connections. We have discussed how well the APIs reflect the needs of typical applications, and how they support the use of advanced operating system concepts. Finally, we propose directions for further development of QoS-aware APIs, namely unification of QoS descriptions with better abstraction from idiosyncrasies of the respective signaling mechanisms, support for flows not involving the main processor, and improved means to cover the entire end-to-end data path with QoS guarantees.

References

[1] CCITT Recommendation G.114. *Mean One Way Propagation Time*, Blue Book (1988), Vol III, Fascicle III.1, pg 84–94.

[2] Coulson, G.; Cambell, A.; Robin, P.; Blair, G.; Papathomas, M.; Sheperd, D. *The Design of a QoS-Controlled ATM Based Communications System in Chorus*, IEEE Journal on Selected Areas in Communications, Vol. 13, No 4, May 1995.

[3] Schmidt, D.; Gokhale, A.; Harrison, T.; Parulkar, G. *A High Performance End System Architecture for Real-Time CORBA*, IEEE Communications Magazine, February 97, pg 72–77.

[4] Campbell, A.; Aurroechea, C.; Hauw, L. *A Review of Quality of Service Architectures*, Proceedings of 4th IFIP International Workshop on QoS (IWQoS'96).

[5] Wright, S.; Jarrett, D. *Analysis of Jitter in Supporting Multimedia Services Agreements over ATM*, Proceedings of IEEE Southeastcon, Raleigh NC, 1995.

[6] Jeffay, K.; Bennet, D. *A rate Based Execution abstraction for multimedia computing*, Proceedings of Fifth International Workshop on Network and Operating Systems support for Digital Audio and Video, Durham NH, 1995.

[7] Leopold, H.; Campbell, A.; Hutchison, D.; Singer, N. *Towards an Integrated Quality of Service Architecture (QOS-A) for distributed Multimedia Communications*, High Performance Networking IV, Danthine, A. and Spaniol, O. (Eds.), Elsevier Science Publishers, 1993 IFIP.

[8] Sahai, A.; Tseng, K.; Wang, W. *A QoS-Controlled Distributed Interactive Multimedia System on ATM Networks*, Proceedings Globecom'95, pg 188–192, IEEE 0-7803-2509-5/95

[9] RFC1633; Braden, Bob; Clark, David; Shenker, Scott. *Integrated Services in the Internet Architecture: an Overview.*, IETF, June 1994.

[10] Braden, Bob; Zhang, Lixia; Berson, Steve; Herzog, Shai; Jamin, Sugih. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification* (work in progress), Internet Draft `draft-ietf-rsvp-spec-14.ps`, November 1996.

[11] Braden, R.; Hoffman, D. *RSVP Application Programming Interface (RAPI) for SunOS/BSD: V4.0* (work in progress), Internet Draft `draft-ietf-rsvp-bsdapi.ps`, August 1996.

[12] Wroclawski, John. *The Use of RSVP with IETF Integrated Services* (work in progress), Internet Draft `draft-ietf-intserv-rsvp-use-01.txt`, October 1996.

[13] The ATM Forum. *ATM User-Network Interface (UNI) Specification, Version 3.1*, `ftp://ftp.atmforum.com/pub/UNI/ver3.1`, Prentice Hall, 1994.

[14] The ATM Forum, Technical Committee. *ATM User-Network Interface (UNI) Signalling Specification, Version 4.0*, `ftp://ftp.atmforum.com/pub/approved-specs/af-sig-0061.000.ps`, The ATM Forum, July 1996.

[15] The ATM Forum, Technical Committee. *Native ATM Services: Semantic Description, Version 1.0*, `ftp://ftp.atmforum.com/pub/approved-specs/af-saa-0048.000.ps`, The ATM Forum, February 1996.

[16] The WinSock Group. *Windows Sockets 2 Application Programming Interface – An Interface for Transparent Network Programming Under Microsoft Windows – Revision 2.2.0*, May 1996.

[17] Tai, C. *WinSock2 ATM Annex*, ATM Forum/96-0190R1 (Anchorage), April 14-19, 1996.

[18] Tai, C. *Mapping of the ATM Forum SAA/API Semantic Description to the Winsock2 API*, ATM Forum/96-0191R1 (Anchorage), April 14-19, 1996.

[19] RFC1363; Partridge, Craig. *A Proposed Flow Specification*, IETF, September 1992.

[20] Periyannan, Alagu; Harford, James J. *XNET ATM API Specifications, Use of XTI to Access ATM*, ATM_forum/96-1169R1

[21] Periyannan, Alagu; Harford, James J. *XNET ATM API Specifications, Use of XSockets to Access ATM*, ATM_forum/96-1169R1

[22] Almesberger, Werner; Le Boudec, Jean-Yves; Oechslin, Philippe. *Application Requested IP over ATM (AREQUIPA) and its Use in the Web*, Global Information Infrastructure (GII) Evolution, pp. 252–260, IOS Press, 1996.

[23] Almesberger, Werner. *Arequipa: Design and Implementation*, `ftp://lrcwww.epfl.ch/pub/arequipa/`

`aq_di-1.tar.gz`, Technical Report 96/213, DI-EPFL, November 1996.

[24] Almesberger, Werner. *Linux ATM API*, `ftp://lrcftp.epfl.ch/pub/linux/atm/api/`, EPFL, July 1996.

[25] Luciani, James V.; Katz, Dave; Piscitello, David; Cole, Bruce. *NBMA Next Hop Resolution Protocol (NHRP)* (work in progress), Internet Draft `draft-ietf-rolc-nhrp-10.txt`, October 1996.

[26] The ATM Forum, Multiprotocol Sub-Working Group. *MPOA Baseline Version 1*, `ftp://ftp.atmforum.com/pub/mpoa/baseline.ps`, September 1996.

[27] Wroclawski, John. *The Use of RSVP with IETF Integrated Services* (work in progress), Internet Draft `draft-ietf-intserv-rsvp-use-01.txt`, October 1996.

[28] Shenker, Scott; Partridge, Craig; Guerin, Roch. *Specification of Guaranteed Quality of Service* (work in progress), Internet Draft `draft-ietf-intserv-guaranteed-svc-07.txt`, February 1997.

[29] ITU-T Recommendation Q.2963.1. *Peak cell rate modification by the connection owner*, ITU, July 1996.

[30] The ATM Forum, Technical Committee. *ATM Forum Traffic Management Specification, Version 4.0*, `ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.ps`, April 1996.

[31] Coulson, G.; Blair, G. *Architectural Principles and Techniques for Distributed Multimedia Application Support in Operating Systems*, ACM Operating Systems Review, October 1995, pg 17–24.

[32] Harford, James J. *AAL SSCS for TCP Applications*, ATMF/96-1751, December 1996 Vancouver

[33] Hoffman, Don. *RSVP API Issues*, `ftp://playground.sun.com/pub/rsvp/ietf_api_slides.ps.Z`, December 1996.

[34] Shanley, T.; Anderson, D. *PCI System Architecture, 3rd Ed.*, Addison Wesley, 1995, ISBN 0-201-40993-3.

[35] GO-MVIP. *ATM Bus specification*, `http://www.mvip.org/`

[36] Hayter, M.; McAuley, D. *The Desk Area Network*, ACM Operating Systems Review, Vol. 25, No.4, pp. 14-21, October 1991.

[37] Ganger, G. R.; Patt, Y. N. *The Process-Flow Model: Examining I/O Performance from a System's Point of View*, Proceedings of ACM SIGMETRICS'93, pg 86–97, ACM 0-89791-581-X/93/0005/0086

[38] Oikawa, S.; Tokuda, H. *Reflection of Developing User-Level Real-Time Thread Packages*, ACM Operating Systems Review, October 1995, pg 63–76.

[39] Nirkhe, V.; Baugher, M. *Quality of Service support for Networked Media Players*, Proceedings COMPCON'95, pg 234–238, IEEE 1063-6390/95