QoS Adaptive Transports: Delivering Scalable Media to the Desktop

Andrew T. Campbell, Columbia University
Geoff Coulson, Lancaster University

Abstract

By trading off temporal and spatial quality with 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 network conditions with minimal perceptual distortion. In this article the authors describe the implementation of an adaptive transport system that incorporates a QoS-oriented API and a range of QoS mechanisms that best assist multimedia applications in adapting to fluctuations in the delivered network QoS. The system, which is an instantiation of the transport and network layers of a QoS architecture, 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 a video-on-demand application scenario is presented and discussed. A novel aspect of the system is the implementation of a "QoS adaptation" algorithm which allows applications to delegate to the transport system responsibility for augmenting or reducing the perceptual quality of video and audio flows when network resource availability increases or decreases, respectively.

The interplay between application-level quality of service (QoS) requirements, the delivery of scalable audio and video flows, and end-to-end communication support is an active area of research [1]. Existing hierarchical coding techniques used by coders such as Motion Picture Experts Group version 1 (MPEG-1), MPEG-2, and H.261 make possible a range of creative adaptation strategies whereby fluctuating bandwidth availability can be accommodated by selectively adding or removing coding layers.

Transport mechanisms that intrinsically supported adaptive approaches were first recognized in the late '70s by Cohen [2] as part of research in carrying voice over packet-switched networks. More recently, adaptive mechanisms have been introduced as part of the Internet suite of application-level multimedia tools (e.g., `vat` [3], `ivs` [4], and `vic` [5]) for dealing with fluctuations in the delivered QoS. For example, `vat`, which is used for voice conferencing, recreates the timing characteristics of voice flows by having the sender timestamp ongoing voice samples. The receiver then uses these timestamps as a basis for reconstructing the initial flow, removing any network-induced jitter prior to playout.

In this article we describe the design, implementation, and evaluation of the QoS architecture (QoS-A) [6, 7] transport system, called the *multimedia enhanced transport system* (*METS*). METS supports multilayer coded flows in a multicast, multimedia networking environment in which client workstations have varying capabilities. In terms of services, METS offers a flexible QoS-configurable application programming interface (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. While application-level multimedia tools such as `vic` and `vat` incorporate QoS mechanisms for adaptivity in the application itself, we argue that QoS adaptive mechanisms should be part of the underlying transport system itself.

The novel aspects of METS relate to the *protocol*, *QoS maintenance*, and *flow management* planes of the QoS-A as illustrated in Fig. 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, and QoS-based routing and resource reservation), *QoS mapping*, which translates QoS representations between layers, and *coarse-grained QoS management* (e.g., renegotiation of QoS).

This article 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.

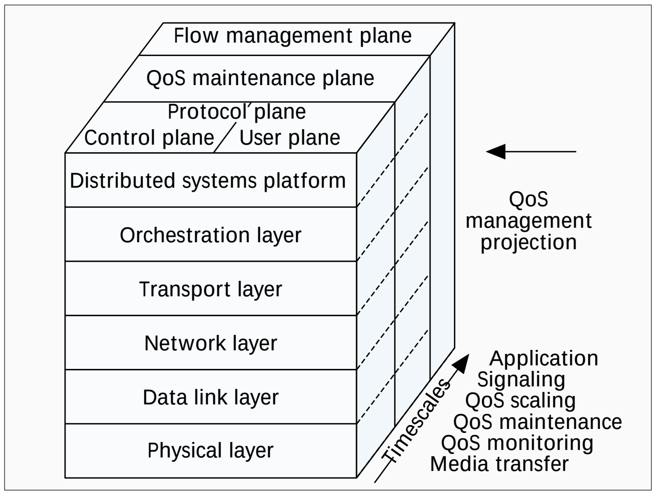
The other key aspect of METS described in this article is *dynamic QoS adaptation*. This is a flow management plane mechanism designed to exploit the layered encoding property of currently popular media formats. An example of a media format with layered encoding is MPEG [8]. MPEG structures video streams in terms of three layers: a coarse or base representation of the signal plus successive enhancement layers. In the case of MPEG-1, 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 or instantiating filters) when resource availability and/or user QoS requirements change. *Scaling* is a term, first proposed in [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) [10].

The remainder of this article is structured as follows. We first present, in the following section, a detailed description of the METS API and some of its key internal mechanisms. Then, in the third section, 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 optimization. We present our conclusions in the final section.

QoS-A Adaptive Transport System

Transport Application Programming Interface

The METS API is realized 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, and jitter. 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 reduce QoS when resources are in short supply (perhaps by frame dropping [11] or shaping filters [12]), or simply that the user be informed of any degradations. A QoS policy may also require that QoS be *raised* when resources become available; for example, by adding an enhancement layer to a hierarchically structured video flow [13, 14].



■ Figure 1. *QoS-A model.*

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 switched virtual circuit, 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:

- *Media sockets*, used for the transfer of continuous media; simplex and QoS-configurable via a flow specification and QoS policy
- *Control sockets*, used for the transfer of application-level control information; full duplex and ensured (i.e., they provide a reliable delivery service)
- *Management sockets*, used to interface with the QoS maintenance and flow management planes

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
- *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; the flow management primitives (Table 1) 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 ongoing flow.

At any point during a session, group members may change the QoS negotiated during the connection establishment

[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 easily be modified to take advantage of underlying QoS support; see the next section for more details of the implementation environment.*

Flow management primitives	Parameters
`registerSoc`	Management socket, media socket

`changeQoS Req`
`changeQoS Ack`
`changeQoS Nak`
	Management socket
	options (flowSpec | QoSPolicy | maint | monitor | signal | adapt | filter | event),
	structure (flowSpec | QoSPolicy | maint | monitor | signal | adapt | filter | event),
	sizeof (flowSpec | QoSPolicy | maint | monitor | signal | adapt | filter | event)
	Status

`signalQoS`	type (signal | event) option (QoSMetric | QoSEvent)

■ Table 1. *Flow management primitives.*

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.[2]
- `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[3] the flow; 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 [16], which permit fully continuous adaptation of bit rates.
- `Filter` is used to explicitly select new filters (e.g., a jitter filter at the receiver or picture dropping and lowpass filters [10] in the network)
- `Event` is used to allow applications to attach alarms to the occurrence of particular event thresholds. Should the threshold be exceeded, 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.

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 subsections.

The transport system comprises 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 Fig. 2, within each of these per-plane modules METS provides a range of QoS mechanisms. Rather than describe the full range of mechanisms (all detailed in [15]), we concentrate here on those mechanisms (highlighted in Fig. 2) in the user plane 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 mechanisms are the main focus of the performance evaluation in the third section.

Transport QoS Mechanisms

Flow Scheduler — The flow scheduler operates in conjunction with the flow shaper (Figs. 2 and 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 (ALFs) [17] on a coarse-grained frames-per-second basis. It is implemented in a user space library and 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.[4]

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 [18] is used which takes into account any missed deadlines. If a deadline has been missed, 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 predefined missed deadline threshold. As described previously, flow management can upcall applications (via the management socket) to inform them of such events.

When layered-encoded flows are being used, QoS-A applications are generally designed to only transmit BL frames when deadlines are 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.

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 asynchronous transfer mode (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.

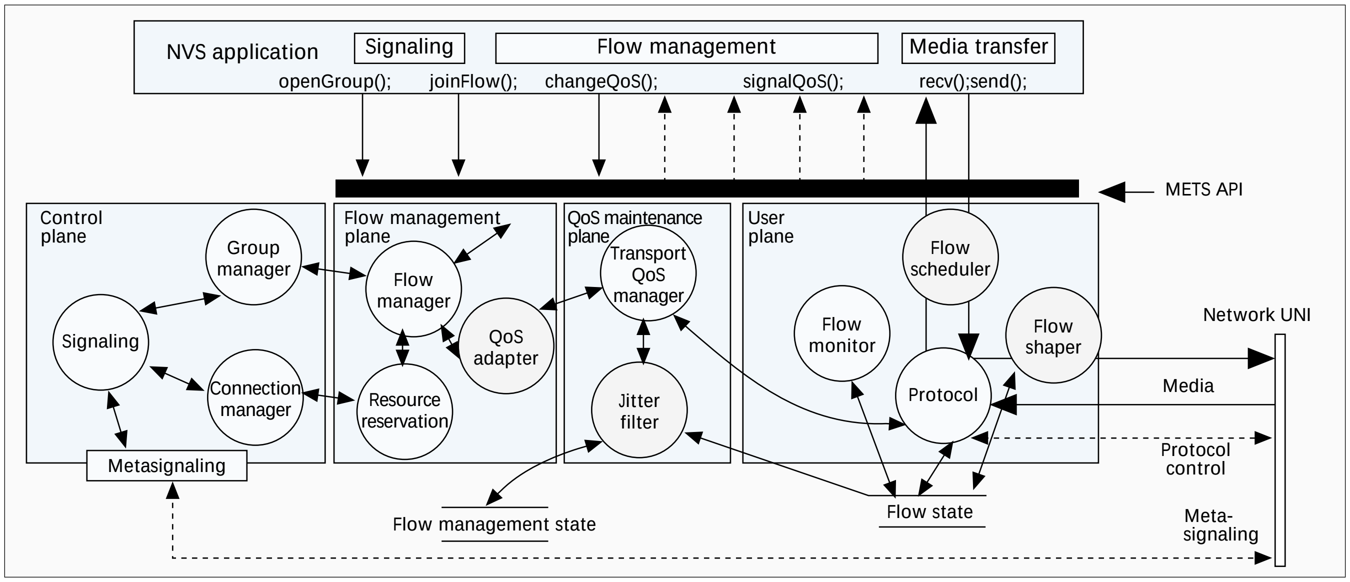
The flow shaper maintains the following per-flow state:

- A *token budget*, b, which represents the capacity or depth of the token bucket
- A *token credit*, r, which represents the remaining credits ($0 < r < b$) left in the token bucket at any point in an interval

[2] *We do not exhaustively specify the various fields in a flow specification or QoS policy in this article; these details are available in [15].*

[3] *This means that it instantiates mechanisms such as those detailed in the next subsection to attempt to deliver the required QoS in the face of QoS fluctuations in the underlying network/end systems.*

[4] *In this role, the jitter filter adjusts the deadline of delivered frames but not the rate.*



■ Figure 2. *METS transport system.*

- A *token refresh rate, p*, which represents the rate at which the token bucket is refilled
- A *token timeout, t*, which represents the number of ticks remaining until the token refresh rate expires and token credits are refreshed

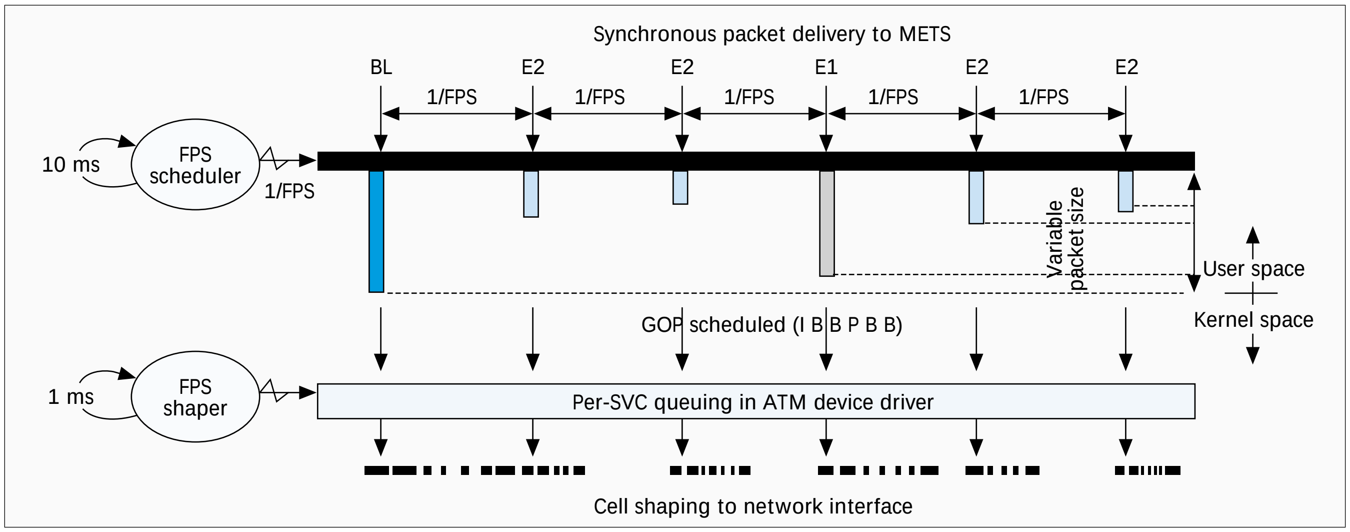
The token bucket scheme operates in a rather simple fashion. When the token timeout expires, the token credit is set equal to the token budget. One credit represents one cell transmission opportunity. When the flow shaper executes, it visits all the per-flow queues in a round-robin fashion and, if the queue contains cells and has available credits, the shaper transmits one cell from the queue and decrements the token credit state variable. This achieves the desired effect of interleaving 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, all the credits have been used during the interval. If no cells remain queued, all queued cells have been dispatched to the network during the interval with ($0 \leq r < b$) credits potentially remaining. At the end of a token refresh cycle, any credits remaining at the end of the interval are lost.

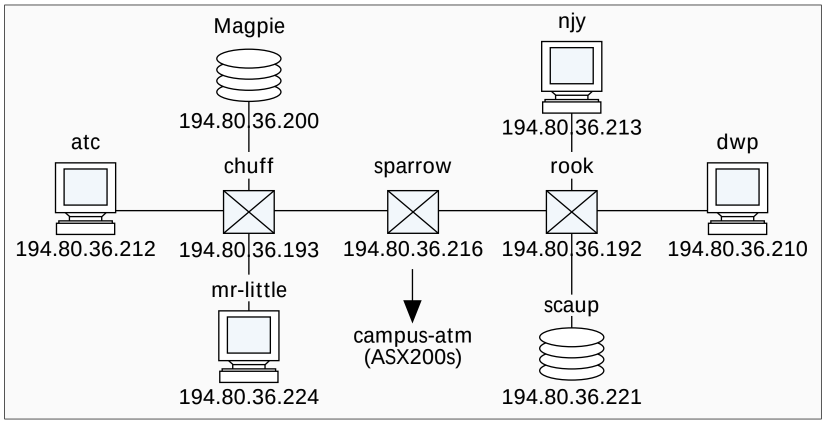
Jitter Filter — The objective of the jitter filter is to restore the timing properties of the originally transmitted data flow at the source before the flow is delivered to the playout device at the sink (i.e., to remove end-system- and network-induced jitter).

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 in [19, 20] and using synchronized clocks [21]) statistical analysis of per-packet delay to estimate the maximum delay; that is, $D = d + r * s$, where d is the average delay (see below), s is the standard deviation (see below), and r is a filter coefficient that is chosen as a function of the form of the distribution curve and the number of failures one is ready to accept [22].

Each failure corresponds to a transmission delay larger than the estimated maximum. Packets that arrive after D are considered too late to help reconstruct the signal; 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 corre-


Synchronous packet delivery to METS
BL
E2
E2
E1
E2
E2
1/FPS
10 ms
FPS scheduler
1/FPS
Variable packet size
User space
Kernel space
GOP scheduled (I B B P B B)
1 ms
FPS shaper
Per-SVC queuing in ATM device driver
Cell shaping to network interface

■ Figure 3. *Flow scheduling and shaping.*



■ Figure 4. *METS testbed.*

sponds to an accepted loss of about 1 percent if one assumes a Gaussian distribution of the delays [22]. From an intuitive point of view the 2*s* 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 [20]. For a full discussion of these issues see [19].

The jitter filter continuously estimates the average delay and standard deviation. It is based on the "lowpass filtering algorithm" used in Transmission Control Protocol (TCP) for the estimation of the acknowledgment delay time [19], 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 timestamp. 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 [3]) is to keep any adjustments as imperceptible to the human visual system as possible. 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.

Network QoS Mechanisms

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 BL is guaranteed, based on end-to-end admission control and resource reservation (for full details see [15]). 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 an 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 an RES signaling 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 Reservation Protocol (RSVP) [23]. In the case where the adaptation protocol determines that congestion[5] 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, 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* [24] of the multicast SVC. 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/s. Because we support the concept of end-to-end adaptivity in the QoS-A, the adaptation protocol at the source also takes into account end-system resource availability. This allows the source-side adaptation protocol to reduce the advertised rate (e.g., 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 the next subsection) 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 the next subsection) 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 Fig. 4. If a filter is 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 downstream 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

[5] *If a switch detects that queues are building beyond a predefined 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 Fig. 3). At the end of an era the QoS adaptation protocol reads the congestion state and resets it to uncongested. See [7] for full details.*

{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, 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 BL reservations are accounted for. Each switch and end system uses 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, that is, 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 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, resources are returned to the pool and the advertised rate appropriately adjusted. For full details of this scheme see [14].

Media Selector — The media selector resides in network switches and works with the QoS adapter to ensure that the appropriate combinations of BL 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` (Fig. 4). 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, 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 ATM adaptation layer type 5 (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).[6] 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 frame type (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 bits 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 the following section for details of the ATM switches used in the implementation.

[6] *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.*

Experimental Evaluation

The Experimental Environment

The METS testbed consists of four 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, NICs, and ATM switches are all experimental devices supplied by Olivetti and Oracle Research Laboratories (see [25]).

The Olivetti NICs deliver ATM cells at 100 Mb/s and interfaces to a standard PC ISA bus. An AAL5 adaptation layer is implemented in software in the Linux device driver [26]. The ATM switches are 4 x 4s and share the same port design as the NICs. The switches have a measured aggregate switching capability of 200,000 cells/s. The switch architecture comprises an ARM processor connected via direct memory access (DMA) channels to the ports. Because 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 [24]. 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.

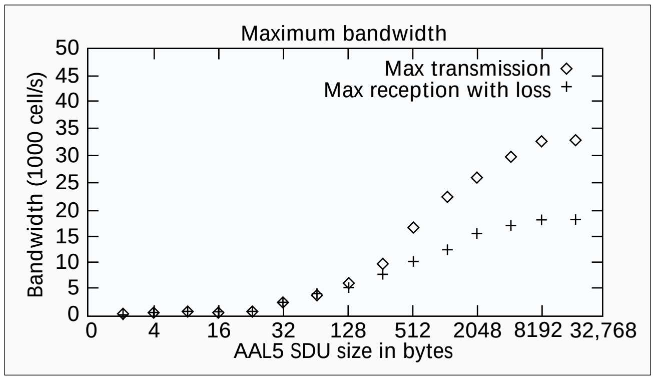
As illustrated in Fig. 4, the testbed includes the following client nodes: the `atc` and `dwp` end systems, which are 90 MHz Pentiums, and the `mr-little` and `njy` end systems, which are 66 MHz 486 machines. The RAID-3 storage servers (the `magpie` and `scaup` end systems) utilize the ARM 610 RISC processor and incorporate five SCSI interface controllers and disk drives.

The Experiments

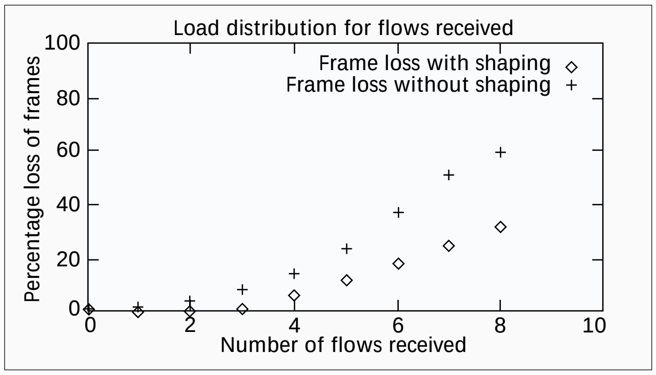
In order to gain experience with the METS testbed (which is based on a native ATM communication stack) and evaluate its performance, we have designed a set of experimental test suites:

- 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 the receiver
- 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 `dwp` end system. In this configuration we utilize the higher-end 90 MHz Pentium machines. 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 three 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 arrival 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



■ Figure 5. *Comparison of maximum transmission and reception rates.*



■ Figure 6. *METS frame loss distribution with and without traffic shaping.*

delay, and jitter of received packets and loss of METS packets at the transport level. While the testbed comprises a limited number of switches and end systems, we found it more than adequate to test the transport system software.

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 traverse the METS transport system, AAL5, and ATM NIC without consideration of the receiver's ability to consume media.

The results are presented in the upper curve of Fig. 5, 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 32,000 cells/s (13.6 Mb/s) compared to the theoretical NIC line rate of over 235,000 cells/s.

This experiment (the lower curve in Fig. 5) 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 on between the transmitter and the receiver. A maximum goodput of 17,500 cells/s (7.4 Mb/s) was measured. This is 54 percent of the maximum transmission rate obtained in the open loop test.

Loss Analysis

The objective of this experiment (Fig. 6) 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 frames/s (fps). Tests were performed both with and without traffic shaping at the source.

The upper curve in Fig. 6 depicts the loss occurring from an unregulated source; 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 percent 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 percent — which is starting to be significant to the end user. Regulated traffic, on the other hand, exhibits 10 percent loss when the number of flows approaches 5.

Delay Analysis

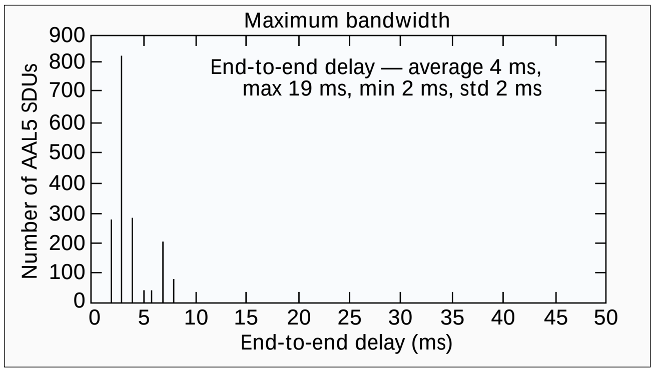
Single-Flow Delay — The configuration of this experiment is a lightly loaded network with one video flow running at 24 fps. The average delay (Fig. 7) measured by the transport protocol at the receiver is 4 ms. The minimum and maximum delays recorded are 2 ms and 19 ms, respectively, with a standard deviation of 2 ms.

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 Fig. 8, there is little difference, just 6 ms, in the *average* delay measured as the number of flows increase. Variation in the *maximum* delay experienced, however, is significantly large at 42 ms, the maximum delay measured being 61 ms This indicates that there are significant spikes in the measured delay, while the average remains close to the unloaded case identified previously.

Jitter Filtering Analysis

This experiment investigates the ability of the jitter filtering mechanism to adaptively adjust the playout delay experienced by flows at the receiver to meet end-to-end delay and jitter requirements. The source packetizes 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 being handled simultaneously by the same receiver.

In Fig. 9, the playout curve tracks the arrival time curve to the first point of loss — the region between 42,500 and 43,000 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 43,000 and 43,500 ms) also represents a large increase in mea-



■ Figure 7. *Per-packet end-to-end delay statistics for one flow.*

sured 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 43,000 and 43,500 ms due to underestimation of the maximum end-to-end delay.

QoS Adaptation Analysis

The final experiment demonstrates the ability of the QoS adaptation algorithm to switch new enhancements in and out at the receiver as the available bandwidth fluctuates. In the experiment, the network provides "hard" guarantees to the BL of a multilayer flow but gives no such guarantees to the enhancement layers, E1 and E2, which must compete for residual bandwidth with all other flows, as discussed earlier.

In the experiment (Fig. 10) the receiver selects three flows for playout in the first instance. These are:

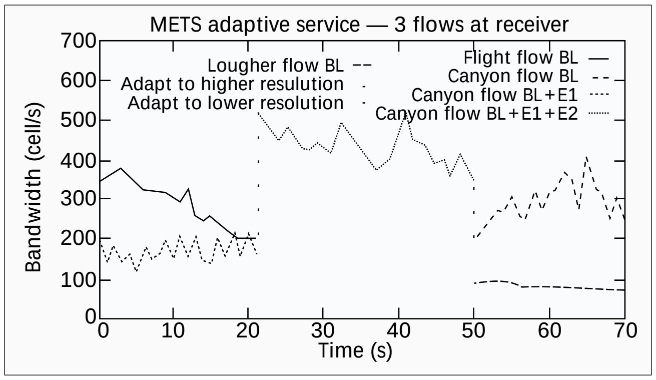
• A "Canyon.mpg" video flow (selecting the BL, E1, and E2 components) at 24 fps
• A "video flow" (selecting the BL only) at 5 fps
• A "Flight.mpg" video flow (selecting the BL only) at 5 fps

The scenario shows the consumption of the Canyon and Flight video clips beginning at time zero. Both BLs are supported. The QoS adapter determines that none of the Canyon flow's higher resolutions can be accommodated given the available resources. 20 s 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.

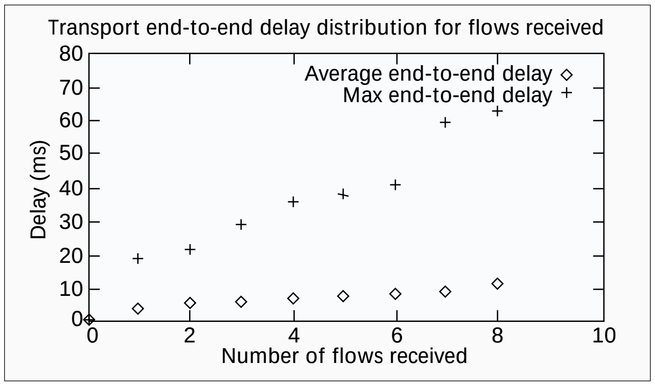
This situation remains until the user chooses to display the Lougher video 50 s into the trace. Resources are thus allocated to meet the BL QoS requirements of the new video. The QoS adapter protocol sends an RES message toward the core requesting resources to meet the highest resolution (E2) of the 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).

Discussion of Results

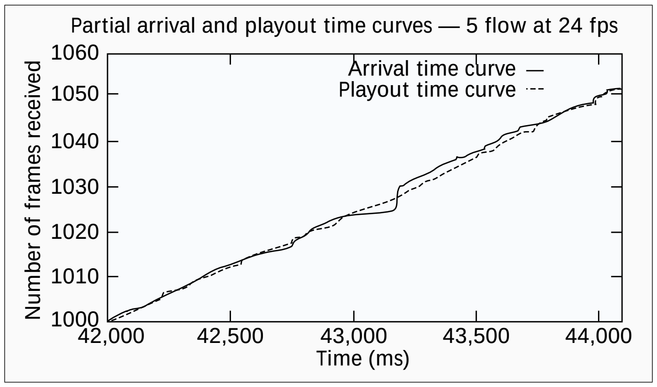
The results of the first three experiments show the raw performance limitations of the experimental implementation. Regarding the bandwidth analysis, the bottleneck appears to be a combination of the limitations imposed by the ISA system bus and the experimental prototype NIC. For example, the host central processing unit (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.[7] Although an obvious solution to these problems would be to employ an



■ Figure 10. *QoS adaptation scenario.*



■ Figure 8. *End-to-end delay statistics.*



■ Figure 9. *Arrival and playout time distribution high load.*

NIC that offered AAL processing and DMA data copying onboard, the experimental NIC is very valuable to us for experimentation because we have the flexibility of host CPU control of functions such as flow scheduling and shaping.

As presented in Fig. 5, receivers in our testbed can accommodate a maximum transmission rate of up to 8000 cells/s, 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 Fig. 5.

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 percent. This rose to 33 percent loss for eight flows consumed. Corresponding performance for the unregulated case was measured to be 15 percent and 60 percent for four and eight flows consumed, respectively.

The video sequences selected [*viz*., canyon.mpg, lougher.mpg,flight.mpg] for the experiments represents a selection of high-motion bursty traffic and talking-head-type scenarios.

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 measured. The average delay difference experienced between one flow and eight flows was found to be 6 ms, showing that the average delay did not increase significantly as the number of consumed flows increased. This

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

was not so for the maximum delay measured during the two experiments. As the number of flows increased from 1 to 8, the maximum delay measured increased from 19 to 61 ms. Since the jitter filter works by estimating the maximum delay, the latter results are significant in adjusting the playout time and buffering requirements of continuous media.

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 Fig. 9, is difficult to deal with. A solution is to increase the value of the "confidence factor" 2*s*; 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 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 [19, 20]

Regarding the QoS adaptation experiments, it was demonstrated that the distributed QoS adaptor scheme successfully maximizes the utilization 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 [12]. The integration of dynamic rate shaping into the current testbed is an issue for future work.

Conclusion

This article 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 [27], 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 delegating all responsibility for dynamic QoS management to the underlying system, to simply exploiting API upcalls to perform all dynamic QoS management for themselves (cf. `vat`, `vic`, etc.). In all cases, applications choose their preferred strategy by appropriately initializing a QoS policy.

While 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 through the use of sensible default values for QoS parameters and policies.

Future Work

Many adaptive algorithms reported in this article are applicable to wireless and mobile networking environments due to the existence of large-scale mobility requirements, limited radio resources, and fluctuating network conditions. As part of a new research initiative in wireless media systems research [28] we are developing a QoS-aware middleware platform called *mobiware* [29] which comprises a number of QoS adaptive algorithms. The mobiware adaptive algorithms include QoS-controlled handoff [16] and an active transport system [30] designed to operate over wireline/wireless ATM networks. QoS-controlled handoff uses the notion of an adaptive network service to provide hard guarantees to base layers and soft guarantees to enhancement layers as mobile devices roam. The active transport system uses the notion of mobile transport objects [31] which can be dispatched on demand to strategic points in the network (e.g., base stations) to provide value-added QoS support when and where needed.

References

[1] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A Survey of Quality of Service Architectures," *Multimedia Sys. J.*, 1997.

[2] D. Cohen, "Issues in Transit Packetized Voice Communication," *Proc. 5th Data Commun. Symp.*, Snowbird, UT.

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

[4] T. Turletti, "A H.261 Software Codec for Video-Conferencing over the Internet," INRIA Tech. Rep. 1834, France.

[5] S. McCanne and V. Jacobson, "VIC: Video Conference," U.C. Berkeley and Lawrence Berkeley Laboratory; software available via ftp://ftp.ee.lbl.gov/conferencing/vic

[6] A. T. Campbell *et al.*, "Integrated Quality of Service for Multimedia Communications," *Proc. IEEE Infocom '93*, Hotel Nikko, San Francisco, CA, Mar. 1993.

[7] A. T. Campbell, G. Coulson, and D. Hutchison, "A Quality of Service Architecture," *ACM Comp. Commun. Rev.*, Apr. 1994.

[8] ISO H.262, "Information Technology — Generic Coding of Moving Pictures and Associated Audio," Committee Draft, ISO/IEC 13818-2, U.K., Mar. 1994.

[9] L. Delgrossi *et al.*, "Media Scaling for Audio-Visual Communication with the Heidelberg Transport System," *Proc. ACM Multimedia '93*, Anaheim, CA, 1993.

[10] N. Yeadon *et al.*, "QoS Adaptation and Flow Filtering in ATM Networks," *Proc. 2nd Int'l. Wksp. Adv. Teleservices and High Speed Commun. Architectures*, Heidelberg, Germany.

[11] N. Yeadon, "QoS Filtering," Ph.D. thesis, Department of Computing, Lancaster Univ., U.K., 1996.

[12] A. Eleftheriadis, and D. Anastassiou, "Meeting Arbitrary QoS Constraints Using Dynamic Rate Shaping of Code Digital Video," *Proc. 5th Int'l. Wksp. Network and Op. Sys. Support for Digital Audio and Video*, Durham, New Hampshire, 1995.

[13] N. Shacham, "Multipoint Communication by Hierarchically Encoded Data," *Proc. IEEE INFOCOM '92*, Florence, Italy, vol. 3, 1992, pp. 2107–14.

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

[15] A. T. Campbell, "A Quality of Service Architecture," Ph.D. thesis, Depart-

ment of Computing, Lancaster Univ., UK, 1996, http://comet.ctr.columbia.edu/~campbell

[16] A. T. Campbell, "End-to-End Programmability for QoS Controlled Mobility in ATM Networks and their Wireless Extension," *Proc. 3rd Int'l. Wksp. Mobile Multimedia Commun.*, Princeton, NJ, Sept. 1996.

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

[18] S. Simpson, "A Dynamic Rate Shaping Mechanism," Internal rep., Dept. of Computing, Lancaster Univ., UK, 1995.

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

[20] R. Ramjee *et al.*, "Adaptive Playout Mechanisms for Packetized Audio Applications in the Wide-Area Network," *Proc. IEEE Infocom '93*, 1993.

[21] D. Mills, "Network Time Protocol (Version 3): Specification, Implementation and Analysis," IETF RFC, No. 1305, Networking Info. Ctr., SRI Int'l., Menlo Park, CA, Mar. 1992.

[22] C. Huitema, *Routing in the Internet*, Upper Saddle River, NJ: Prentice Hall, 1994.

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

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

[25] L. J. French, I. D. Wilson, and D. P. Gilmurry, "ATMOS II User Manual," Olivetti Research Ltd., Cambridge, U.K., 1994.

[26] A. S. Lunn, "A Mini Cell Architecture for Multimedia Systems," Ph.D. thesis, Dept. of Computing, Lancaster Univ., U.K., Sept. 1995.

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

[28] Wireless Media Systems Project, COMET Group, Center for Telecommun. Res., Columbia Univ.; see http://comet.ctr.columbia.edu/wireless

[29] A. T. Campbell, "Mobiware: QoS-Aware Middleware for Mobile Multimedia Networking," *Proc. 7th Int'l. Conf. High Performance Networking*, White Plains, NY, Apr. 1997.

[30] A. T. Campbell, and M. E. Kounavis, "An Active Transport for Mobile Multimedia Networking," CTR Tech. Rep., Jan. 1997.

[31] A. T. Campbell and A. Balachandran, "Mobile Filters: Delivering Scaled Media to Mobile Devices," CTR Tech. Rep., Jan., 1997.

Biographies

ANDREW T. CAMPBELL (comet.ctr.columbia/~campbell) joined the electrical engineering faculty at Columbia as an assistant professor in January 1996 from Lancaster University, where he conducted research in multimedia communications as a British Telecom Research Lecturer. Before joining Lancaster University, Dr. Campbell worked for 10 years in industry, focusing on the design and development of network operating systems, communication protocols for packet-switched and local area networks, and tactical wireless communication systems. Dr. Campbell is a member of the COMET Group at Columbia's Center for Telecommunications Research, where he is conducting research in wireless media systems (comet.ctr.columbia.edu/wireless). His current research interest includes the development of QoS-aware middleware for mobile multimedia networking. He is currently a co-chair of the IFIP Fifth International Workshop on Quality of Service (IWQoS '97).

GEOFF COULSON received a first class honours degree in computer science and Ph.D. in the area of systems support for multimedia applications from the University of Lancaster, United Kingdom. He is currently a lecturer at Lancaster and is managing a number of research projects. These include the SUMO project, which is investigating operating system support for continuous media communications and application support, and WAND, a European Commission-funded collaborative project which is developing a mobile ATM demonstrator. His research interests are distributed systems architectures, multimedia communications, and operating system support for continuous media. He has organized and served on the program committees of numerous conferences and workshops in his area.