A QoS Adaptive Transport System: Design, Implementation and Experience

Andrew Campbell
COMET Group
Center for Telecommunication Research
Columbia University
New York
USA
http://www.ctr.columbia.edu/~campbell

Geoff Coulson
Distributed Multimedia Research Group
Department of Computing
Lancaster University
Lancaster
LA1 4YR, UK
geoff@comp.lancs.ac.uk

ABSTRACT

Distributed audio and video applications need to adapt to fluctuations in delivered *quality of service (QoS)*. By trading off temporal and spatial quality to available bandwidth, or manipulating the playout time of continuous media in response to variation in delay, audio and video flows can be made to adapt to fluctuating QoS with minimal perceptual distortion. In this paper we describe the implementation of a QoS adaptive transport system that incorporates a QoS oriented API and a range of mechanisms to assist applications in exploiting QoS and adapting to fluctuations in QoS. The system, which is an instantiation of the *QoS Architecture (QoS-A)*, is implemented in a multi ATM switch network environment with Linux based PC end systems and continuous media file servers. A performance evaluation of the system configured to support Video-on-Demand application scenario is presented and discussed.

1. INTRODUCTION

In this paper we describe the design, implementation and evaluation of the *Quality of Service Architecture (QoS-A)* [3] transport system, called the *multimedia enhanced transport system (METS)*. METS supports multi-layer coded flows in a multicast, multimedia networking environment in which client workstations have varying capabilities. In terms of services, METS offers a flexible *quality of service (QoS)* configurable API at the transport layer. In terms of mechanisms, it populates the network layer as well as the transport layer with a number of modules providing control over QoS.

The novel aspects of METS relate to the *protocol*, *QoS maintenance* and *flow management* planes of the QoS-A as illustrated in Figure 1. Briefly, the protocol plane is responsible for transferring media with a target level of QoS.

The QoS maintenance plane is then responsible for the fine grained monitoring and maintenance of the protocol plane. For example, at the transport layer, the QoS maintenance plane monitors rate, loss, jitter and delay and takes remedial action when they fluctuate.

Finally, the flow management plane is responsible for flow establishment (including end-to-end admission control, QoS based routing and resource reservation), QoS mapping, which translates QoS representations between layers, and coarse grained QoS management (e.g., re-negotiation of QoS).

The paper describes *flow scheduling*, *flow shaping* and basic *flow monitoring* in the QoS-A protocol plane. Flow scheduling and shaping are fundamental to the smooth pacing of media onto the network and regulation of media between end-systems. Flow monitoring also plays an important role in measuring the performance of the flow as media is delivered to the receiver. In the QoS maintenance plane, the most important functions are transport QoS management and jitter correction which works in unison with the flow monitor to smooth out network induced jitter. In the flow management plane, METS provides *QoS groups* which encapsulate multicast sessions in which participants with heterogeneous QoS capabilities/ requirements may participate. The flow management plane arranges for per-participant QoS negotiation and resource allocation to take place and is also responsible for informing the user of ongoing QoS performance.

The other key aspect of METS described in this paper is *dynamic QoS adaptation*. This is a flow management plane mechanism designed to exploit the *layered encoding* property of currently popular media formats. An example of a media format with layered encoding is MPEG [13]. MPEG structures video streams in terms of three layers: a coarse or base representation of the signal plus successive enhancement layers. In the case of MPEG1, the base layer (BL) is represented by I pictures and the enhancement layers (E1 and E2) by P and B pictures, respectively. In our dynamic QoS adaptation scheme, *QoS adapters* take remedial action, based on a user supplied QoS policy, to *scale* flows (e.g. by adding or removing enhancement layers

ACM Multimedia 96, Boston MA USA
© 1996 ACM 0-89791-871-1/96/11 ..$3.50

or instantiating filters) when resource availability and/or user QoS requirements change. *Scaling* is a term, first proposed by [9], used to refer to the dynamic manipulation of media flows by objects called *filters* as they pass through a communications channel. Example MPEG filters are coarse grained picture droppers and fine grained low-pass filters (which trade off bandwidth for picture resolution) [33].

The remainder of this paper is structured as follows. We first present, in section 2, a detailed description of the METS API and some of its key internal mechanisms. Then, in section 3, we present a performance evaluation of our METS implementation which throws light on the feasibility of the proposed mechanisms and identifies bottlenecks and pointers for further optimisation. We present our conclusions in section 4.

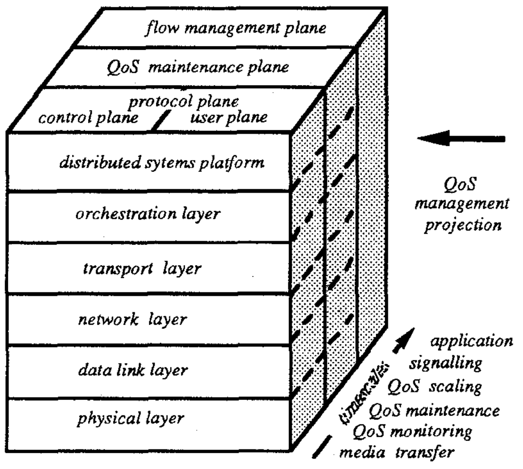


Figure 1: QoS-A Model

2. QoS-A ADAPTIVE TRANSPORT SYSTEM

2.1 Transport Application Programmer's Interface

The METS API is realised as a set of extensions to the Berkeley socket API[1]. QoS is specified at the API in terms of a flow specification and a QoS policy. The flow specification includes parameters such as delay, throughput, jitter etc. The QoS policy allows the user to advise the infrastructure on how to deal with the flow when resource availability changes. For example, the QoS policy may require that the system reduces QoS when resources are in short supply (perhaps by frame dropping [32] or shaping filters [10]), or it may simply require that the user be informed of any degradations. A QoS policy may also require that QoS be *raised* when resources become available; e.g., by adding an enhancement layer to a hierarchically structured video flow [27] [17].

[1] The API is based on a new protocol family called *AF_METS*. By preserving compatibility with the current Berkeley socket API, existing applications (e.g., those using AF_INET) can run unchanged or can easily be modified to take advantage of underlying QoS support; see section 3 for more details of the implementation environment.

The API assumes a client-server model in which servers (i.e. applications offering layer encoded media flows to potential clients) create QoS groups with a given flow specification and QoS policy. After a group has been created and advertised, interested clients may join QoS groups by determining the flow specification and QoS policy, and selecting an appropriate (set of) component part(s) (viz. BL, E1, E2) of the flow. They are then free to join the group (in effect, establish a connection to an underlying multicast SVC) via a set of connection management primitives. In addition to requesting information about the server's QoS profile, clients and servers may also retrieve detailed statistics about membership of a particular group.

The API provides three types of socket: i) *media sockets* used for the transfer of continuous media; these are simplex and QoS configurable via a flow specification and QoS policy; ii) *control sockets* used for the transfer of application level control information; these are full duplex and assured (i.e. they provide a reliable delivery service); and iii) *management sockets* used to interface with the QoS maintenance and flow management planes.

Flow Management Primitives	Parameters
registerSoc	management sock, media sock
changeQoS Req *changeQoS Ack* *changeQoS Nak*	management sock, options (flowSpec | QoSPolicy | maint | monitor | signal | adapt | filter | event) structure (flowSpec | QoSPolicy | maint | monitor | signal | adapt | filter | event) sizeof (flowSpec | QoSPolicy | maint | monitor | ignal | adapt | filter | event) Status
signalQoS	type (signal | event) option (QoSMetric | QoSEvent)

Figure 2: Flow Management Primitives

The API primitives are structured into the following categories:

- *group management* primitives, which allow the user to open a multicast group, get group information and gracefully close a group;
- *connection management* primitives, which allow both clients and servers to join and leave QoS groups; and
- *flow management* primitives, which allow both clients and servers to perform ongoing management and monitoring of flows in which they are participating.

The group management and connection management primitives are conceptually straightforward; it is the flow management primitives (see Figure 2) which provide most of the QoS support. These allow servers and clients to register flows, to change the QoS of flows and to receive

QoS signals associated with a particular flow. Whenever clients and servers create media sockets they register them using the *registerSoc* primitive. This allows the underlying flow manager to interact with the application over the associated management socket to provide monitoring and maintenance information about the on-going flow.At any point during a session, group members may change the QoS negotiated during the connection establishment phase using the *changeQoS* primitive. The options accepted by this primitive are as follows:

- *FlowSpec* is used to submit a new flow specification to renegotiate QoS; we do not exhaustively specify the various fields in a flow specification or QoS policy in this paper; these details are available in [5].
- *QoSPolicy* is used to submit a new QoS policy;
- *Maint*, *Monitor* and *Signal* are used to select QoS maintenance options: in *Maint* mode the transport QoS manager actively maintains the flow - thishis means that it instantiates mechanisms such as those detailed in section 2.2 to attempt to deliver the required QoS in the face of QoS fluctuations in the underlying network/ end-systems; in *Monitor* mode it maintains the flow and also forwards periodic QoS 'signals' (via *signalQoS*) to the user via the management socket; in *Signal* mode it does not maintain the flow but forwards QoS signals anyway;
- *Adapt* is used to change the adaptation mode (viz. discrete or continuous). Discrete mode refers to the addition/ removal of enhancement layers whereas continuous mode involves the instantiation of, for example, source bit rate filters [5], which permit fully continuous adaptation of bit rates;
- *Filter* is used to explicitly select new filters (e.g. jitter filter at the receiver or picture dropping and low pass filters [33] in the network); and
- *Event* is used to allow applications to attach alarms to the occurrence of particular event thresholds. Should the threshold be exceeded then a message is asynchronously sent to the interested party via a *signalQoS* primitive on the management socket.

Note that while a change in QoS initiated by a client only affects the local client's QoS, a change by the server may impact all active clients in the current session.

2.2 METS Mechanisms

We now describe key aspects of the implementation of the METS transport system. This section focuses on the mechanisms used to support QoS maintenance and flow management. Mechanisms in the end-system and in the network are described in separate sub-sections.

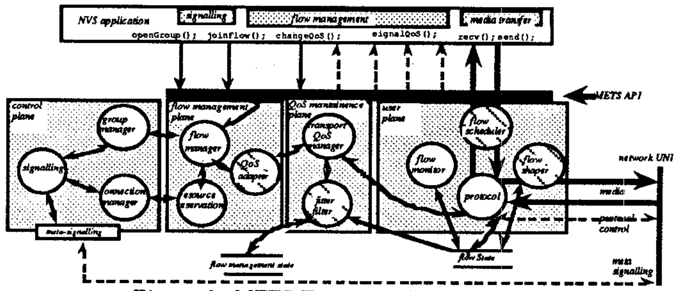


Figure 3: METS Transport System

The transport system is comprised of four implementation modules which map closely to the QoS-A control plane, user plane, QoS maintenance plane and flow management plane, respectively. As illustrated in Figure 3, within each of these per-plane modules METS provides a range of QoS mechanisms. Rather than describe the full range of mechanisms (these are all detailed in [5]), we concentrate here on those mechanisms (highlighted in Figure 3) in the data plane which are responsible for flow scheduling and shaping, those in the QoS maintenance responsible for controlling jitter and those in the flow management plane responsible for coarse grained adaptation to changes in QoS by means of adding/ removing enhancement layers and/or instantiating filters. These are the mechanisms that are the main focus of the performance evaluation in section 3.

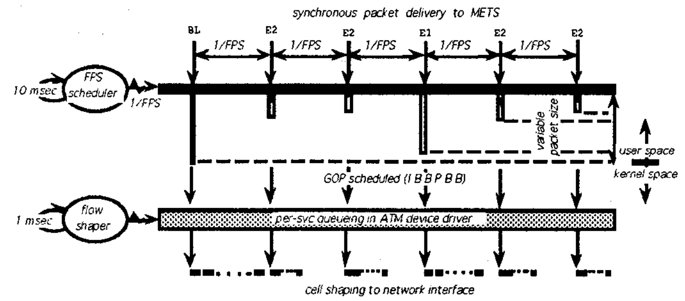
2.2.1 Transport QoS Mechanisms

2.2.1.1 Flow Scheduler

The flow scheduler operates in conjunction with the flow shaper (see Figure 4 below; also see Figure 3) to provide appropriate rate control to ensure per flow bandwidth guarantees and help in jitter management.

The role of the flow scheduler is to schedule application level frames (ALF) [7] on a coarse grained frames-per-second basis. It is implemented in a user space library and is used in both transmission and reception. In transmission, the scheduler uses the standard Linux clock timer to dispatch application level frames to the flow shaper stage according to deadlines derived from the flow specification. A variable bit rate service is provided by isochronously scheduling variable sized packets to the lower layers. At the receiver, the flow scheduler relies on the jitter filter (described below) to provide the scheduling deadlines. In this role, the jitter filter adjusts the deadline of delivered frames but not the rate.

A problem with implementing the flow scheduler in the Linux environment is that it relies on the coarse grained Linux clock which can result in *slippage* or *drift*. To alleviate this problem a drift compensation function [28] is used which takes into account any missed deadlines. If a deadline has been missed then the flow scheduler immediately allows the application to transmit or receive media. The duration of the next scheduling opportunity is then calculated and takes any drift in the isochronous rate of the transmitter into account. The flow scheduler keeps track of any missed deadlines and informs a module in the flow management plane should the number of missed deadlines exceed a pre-defined missed deadline threshold. As described in section 2.1, flow management can upcall applications (via the management socket) to inform them of such events.


synchronous packet delivery to METS
BL
E2
E2
E1
E2
E2
1/FPS
10 msec
FPS scheduler
1/FPS
variable packet size
user space
kernel space
GOP scheduled (I B B P B B)
1 msec
flow shaper
per-svc queueing in ATM device driver
cell shaping to network interface
Figure 4: Flow Scheduling and Shaping

When layered-encoded flows are being used, QoS-A applications are generally designed to only transmit base layer frames when deadlines are being missed consistently. When congestion has cleared, flow management informs the application via a QoS event signal to resume the original rate. Of course, the flow scheduler does not understand the semantics of layered flows.

2.2.1.2 Flow Shaper

The flow shaper provides open loop flow control based on a token bucket scheme that paces cells to the network interface. It is implemented as part of the kernel-level ATM device driver and is invoked every 1 ms using a dedicated hardware timer. The token bucket scheme is a variant of the leaky bucket algorithm. In this scheme, flows accumulate *credits* which represent the number of ATM cells that can be transmitted to the network over the next interval.

The flow shaper maintains the following per flow state: i) a *token budget*, b, which represents the capacity or depth of the token bucket; ii) a *token credit*, r, which represents the remaining credits ($0 < r < b$) left in the token bucket at any point in an interval; iii) a *token refresh rate*, p, which represents the rate at which the token bucket is refilled; and iv) a *token timeout*, t, which represents the number of ticks remaining until the token refresh rate expires and token credits are refreshed.

The token bucket scheme operates in a rather simple fashion. When the token timeout expires the token credit is set to be equal to the token budget. One credit represents one cell transmission opportunity. When the flow shaper executes, it visits all the per flow queues in a round robin fashion and, if the queue contains cells and has available credits, then the shaper transmits one cell from the queue and decrements the token credit state variable. This achieves the desire effect of interleaving of cells from different SVCs onto the network.

There are two possible outcomes at the end of a token refresh interval: either all the cells have been drained or cells remain in the queue. If cells remain in the queue then all the credits have been used during the interval. If no cells remained queued then all queued cells have been dispatched to the network during the interval with ($0 <= r < b$) credits potentially remaining. At the end of a token refresh cycle any credits remaining at the end of the interval are lost.

2.2.1.3 Jitter Filter

The objective of the jitter filter is to restore the timing properties of the originally transmitted data flow at the source before the flow is delivered to the playout device at the sink (i.e. to remove end-system and network induced jitter).

It is a straightforward process to recover the original timing in communication systems that provide a hard bound on delay (it is only required to buffer packets at the sink until time $T+D$, where T is a timestamp placed in the packet at the source and D is the bounded maximum end-to-end delay). Many networks, however, are unable to guarantee a hard maximum delay bound; in such cases the receiver must form a continuously updated estimate of the maximum delay as the basis on which to calculate buffering time. In the algorithm we have adopted (based on work by [13] [24] and using synchronised clocks [21]) statistical analysis of per-packet delay is used to estimate the maximum delay; i.e. $D = d + r * s$, where d is the average delay (see below), s is the standard deviation (see below) and r is a filter coefficient which is chosen as a function of the form of the distribution curve and the number of failures that one is ready to accept [14].

Each failure corresponds to a transmission delay larger than the estimated maximum. Packets that arrive after D are considered to be too late to help reconstruct the signal and, in this case, flow management is informed of a late packet event and the packet is dropped. A popular value for r is 2, which corresponds to an accepted loss of about 1% if one assumes a Gaussian distribution of the delays [14]. From an intuitive point of view the $2s$ term is used to set the playout time to be "far enough" beyond the delay estimate so that only an acceptable fraction of the arriving packets should be lost due to late arrival [24]. For a full discussion of these issues see [13].

The jitter filter continuously estimates the average delay and standard deviation. It is based on the 'low pass filtering algorithm' used in TCP for the estimation of the acknowledgement delay time [13] and operates as follows. When a METS packet arrives, the transmission delay, t, is determined as the difference between the received time and the emission time stamp. The average delay and standard deviation are then calculated as follows: $d = d_{old} + a(t - d)$, $s = s_{old} + b(|t - d| - s)$. The constants a and b ($a, b < 1$) are smoothing coefficients. Typical vales are 1/8 and 1/16, respectively, which make the calculation of d and s particularly efficient.

Having determined a method for calculating a continuously updated estimate of D, it is necessary to decide on an appropriate time granularity at which D, and thus the playout time, should be updated. It is clearly not desirable to continuously alter the playout time on a frame by frame basis. In our scheme, the playout time is only used to influence playout at the beginning of each MPEG 'group of pictures' (GOP). The objective (as in the *vat* audio tool [15]) is to keep any adjustments as imperceptible as possible to the human visual system. If flow management determines that too many losses have occurred it calculates a new playout time. When no losses are detected the same playout point is adhered to. If no losses occur over a number of monitoring periods the playout time is recalculated. In this way the jitter filter can, wherever possible, 'pull in' the playout estimate to decrease end-to-end delay.

2.2.2 Network QoS Mechanisms

2.2.2.1 Distributed QoS Adapter

The distributed QoS adapter resides in the flow management plane and is responsible for informing source and sink applications (in a manner controlled by the QoS policy) when additional resources become available that can be used to support additional enhancement layers (i.e. E1 and E2) in layer-encoded flows. Note that in our scheme the QoS of the base layer (BL) is guaranteed. This is based on end-to-end admission control and resource reservation (see [5] for full details). The base layer is, therefore, independent of the QoS adaptation process and the latter only manages the enhancement layers. The QoS adapter may also unilaterally choose to discard a enhancement layer (again, dependent on the QoS policy) if resources become scarce. Instances of the QoS adapter are present in both end systems and network switches.

The QoS adapter operates by periodically probing the network to test for resources that have just become available or unavailable. At the start of each cycle (called an *era*), the local QoS adapter instance of each receiver determines its local resource availability and sends a RES signalling message containing an indication of the additional bandwidth required to support the addition of enhancement layers. This is inspired by a similar mechanism in the RSVP protocol [26]. In the case where the adaptation protocol determines that the congestion[1] has been detected over the preceding interval it reduces its request (from E2 to E1 or E1 to 0) before sending the RES message. Once the QoS adaptation protocol notes that congestion has passed; that is to say, when it notes that no congestion indications have been received over the last interval, it increases its bandwidth demands accordingly. RES messages are forwarded toward the *core switch* [2] of the multicast SVC (see section 3.1). These messages can be updated on their way to the source by all intervening switches to reflect the resource availability at traversed switches. When the RES message arrives at the source, it indicates the advertised rate (i.e., bandwidth available) to the source over the next era. The advertised rate can be zero, E1 or E2 cells/second. Because we support the concept of end-to-end adaptivity in the QoS-A, the adaptation protocol at the source also takes into account the end-system resource availability. This allows the source side adaptation protocol to reduce the advertised rate (e.g., reducing E2 to E1 or E1 to zero) in the RES messages if need be. Following this, the source informs the application (via its management socket) should there have been a change in the advertised rate over successive intervals. If there is a change, the QoS adaptation protocol requests the flow shaper to reconfigure the per token bucket flow state (see section 2.2.1.2) based on the new advertised rate. The source then responds to the RES message by multicasting an ADAPT message to all receivers indicating the available bandwidth over the coming era. When it receives an ADAPT message indicating that it may add or must remove an enhancement layer, the QoS adapter at a receiver informs the application so that it can arrange to start dealing with a modified flow.

The adaptation protocol is complicated by the potential presence of QoS filters and media selectors (see section 2.2.2.2) at switches in the network. QoS filters and media selectors are a concern in that the bandwidth requirement on the input side (i.e., upstream) of a filter/media selector may not be equal to the bandwidth requirement on the output side (i.e., downstream). This issue is resolved by treating network nodes supporting filters and media selection as *virtual sources* and/ or *virtual sinks*. To understand the concept of virtual sources and sinks refer to Figure 5. If a filter is placed on the located at the `rook` ATM switch a client consuming media at `dwp` from a source at `atc` would consider `rook` a virtual source. Similarly, `atc` would consider `rook` as a virtual receiver.

In order to support QoS adaptation in a multicast environment, switches must be able to merge RES messages from different down stream branches of the multicast distribution tree. Merging is rather simple in our system: a merged RES message carries a min:max pair (i.e., either {0,0}, {0,E1} or {E1,E2}) which corresponds to the aggregation of all possible enhancement requests. As a consequence of this process the QoS adaptation protocol provides support for merging and forwarding of the RES messages in the network. It does this by building periodic merged messages based on all RES messages received over the last *RES interval.* Conversely, if no RES messages have been received at a merge point during the last RES interval then the timer is simply reset and no further action is taken.

Another issue addressed by the adaptation algorithm is the fair allocation of residual bandwidth resources among all flows competing to add new enhancement layers. Residual resources are those remaining once all base layer reservations are accounted for. Each switch and end-system use a simple 'fair share' bandwidth allocation algorithm which allocates the residual bandwidth equally among competing flows. This plays an important part in providing the advertised rate, i.e., the portion of the residual bandwidth which can be used by a particular flow as it traverses a switch. The fair share algorithm only allocates

[1] If a switch detects that queues are building beyond a pre-defined threshold the switch sets the congestion bit in the ATM header of cells. The flow monitor detects this condition and notes it in the flow's congestion state (see Figure 4). At the end of an era the QoS adaptation protocol reads the congestion state and reset it to uncongested. See [6] for full details .

resources to enhancement layers if it can meet one of the min:max constraints in the RES message. In the case where a flow's fair share is insufficient to support an enhancement layer then resources are returned to the pool and the advertised rate appropriately adjusted. For full details of this scheme see [5].

2.2.2.2 Media Selector

The media selector resides in network switches and works with the QoS adapter to ensure that the appropriate combinations of base layer and enhancement layer(s) of flows are forwarded to appropriate switch output ports of the ATM switch.

As an example of the operation of the media selector, consider a multicast virtual connection between a source located at the `atc` end-system and two clients at `mr-little` and `dwp` (see Figure 5). The source node, `atc`, transmits full resolution video (i.e. BL+E1+E2), `dwp` selects the E1 resolution and `mr-little` selects the E1 and E2 resolutions. If an E1 cell arrives at the `chuff` switch and the network can meet both recipients' QoS demands then the media selector will forward the cell to both `dwp` and `mr-little`. On the other hand, when an E2 cell arrives on the same connection the media selector will forward it to `mr-little` only.

In order to carry out such functions, the media selector must be able to delimit individual AAL5 frames in the cell stream so that BL, E1 and E2 frames can be distinguished (BL, E1 and E2 packets are all carried on the same SVC)[1]. To delineate AAL5 frames, the media selector exploits the ATM user-to-user bit: a user-to-user bit of 1 represents the first cell of an AAL5 packet, and the first 16 bits of the AAL5 payload then represent the type of the frame (BL, E1 or E2). It is important to note that the media selector does not have to buffer AAL5 packets to determine the type before forwarding. All that is required is to continuously monitor the user-to-user bit of cells traversing a switch and to perform a 16 bit comparison on the first 16 bits of the payload of the first cell of a new AAL5 frame. As a result cells are streamed through the switch with very little additional delay over the unmodified switch code. A fundamental assumption is that the switch architecture is capable of supporting such a scheme in software; see section 3.1 for details of the ATM switches used in the implementation.

3. Experimental Evaluation

3.1 Experimental Environment

The METS testbed consists of 4 Linux based PCs and two RAID-3 storage servers. Each PC and storage server is equipped with ATM network interface controllers (NICs) for connection to the ATM network. The storage servers, network interface cards and ATM switches are all experimental devices supplied by Olivetti Research Labs (see [11]).

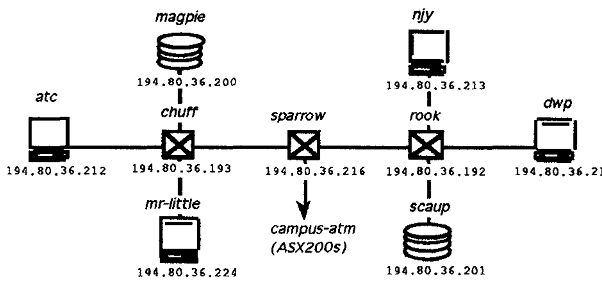


Figure 5: METS Testbed

The Olivetti NICs deliver ATM cells at 100 Mbps and interfaces to a standard PC ISA bus. An AAL5 adaptation layer is implemented in software in the Linux device driver [20]. The ATM switches are 4x4's and share the same port design as the NICs. The switches have a measured aggregate switching capability of 200 K cells/sec. The switch architecture comprises an ARM processor connected via DMA channels to the ports. As switching is performed in software on the ARM processor it has proved relatively easy to modify the switches to support the QoS adapter protocol, multicasting and media selection. The multicasting implementation is based on the notion of a designated 'core switch' at which all clients involved in a single flow 'rendezvous' with the output provided by the server [2]. For example, an appropriate core switch for a multicast flow sourced at `scaup` and sinked at `njy` and `dwp` would be `rook`. In the current implementation, the core switch is not changed during the lifetime of a flow even if the newly joined recipients cause the topology to become less than optimal.

As illustrated in Figure 5 the testbed includes the following client nodes: the `atc` and `dwp` end-systems, which are 90 MHz Pentiums, and the `mr-little` and `njy` end-systems which are 66 MHz 486 machines. The RAID-3 storage servers (the `magpie` and `scaup` end-systems) utilise the ARM 610 RISC processor and incorporate five SCSI interface controllers and disk drives.

3.2 The Experiments

In order to gain experience with the METS testbed (which is based on a native ATM communication stack) and evaluate its performance, we have designed a set of experimental test suites:

- bandwidth analysis, which evaluates the ability of the flow scheduling, flow shaping and ATM infrastructure to respond to varying bandwidth demands;
- loss analysis, which evaluates the role of the flow scheduler and shaper in reducing losses;
- delay analysis, which evaluates the effect of multiple flows on delay distributions;
- jitter filtering analysis, which evaluates the jitter filter's delay estimation and playout algorithms at

1 The media selector must also know which output ports of which multicast SVCs require which AAL5 frames to be copied to them; this information is obtained from the QoS adapter.

the receiver; and

- adaptation analysis, which evaluates the QoS adapter mechanisms at the end-systems and network.

All performance measurements are taken from video flows sourced at the atc end-system and played out at the mr-little and dwp end-systems. The designated core ATM switch used during the multicast sessions is chuff. All measurements are captured and logged at the atc and dwp end-systems. The distance between the server and clients is 3 hops (i.e. flows emanating from atc are played out at dwp traversing the chuff, sparrow and rook ATM switches, respectively). In all cases the server and clients maintain logs of METS packet departure and arrivals times, respectively. The Network Time Protocol [21] provides global timing facilities between all end-systems involved in the experimentation and logging process. The client log includes arrival times, absolute delay and jitter of received packets and loss of METS packets at the transport level.

3.3 Bandwidth Analysis

The object of this experiment is to determine the maximum possible transmission rate achievable from application space to the network. This is a measure of the maximum throughput of the METS communications stack in the end-system. The Canyon video clip is transmitted as rapidly as possible. Additionally, the traffic shaper and ATM device driver receive interrupt are disabled. Media traverses the METS transport system, AAL5 and ATM NIC without consideration of the receiver's ability to consume media.

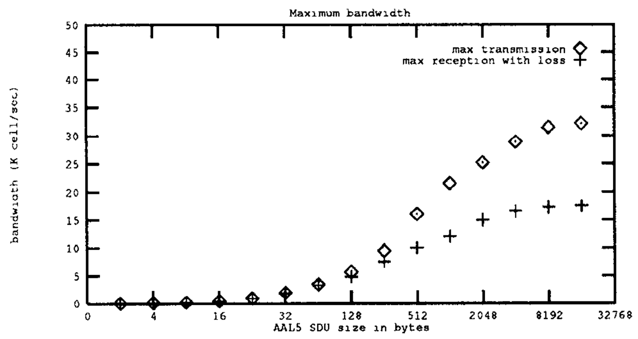


Figure 6: Maximum Transmission and Reception Rates

The results are presented in the upper curve of Figure 6 which shows the throughput achieved when the METS packet size is varied incrementally between 1 byte and 32 KBytes. The maximum transmission rate attained is 32 K cells/sec (13.6 Mbps) compared to the theoretical NIC line rate of over 235 K cells/sec.

This experiment (the lower curve in Figure 6) measures the 'goodput' achieved between a server and a client using flow scheduling and open loop flow control (through the flow shaper). Goodput is the maximum transmission rate at which cells can be injected into the network and consumed by the receiver with no significant loss resulting. To achieve the optimum goodput cells are 'paced' into the network at a rate agreed to between the transmitter and the receiver. A maximum goodput of 17.5 K cells/sec (7.4 Mbps) was measured. This is 54% of the maximum transmission speed figure.

3.4 Loss Analysis

The objective of this experiment (see Figure 7) is to determine the percentage of lost packets at the receiver as a function of the number of video flows received. Video was played at 12 fps. Tests were performed both with and without traffic shaping at the source.

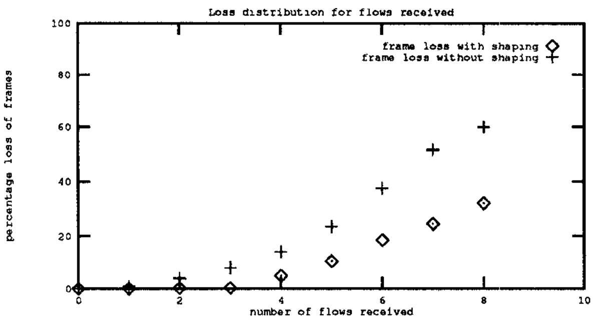


Figure 7: Frame Loss With and Without Traffic Shaping

The upper curve in Figure 7 depicts the METS frane loss occurring from an unregulated source while the lower curve depicts the loss resulting from flows shaped by the METS flow shaper. The number of received video flows varies from 1 to 8. The maximum percentage loss measured at the receiver varies between 33% and 60% for regulated and unregulated traffic, respectively. In the case of unregulated traffic, performance degrades rapidly when four flows are received simultaneously. This represents a loss of greater than 10% - which is starting to be significant to the end-user. Regulated traffic on the other hand only exhibits 10% loss when the number of flows approaches 5.

3.5 Delay Analysis

3.5.1 Single Flow Delay Distribution

The configuration of this experiment is a lightly loaded network with one video flow running at 24 fps. The average delay (see Figure 8) measured by the transport protocol at the receiver is 4 ms. The minimum and maximum delays recorded are 2 ms and 19 ms, respectively, with a standard deviation of 2 ms.

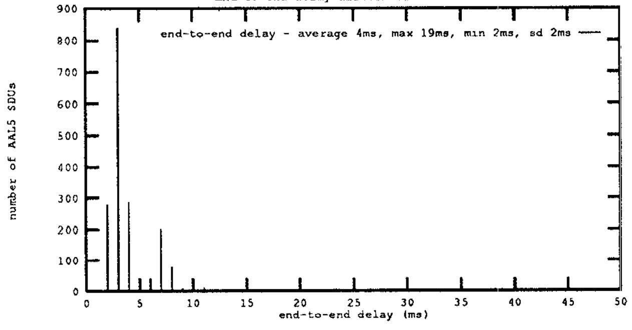


Figure 8: Per-Packet e2e Delay Distribution for One Flow

3.5.2 Effect of Multiple Flows on Delay

This second experiment in the delay suite measures the delay statistics experienced when the number of transmitted flows is varied between one and eight. As can be seen in the lower curve of Figure 9, there is little difference, just 6 ms, in the *average* delay measured as the number of flows increase. Variation in the *maximum* delay experienced is, however, significantly large at 42 ms.

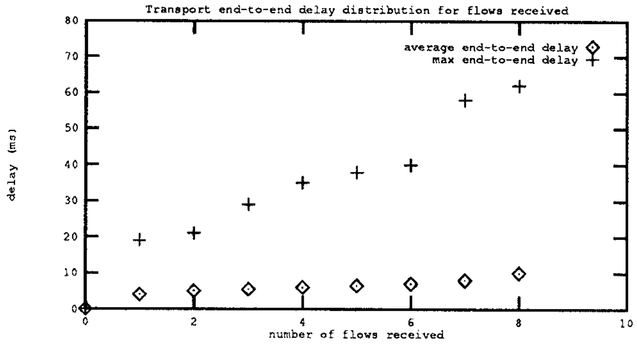


Figure 9 End-to-end Delay Distribution

3.6 Jitter Filtering Analysis

This experiment investigates the ability of the jitter filtering mechanism (see section 2.2.1.3) to adaptively adjust the playout delay experienced by flows at the receiver to meet end-to-end delay and jitter requirements. The source packetises a single video stream and attempts to transmit it at an isochronous rate of 24 fps. At the same time, four other background flows are simultaneously being handled by the same receiver.

In Figure 10, the playout curve tracks the arrival time curve to the first point of loss - the region between 42500 and 43000 ms. The first point of inflection represents a sudden increase in the end-to-end delay and subsequent loss of a number of METS packets. The second point of inflection (between 43000 and 43500 ms) also represents a large increase in measured delay, subsequent loss of packets and then the simultaneous arrival of a group of packets at the receiver. It can be seen that significant packet loss occurs between 43000 and 43500 ms due to underestimation of the maximum end-to-end delay.

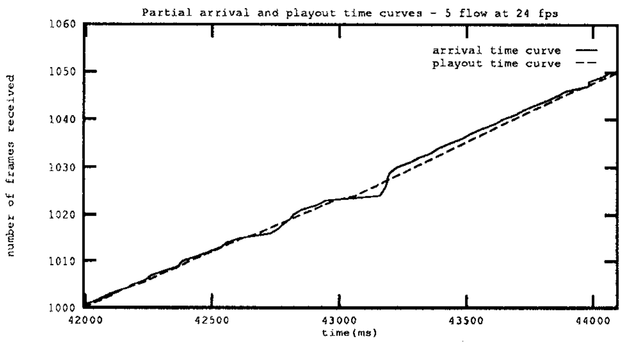


Figure 10: Arrival and Playout Time Distribution High Load

3.7 QoS Adaptation Analysis

The final experiment investigates the performance of the distributed QoS adaptation mechanism. In the experiment, the network provides "hard" guarantees to the base layer (BL) of a multi-layer flow but gives no such guarantees to the enhancement layers (E1 and E2). The enhancement layers must compete for residual bandwidth with all other flows as discussed in section 2.2.2.1.

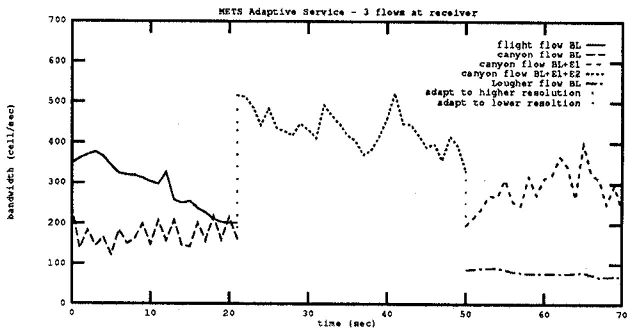


Figure 11: QoS Adaptation Scenario

In the experiment (see Figure 11) the receiver selects three flows for playout in the first instance. These are:

- a "Canyon.mpg" video flow (selecting the BL, E1 and E2 components) at 24 fps;
- a "Lougher.mpg" video flow (selecting the BL only) at 5 fps; and
- a "Flight.mpg" video flow (selecting BL) only at 5 fps.

The scenario shows the consumption of the Canyon and Flight video clips beginning at time zero. Both base layers are supported. The QoS adapter determines that none of the Canyon flow's higher resolutions can be accommodated given the available resources. Twenty seconds into the scenario trace the Flight video terminates thus freeing up resources. At this point the QoS adapter judges that the highest resolution of the Canyon video (i.e. BL+E1+E2) may be displayed.

This situation remains until the user chooses to display the Lougher video 50 seconds into the trace. Resources are thus allocated to meet the base layer QoS requirements of the new video. The QoS adapter protocol sends a RES message toward the core requesting resources to meet the highest resolution (E2) of the Canyon.mpg video. In this instance, however, there are insufficient resources available to meet the QoS requirements of the highest resolution although resources are adequate to support the lower resolution (E1).

3.8 Discussion of Results

The results of the first three experiments show the raw performance limitations of the experimental implementation. Regarding the bandwidth analysis, the bottleneck appears to be a combination of the limitations imposed by the ISA system bus and the experimental prototype NIC. For example, the host CPU must perform the segmentation and reassembly of AAL5 SDUs, and copying of cells to/ from host memory. Other limitations of the NIC are limited buffering and the fact that there is an interrupt at the receiver for each ATM cell delivered by the NIC[1]. Although an obvious solution to these problems would be to employ a NIC that offered AAL processing and DMA data copying on-board, the experimental NIC is very valuable to us for experimentation as we have the flexibility of host CPU control of functions such as flow scheduling and shaping.

As presented in Figure 6, receivers in our testbed can accommodate a maximum transmission rate of up to 8 K cells/sec after which they start to drop cells. In the case of the current implementation the loss of one cell causes an entire AAL5 packet to be lost. Thus, the loss of a few isolated cells can have a seemingly disproportionate effect on the delivered bandwidth, especially if packets are large. By adding 'dummy' replacement cells at the receiver whenever cell loss is detected, it should be possible to improve the results shown in Figure 6. Our experiments, though, have shown that traffic shaping is also a valuable technique for reducing cell loss. Receivers could consume four *regulated* Canyon flows at 12 fps with an overall frame loss of 10%. This rose to 33% loss for eight flows consumed. Corresponding performance for the unregulated case was measured to be 15% and 60% for four and eight flows consumed, respectively.

The loss analysis experiments show that traffic shaping is a valuable technique for reducing cell loss. Receivers could consume four *regulated* Canyon flows at 12 fps with an overall frame loss of 10%. This rose to 33% loss for eight flows consumed. Corresponding performance for the unregulated case was measured to be 15% and 60% for four and eight flows consumed, respectively.

The delay analysis highlighted that, while there were minor variations in the average end-to-end delay as more flows were consumed, there was considerable variation in the maximum end-to-end delay. The average delay difference experienced between one flow and 8 flows was found to be 6 ms. In contrast, the maximum delay difference measured was 42 ms. Since the jitter filter works by estimating the maximum delay the latter results are significant.

Turning to the jitter filter analysis itself, the jitter filter proved to be highly successful. One remaining problem is that sudden, unexpectedly large, jitter such as that evident in Figure 10 is difficult to deal with. A solution is to increase the value of the 'confidence factor' $2s$ (see section 2.2.1.3); overestimation of this kind, however, can lead to large buffer requirements at the receiver and an overall increase in end-to-end delay. Note that the jitter filter's playout algorithm adjusts to fluctuations detected in the measured maximum delay but that these adjustments are rather conservative in nature. Conservatism, however, appears to be a good policy in the long term since, as is seen, the estimate is quickly back on track. Appropriate choice of filter coefficients, which influence the responsiveness of the playout algorithm to track changes in arrival patterns, is another key issue. Choosing larger coefficients would cause the playout to mirror fluctuations in the arrival time distribution; this, however, is not always the best policy. Optimally, the playout time should evolve in response to trends in the arrival time patterns and not in response to occasional spikes. During experimentation, the coefficient values of a=1/8 and b=1/16 were determined to be the most appropriate for our local area ATM environment. For a full discussion on filter coefficients see [16] and [24]

Regarding the QoS adaptation experiments, it was demonstrated that the distributed QoS adaptor scheme successfully maximises the utilisation of the available bandwidth by dynamically adjusting the resolutions of flows to meet the specific needs of different clients. While discrete adaptation was noticeable, the resulting perceptible changes were not unacceptable to casual users and the concept appears to be very promising. One disadvantage of the currently implemented scheme, however, is that discrete fluctuations may be undesirable in certain application contexts. A solution to this problem is the notion of *continuous adaptation* using the dynamic rate shaping filter [10]. The integration of dynamic rate shaping into the current testbed is an issue for future work.

4. Concluding Remarks

This paper has described and evaluated an instantiation of the QoS-A transport layer in a local ATM environment. Although the implementation is successful, it remains to be seen how well the design - particularly the QoS adapter protocol and multicasting mechanisms - will translate to a wide area context with large numbers of receivers with heterogeneous QoS demands. The implementation was carried out in a conventional OS environment (Linux) in contrast to our previous implementation work [8] which was embedded in a more deterministic system based on the Chorus microkernel. The fact that two separate instantiations of the QoS-A now exist provides evidence that the QoS-A framework is valid and useful.

It has been demonstrated by performance evaluation that the METS QoS mechanisms (viz. flow scheduling, flow shaping and jitter filtering), which were specifically designed to operate in an adaptive environment, can successfully mask many of the finer-grained effects of fluctuating network and end-system resource availability. In addition, we have shown how the distributed QoS adapter protocol permits a far grosser level of adaptation while still maintaining a high degree of transparency for applications. We believe that our results vindicate the QoS-A approach of *selective transparency* of QoS mechanisms. Applications can choose QoS management strategies from a range of possibilities on a continuum from i) delegating all responsibility for dynamic QoS management to the underlying system, to ii) simply exploiting API upcalls to perform all dynamic QoS management for themselves (cf. *vat*, *vic* etc.). In all cases, applications choose their

[1] The ISA ATM device driver, however, reduces this overhead by checking whether any cells have arrived at the end of each receive interrupt cycle.

preferred strategy by appropriately initialising a QoS policy.

Finally, as well as throwing light on performance measures, our implementation has also made possible a qualitative assessment of the METS API. The relative complexity of the API was, in fact, found to be fairly easy to use in practice. The separation of concerns achieved through the use of separate sockets for data, control and management proved successful and application writers porting the NVS system from a standard Berkeley sockets environment found little difficulty. The potential complexity of QoS specification was largely avoided though the use of sensible default values for QoS parameters and policies. In the future we intend to further ease the complexities of QoS specification by dynamically deriving per-user QoS preferences from live sessions and storing them in a database for future application runs. At Columbia University we are experimenting with an open programmable networking environment called **xbind** [19] based on CORBA technology. We are currently investigating the suitability of our transport system, adaptation protocol and QoS filtering techniques for wired-to-wireless ATM environments [23]; that is, we are augmenting **xbind** with QoS adaptive multimedia applications, QoS adaptive transports [12] and dynamic rate shaping to provide *QoS controlled mobility* [6] in ATM networks and their wireless extension.

References

[1] Aurrecoechea, C., Campbell, A. T., and L. Hauw "A Survey of Quality of Service Architectures", *Multimedia Systems Journal,* 1996 (to appear).

[2] Ballardie, T., Francis, P. and Jon Crowcroft, "Core Based Tree (CBT) An Architecture for Scalable Inter-Domain Multicast Routing", *Proc. ACM SIGCOMM '93*, San Francisco, USA, 1993.

[3] Campbell, A.T., Coulson, G., García, F., Hutchison, D., and H. Leopold, "Integrated Quality of Service for Multimedia Communications", *Proc. IEEE Infocom'93*, Hotel Nikko, San Francisco, CA, March 1993.

[4] Campbell, A.T., Coulson, G. and Hutchison, D., "A Quality of Service Architecture", *ACM Computer Communications Review*, April 1994.

[5] Campbell, A.T., "A Quality of Service Architecture", Ph.D Thesis, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1996, http://www.ctr.columbia.edu /~campbell/papers/thesis.ps.gz

[6] Campbell, A.T., "Towards End-to-End Programmability for QoS Controlled Mobility in ATM Networks and their Wireless Extension", *Proc. 3rd International Workshop on Mobile Multimedia Communications,* Princeton, September 1996, (to be presented).

[7] Clark, D., and D.L. Tennenhouse, "Architectural Consideration for a New Generation of Protocols", *Proc. ACM SIGCOMM '90,* Philadelphia, USA, 1990.

[8] Coulson, G., Campbell, A. T. and P. Robin, "Design of a QoS Controlled ATM Based Communication System in Chorus", *IEEE Journal of Selected Areas in Communications (JSAC)*, Special Issue on ATM LANs: Implementation and Experiences with Emerging Technology, 1995.

[9] Delgrossi, L., Halstrinck, C., Henhmann, D.B, Herrtwich R.G, Krone, J., Sandvoss, C., and C. Vogt, "Media Scaling for Audio-visual Communication with the Heidelberg Transport System", *Proc. ACM Multimedia'93*, Anaheim, USA, 1993.

[10] Eleftheriadis, A., and D. Anastassiou, "Meeting Arbitrary QoS Constraints Using Dynamic Rate Shaping of Code Digital Video", *Proc. Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, New Hampshire, USA, 1995

[11] French, L.J., Wilson, I.D., D.P. Gilmurry, "ATMOS II User Manual", Olivetti Research Limited, 24a Trumptington Street, Cambridge, 1994.

[12] Huard, J-F., I. Inoue, A. A. Lazar and H. Yamanaka, "Meeting QOS Guarantees by End-to-End QOS Monitoring and Adaptation", *Proc. Workshop on Multimedia and Collaborative Environments of the Fifth IEEE International Symposium On High Performance Distributed Computing (HPDC-5)*, Syracuse NY, Aug. 1996 and http://www.ctr.columbia.edu/~jfhuard/research.html

[13] H.262, "Information Technology - Generic Coding of Moving Pictures and Associated Audio", Committee Draft, ISO/IEC 13818-2, International Standards Organisation, UK, March 1994.

[14] Huitema, C., "Routing in the Internet", *Prentice Hall*, ISBN 1-13-132192-7, 1994.

[15] Jacobson, V, " Congestion Avoidance and Control", *Proc. ACM SIGCOMM'88*, Stanford, 1988.

[16] Jacobson, V., "VAT: Visual Audio Tool", *vat manual pages*, 1993.

[17] Kanakia, H., Mishra, P., and A. Reibman, "An Adaptive Congestion Control Scheme for Real Time Packet Video Transport", *Proc. ACM SIGCOMM '93*, San Francisco, USA, October 1993.

[18] Keshav and Saran, "Semantics and Implementation of a Native-Mode ATM Protocol Stack", Bell Labs Technical Memorandum, http:// www.cs.att.com/ csrc/keshav/papers.html, 1994.

[19] Project **xbind** http://www.ctr.columbia.edu/comet/ xbind/ xbind.html

[20] Lunn, A. S., "A Mini Cell Architecture for Multimedia Systems"", PhD Thesis, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, Sept 1995.

[21] Mills, D., "Network Time Protocol (Version 3): Specification, Implementation and Analysis", *Internet Request for Comments*, No. 1305, Networking Information Center, SRI International, Menlo Park, CA, March 1992.

[22] Partridge, C., "A Proposed Flow Specification; RFC-1363" *Internet Request for Comments*, No. 1363, Network Information Center, SRI International, Menlo Park, CA, September 1990.

[23] Porter, J. and A. Hopper, "An ATM Based Protocol for Wireless LANs", *Technical Report*, Olivetti Research Limited, April 1994.

[24] Ramjee, R., Kurose, J., Towsley, D., and H. Schulzrinne, "Adaptive Playout Mechanisms for Packetised Audio Applications in the Wide-Area Network", *Proc. IEEE Infocom'93*, 1994.

[25] Rowe, L., and Smith, "A Continuous Media Player", *Proc. Third International Workshop on Network and Operating System Support for Digital Audio and Video,* San Diego, 1992.

[26] Zhang, L. et al., "Resource Reservation Protocol (RSVP) - Version I Functional Specification", Working Draft, draft-ietf-rsvp-spec-07.ps, 1995.

[27] Shacham, N, "Multipoint Communication by Hierarchically Encoded Data", *Proc. IEEE INFOCOM'92*, Florence, Italy, Vol. 3, pp 2107-2114, 1992.

[28] Simpson, S., "A Dynamic Rate Shaping Mechanism", *Internal Report*, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1995.

[29] Steenstrup, M., "Fair Share for Resource Allocation", pre-print.

[30] Tokuda H. and T. Kitayama ,"Dynamic QOS Control Based on Real-Time Treads" *Proc. Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, Lancaster University, Lancaster LA1 4YR, UK, 1993.

[31] Turner, J. "ATM-Soft: A Mini-Proposal for ATM Network Control Using Soft State", *Technical Note*, Washington University, 1995.

[32] Yeadon, N., Garcia, F., Campbell, A. T. and D. Hutchison, "QoS Adaptation and Flow Filtering in ATM Networks", *2nd International Workshop on Advanced Teleservices and High Speed Communication Architectures*, Heidelberg, Germany.

[33] Yeadon, N., "QoS Filtering", *Ph.D Thesis*, Department of Computing, Lancaster University, Lancaster LA1 4YR, UK, 1996.