Multimedia Tools and Applications, 13, 285–306, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

QoS Filtering and Resource Reservation in an Internet Environment

ANDREAS MAUTHE andreas.mauthe@gmx.de
Tecmath AG, Kaiserslautern, Germany

FRANCISCO GARCIA frankie_garcia@agilent.com
Agilent Laboratories, Edinburgh, UK

DAVID HUTCHISON dh@comp.lancs.ac.uk
Computing Department, Lancaster University, Lancaster, UK

NICHOLAS YEADON nick.yeadon@rd.bbc.co.uk
BBC Research & Development, Tadworth, UK

Abstract. Multimedia group applications often operate in an environment where the various participants are located on systems and communication links with different capabilities. Mechanisms are required that ensure full-quality media for high-performance workstations but lower-quality media for playout at low-end systems. QoS filters have been proposed as a way to adapt QoS to the user specified level by changing the structure of a media stream in a well defined way. Resource reservation and QoS filter instantiation should be closely integrated since both represent one particular aspect of the provision of individualistic QoS for heterogeneous users in multipeer communications. The Internet reservation protocol RSVP is receiver oriented and allows each receiver to specify its resource requirements. However, no actual mechanisms are defined that adapt the data stream to the receiver specified QoS requirements. In this paper we present an enhanced version of RSVP (called *RSVP++*) that integrates resource reservation and QoS filter control. In order to achieve this integration we extend the RSVP functional model and define a new QoS service class. RSVP++ can coexist with common RSVP systems, thus, openness and interoperability of the system are ensured.

Keywords: QoS filtering, media stream adaptation, resource reservation, RSVP, heterogeneous multipeer communications

1. Introduction

Today's computer and network infrastructures are often characterized by heterogeneity in hardware and software. This is totally acceptable in isolated systems; as such systems are often tailored to the particular task. Once these systems are connected together, the problems of establishing a true open environment emerge. This is especially critical for multimedia group applications where different participants are located on systems with different processing capabilities connected via communication links ranging from high speed networks to mobile connections with relative low bandwidth. Such systems are for instance training applications for field engineers or remote consultancy applications in the area of tele-medicine. All participants in such applications should be provided with the best possible quality, i.e., those with high capability systems should have full quality video and audio whereas others who have only limited capabilities should have the media quality that is best suited to the specific task at hand. In which way the media quality is reduced depends on the specific requirements of the application. For some applications the continuity of a movement is more important than color information or the sharpness of an image (e.g., a video that shows a specific action). At other times it is more important to have a clear, full colored picture rather than the full frame rate (e.g., where the structure of a slow moving object is viewed). Both might actually be the case at different times in the same session (i.e., first a movement is demonstrated and later the structural characteristics are analyzed in more detail). Low bandwidth links in this context do not imply, however, that no resources are reserved and only the quality is adapted to the available (and possibly changing) bandwidth. On the contrary, the users of multimedia group systems such as CSCW applications might be able to cope with a relatively low quality as long as the behavior of it is deterministic. Hence, the requirement of such applications is to adapt the media quality to the requirements and capabilities of different participants *and* to reserve resources to ensure deterministic behavior.

The proliferation of the Internet Protocol (IP) has gone some way to solve the interconnection problems for data transfer. However, the issues relating to the transfer of interactive continuous media are still not resolved. In the Internet the problem of QoS provision is mainly addressed by the resource reservation protocol RSVP [1, 16]. RSVP was designed with multipeer communication in mind. Reservations in RSVP are initiated by the receiver who (knowing his own resource capacities and experiencing the quality of incoming traffic) can define which part of the transmitted data he wants to receive with traffic performance guarantees [16]. However, a sensible way to discard or alter those parts of the data stream that can not be handled appropriately by the receiver(s) is lacking. Thus, data for which not enough resources were reserved might be randomly discarded in the network. Hence, those receivers who have reserved fewer resource than required by the media stream can receive corrupted, completely distorted and unusable data.

Layered coding is one way to adapt the quality of a media stream. However, in the discussed context this is not always suitable since it only allows to reduce the quality of media in a specific way. The QoS filters[1] developed at Lancaster University [13] can provide individualistic QoS and qualities to different receivers and hence are better suited for applications with heterogeneous requirements. These QoS filters can be dynamically placed in end-systems and network nodes and are controlled via a special QoS filter control protocol.

Although the concept of generic mixers and translators (similar to the QoS filter concept) that alter the composition of a data stream exists as part of the Internet protocol RTP (real-time transport protocol) [6], their description appears more vague than generic. RTP does not specify any particular mixer or translator, neither does it define how they can be instantiated and controlled. With the Lancaster QoS filters the structure of a continuous media data stream can be changed and by doing so the QoS requirements are adapted. Ideally resource reservation and QoS filter instantiation are closely integrated since both represent one particular aspect of the provision of individualistic QoS for heterogeneous users in multipeer communications. In this paper we present an enhanced version of RSVP (called

RSVP++) that integrates resource reservation and QoS filter control. In order to achieve this integration we extend the RSVP functional model and define a new QoS service class. RSVP++ can coexist with common RSVP systems, thus, openness and interoperability of the system are ensured.

The paper is structured in six sections. Section two briefly describes the QoS filters developed at Lancaster and presents some experimental results. Subsequently, in section three, QoS provisions in the Internet are discussed and the relevant characteristics of RSVP are outlined. In section four the RSVP++ functional model and the new QoS service class are introduced. Further, the way QoS filters and resource reservation are integrated is described in detail in this section. Section five discusses implementation related issues. Finally, section six gives future directions and conclusions.

2. QoS filters: Addressing the heterogeneity gap

The work on QoS filtering at Lancaster was primarily motivated by the need to support multipeer communication within heterogeneous communication environments. In this section we introduce the QoS filters developed at Lancaster and show how they reduce the resource requirements of a data stream in a structured manner. Since QoS filters can be dynamically placed in the dissemination tree they allow to provide individualist QoS to all participants in a group session.

2.1. QoS filters: General priniciples

In order to provide individual QoS levels to different receivers in multipeer communication while still exploiting the advantages of multicast, the use of QoS filters has been proposed [4, 12, 14]. QoS filters are objects that transform continuous media streams in some way. This may for instance involve the reduction of video frame rate, adjustments to presentation quality or conversion to different compression formats. The effect QoS filters have is a structural change of the data stream while preserving the information content. This is usually accompanied by a reduction of the QoS requirements of a data stream. A QoS filter may be a software only object or enjoy hardware support. The filtering model involves placing QoS filters at strategic points (such as network nodes, gateways, specialised servers, etc.) around a multicast dissemination tree. The designated source may then send at the quality required by the highest capability receiver while low capability receivers acquire a filtered down version of the media stream.

As the characteristics of the underlying network or the nature of the transmitted media change, QoS filters may be added or removed from the dissemination tree dynamically. Filter objects may also logically *move* around the current tree to achieve the optimum location of execution. This is known as filter *propagation* [4, 13].

By implementing this approach all receivers' disparate quality requirements are satisfied while the advantages of multicast transmission protocols can still be exploited. At present filtering seems to be the only realistic solution for heterogeneous QoS within multicast communications.

2.2. QoS filter types

The QoS filter mechanisms currently implemented at Lancaster broadly fall into the 6 categories listed below [11, 14]. Some of these have been designed with wireless communications in mind. They include filters that produce large reductions in data rate, such as the frame dropping filter, and filters that can improve bit-streams error resilience, namely the slicing filter. The QoS filters allow to adapt a media stream in various ways to the available QoS and the requirements of the user. Hence, the filtered data stream is an optimal compromise between resource restrictions and user requirements.

Codec filter. A codec filter can be used to compress or decompress a bit-stream. It is more commonly used to perform transcoding between different compression standards. Depending on the compression scheme used, transcoding can often be performed without the need for full decompression and re-compression.

Frame-dropping filter. This is a media-discarding filter used to reduce frame rates. The filter has knowledge of the frame types (e.g., Intra- or Inter- frame coding type) and drops frames according to importance. For example, the pecking order for dropping frames from an MPEG 1 video stream is B-, P-, and finally I-pictures. The frame-dropping-filter is used to reduce the data rate of a stream in a sensible way by discarding a number of frames and transmitting the remaining frames at a slower rate. The filter may be used to ensure that a receiver gets frames at a rate suitable to its processing capabilities or because it can only decode one type of picture (e.g., I-pictures). This filter operation can also be employed to save network resources by discarding frames that are late or have been corrupted by loss of constituent packets.

Frequency-filter. A frequency filter performs operations on semi-uncompressed data. That is, it operates in the frequency domain on the values of the DCT-coefficients. Semi-decompression/compression involves just entropy decoding/encoding. Filter mechanisms include low-pass filtering, color reduction filtering and color to monochrome filtering. Low-pass filtering is where the higher frequency DCT-coefficients are discarded on recoding, leaving only the DC DCT-coefficient and a number of low-frequency components. Color reduction filtering performs the same operation as the low-pass filter but only operates on the chrominance information in the bit-stream. The color-to-monochrome filter removes all color information from a bit-stream. A color to DC-color filter performs the same operation as the color to monochrome filter except that the DC DCT-coefficient is left untouched, giving coarse blocks of color. Unlike the frame-dropping filter the frequency filter, and later on the re-quantization filter, reduces the required bandwidth without affecting frame rate. Of course this is at the cost of image quality.

Mixing filter. This filter is used to mix streams together or to multiplex audio and video where the encoding supports this kind of structure (e.g., MPEG). Mixing can be performed in a couple of ways, either on a frame basis or a slice basis. Frame based mixing simply interleaves two streams into the same stream. This requires a more complex demultiplexing or decompression process in the end-system. Slice-mixing filtering involves more processing by the mixer but can be decoded as a single stream. The slice-mixing merges two frames from different sources into the one larger frame by adding slices from each stream into the target stream, see figure 1.

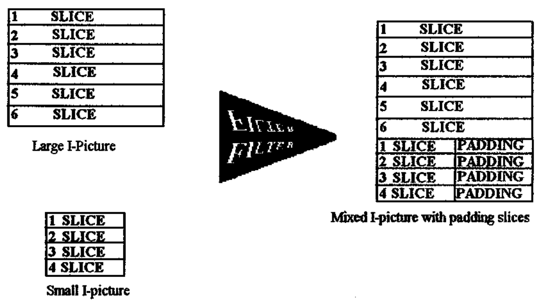


Figure 1. Slice based mixing.

The slice structure of the frames sometimes needs to be altered so that each row of macro-blocks is contained in one slice (see slicing filter below). Extra blank padding slices are needed if the two streams are of different window sizes; to make the resultant image rectangular. Mixing two streams in this way means that the receiving end-system needs only one decompression board or one decoding process. Also there is a reduction in transmission overhead as one and not two streams will be transferred.

Re-quantization filter. The re-quantization filter, like the frequency filter, operates on the DCT-coefficients but also dequantizes the coefficients. The coefficients are then re-quantized using a larger quantizer step. As quantization is the most lossy process in the DCT based compression algorithms, requantization can produce some strange edge effects. However, the bit-rate reduction achieved by this filter can be quite substantial.

Slicing filter. The slicing filter increases the number of slices in an MPEG stream, or restart segments in Motion-JPEG, per frame. Slices are identifiable from byte aligned slice headers. Slices are used to provide protection from errors occurring within a picture. The variable length decoder (entropy decoder) is realigned on each slice header. Therefore, if the bit-stream is corrupted and the decoder becomes misaligned with the variable length codes in the bit-stream, erroneous effects will not carried over past the end of the slice. The penalty for adding a slice header consists of a 32 bit byte aligned header code, followed by a 5 bit quantizer scale and a 1 bit extra information bit, i.e., 38 bits plus up to 7 padding bits to make the start code byte-aligned.

2.3. *QoS filter instantiation and control*

The Lancaster QoS filters currently operate independently of any resource reservation scheme. QoS filters are instantiated is shown in figure 2. The client application is constructed around the MPEG 1 software video player available as shareware from the University of California, Berkeley [5]. The software has been modified to receive data from the network and take as input the host name of the required file server (or the next up-stream filter server).

The client application also permits a *QoS-event* and *QoS-action* to be specified. The QoS-event identifies an incident which constitutes the breaching of a pre-defined *threshold*

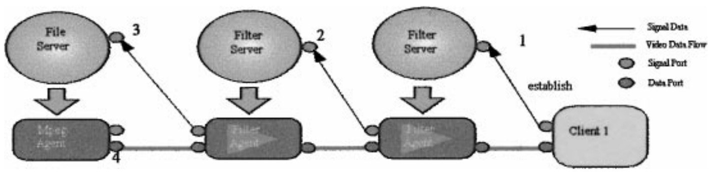


Figure 2. Filter control: Establishment phase.

level. When the QoS-event occurs the specified QoS-action is undertaken. The prototype implementation allows the event to be specified in terms of loss percentage (i.e., packets lost) and the action equates to the automatic instantiation of a filter operation.

The filter object consists of two software elements: a daemon and a filter agent. The filter server daemon waits for requests from down-stream nodes, which are either receivers or other filtering nodes. The filter server spawns a filter agent with the necessary filter operation and parameters. The parameters the daemon receives and passes to the filter agent include the requested service of the down-stream client.

The filter agent instantiates the requested filter operation and forwards the service request up-stream. The next up-stream node may be another filter server or the source file server. The filter agent then waits for the data stream to propagate from the source down through the dissemination tree. The filter agent performs the necessary filter operation on each packet as it is received. Each network node need only know the location and signaling port address of the next up-stream node. This approach allows connections to be quickly set-up and tested. Multipeer communications can also be quickly established by down-stream nodes joining existing streams at the filter agents. The filter agent is then responsible for generating the number of output streams required from the single input stream. Each received packet is filtered and copied to the output buffers of the filter agent.

The prototype filter propagation protocol is based on a simple scheme; at regular intervals (in order to avoid oscillation) the filter agent evaluates the filtering functions being performed on each of its output buffers. If possible the filter operations are combined and a filter request message is sent to the next up-stream node. This changes the operations on the output of the up-stream filter agent.

2.4. *QoS filter performance*

It was shown in [14] that not only can filter operations be performs on the fly but also that the end-to-end delay may actually be reduced by filter operations. This is because the more data a filter discards the less it has to process for re-encoding and hence the faster it operates. This also has implications on the decoding process. As data is discarded at the filter, there is less for the client system decoder to process. Therefore, the delay incurred by placing a filter in the dissemination tree and filtering the data stream is offset by the gain in reduced delay at the software decoder. In figure 3 it can be seen that where the low-pass filter is used with a low cut-off point (CO = 1 and CO = 2) the actual end-to-end delay can be reduced.

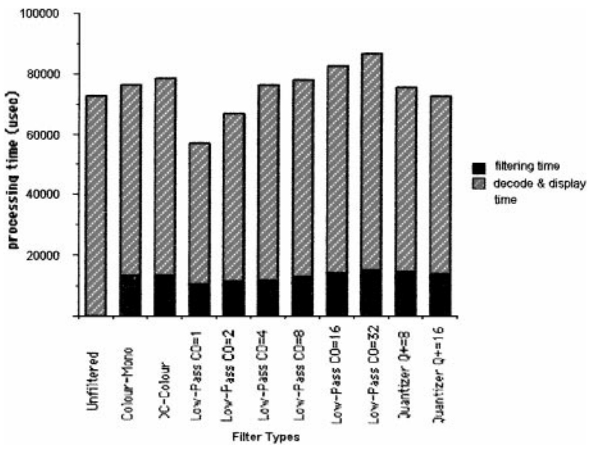


Figure 3. Mean processing time to filter and display an I-frame.

3. QoS in the internet

The Internet is an accumulation of a vast number of heterogeneous networks ranging from multi-gigabit fibre optic based backbones to 10 Mb/s ethernets. The provision of QoS in such an environment is a complex issue, especially if multipeer communication with heterogeneous receivers is considered. QoS in the IPng architecture is supported by the concept of flow and traffic classes in IPv6 in conjunction with the resource reservation protocol RSVP and resource management in network nodes. RSVP is used to convey information about the requested QoS between end-systems and network nodes. The traffic specification itself is opaque data to RSVP [1] and is defined separately.

3.1. Resource reservation with RSVP

RSVP is a companion protocol to IP [15]. Its task is twofold, it is used by the end-system to request a specific QoS from the network for certain data streams (so called *flows*), it is further used to propagate QoS and control requests to all routers along the path(s) of the flow(s) and to establish and maintain state information to provide the requested service [1]. RSVP does, however, not guarantee that the requested QoS will be provided or even that resources will be reserved. Further, RSVP does not perform any routing, it only uses local routing databases to establish the route of a flow. Hence, it can operate with different routing protocols.

Reservations in RSVP can be requested for simplex flows. A sender periodically issues *Path* messages which are transmitted hop-by-hop along the data path to the receiver(s). It contains a *Sender Template* (which describes the format of data packets a sender will transmit) and *Sender Tspec* (which describes the traffic characteristics of the sender data stream). Information about available resources along the path(s) is collected while traversing the network and stored in the so-called *Adspec*. While traveling through intermediate systems, *Path* messages set up information for a *reverse route* in the routers. This information is used by *Resv* messages to propagate the receivers' reservation request along the data path towards the sender. The requested QoS is specified in the *Flow Descriptor*. Only *Resv* messages can instantiate a reservation. Hence, it is the receiver who defines how much resources should be reserved for a particular flow. Since there can be a potentially large number of receivers there might be multiple reservation requests for the same flow(s)[2]. Whenever there is more than one such reservation request at a network node, RSVP merges the different requests, i.e., a reservation for the largest flowspec is made and one *Resv* message with this flowspec is forwarded to the previous hop. RSVP paths can also be actively released by means of teardown messages (i.e., *PathTear* and *ResvTear*).

The actual reservations depends on the ability of each router between the sender and the receiver(s) to provide traffic control. The components in a router required to perform traffic control are a *packet classifier*, a *packet scheduler,* and *admission control*. For each flow admission control is performed locally at each router based on the current resource availability.

RSVP filters determine the use of resource reservations[3]. The *filter spec*, which is part of the *Flow Descriptor*, specifies the subset of packets which should benefit from a reservation. These packets can originate at a single sender and belong to one flow only but they can also be sent by multiple senders who are part of the same session. Three types of filters are defined. These are *Wildcard Filters* (a single resource reservation is shared by all flows from all upstream senders of one session), *Shared-Explicit Filters* (the resource is shared among streams from a well defined set of senders) and *Fixed Filters* (the reservation is for packets from one sender only).

3.2. QoS control services in the internet

Three QoS service types are proposed for the Internet, the traditional *best-effort* service, the *controlled-load* service and the *guaranteed* service. The controlled-load service closely approximates the behavior of the best-effort service under unloaded conditions. This service was designed for applications highly sensitive of overload conditions such as adaptive real-time applications. Delay or loss parameters are not specified for this service. The guaranteed service gives bandwidth and delay "guarantees", i.e., it ensures that datagrams arrive within a guaranteed delivery time and will not be discarded in case of congestion.

Two specifications are used to describe QoS for a controlled network service. The Traffic Specification (*TSpec*) describes the traffic pattern for which the service is being requested, and the Service Request Specification (*RSpec*) specifies the QoS a flow wishes to request [9]. Table 1 summarises the set of general control and characterization parameters which should be employed by all QoS control services [8].

Both the controlled-load and guaranteed service uses the TOKEN_BUCKET_TSPEC to describe the traffic characteristics of a flow [7, 10]. For the guaranteed service the *RSpec* specifies a rate and a slack term. The maximum end-to-end queuing delay is given by the parameters C_{tot} (total delay a datagram might experience due to rate parameters of the flow) and D_{tot} (total rate-independent worst case transmit time variation). The guaranteed service is not concerned about delay variation, i.e., it does not minimize jitter [7].

Table 1. General control parameters for QoS service types.

Parameters	Description	Value
AVAILABLE_PATH_BANDWIDTH	Available bandwidth along the data path	*bytes/sec*
MIMIMUM_PATH_LATENCY	Minimal latency of the packet forwarding process associated with a network element	*microseconds*
TOKEN_BUCKET_TSPEC	Describes traffic shape employing a token bucket filter	-token rate r (bytes per packet/sec) -bucket depth b (*bytes*) -peak rate p (*bytes per packet/sec*) -maximum packet size M (*bytes*) -minimum policed unit m (*bytes*)
PATH_MTU	Maximum transmission unit	*bytes*
NUMBER_OF_IS_HOPS	Number of integrated service aware hops	*integer*
NON_IS_HOP	Indicates the presence of network elements not capable of implementing QoS control network services	*bit flag*

4. QoS filters in the internet

Receiver selected QoS is only sensible if individual reservations per receiver are accompanied by a mechanism to reduce the resource requirements of a data stream such as QoS filtering. In order to integrate resource reservation and QoS filtering we have extended the current definition of RSVP. In the enhanced version, RSVP++, the concept of QoS filtering is incorporated in the functional model and a new QoS service class is defined. Filters are controlled using a new object type in RSVP messages as described in detail below.

4.1. The extended functional model

The functional model of RSVP contains five basic components responsible for the provision of QoS. All RSVP messages are handled by the *RSVP Process*. Every time there is a reservation request for a new data flow the *Admission Control* checks if this request can be accommodated and the *Policy Control* verifies the right of the user to make reservations. Incoming data packets are passed in the *Packet Classifier*. This module determines the route and QoS of the packet. Subsequently the packet is queued by the *Packet Scheduler* as indicated by the traffic shape and forwarded to the next node. Additionally, the packet scheduler is responsible for resource allocation on the link-layer medium. If a QoS request can not be supported, RSVP returns an error packet to the originator of the request. Otherwise, Admission Control accepts the request, sets parameters to the Packet Classifier and Scheduler to obtain the desired QoS and forwards the request to the next node. Admission

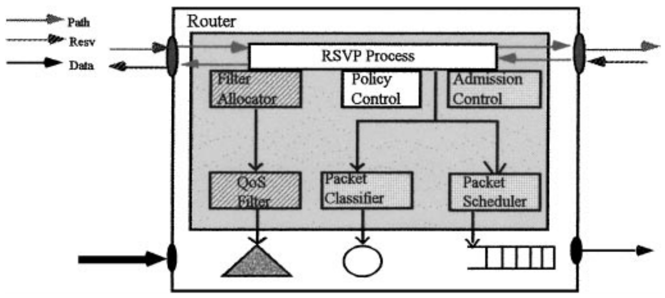


Figure 4. RSVP functional model.

Control, Packet Classifier and Packet Scheduler constitute the *Traffic Control*. Figure 4 illustrates the RSVP functional model.

In [2] we introduced a flow management model with comparable components and a similar structure. The main difference is a component called *Filter Allocator*. This module is responsible for the instantiation and control of QoS filters. QoS filters operate on entire (i.e., unfragmented) application data units such as MPEG frames. They are located at the same system level as RTP mixers or translators [6]. In order to enable QoS filter operations in an RSVP/Internet environment we extend the RSVP functional model by including a *Filter Allocator* and *QoS Filter* module. The Filter Allocator is controlled by the RSVP process that passes QoS filter requests (which are part of the extended flowspec) to the Filter Allocator. The Filter Allocator then instantiates the respective QoS filter by spawning a *Filter Agent* responsible for all QoS filter operations on one flow. The data flow is uniquely identified by the flowID[4] in each message. In the extended model this identifier is used to determine if and what kind of QoS filter operations are to be performed. After filtering the data is passed to the Packet Classifier that treats it according to the reservation for the respective flow. If filtering is not required, no QoS filters are instantiated and packets are directly passed to the packet classifier. The Filter Allocator and QoS Filter component build the *QoS Adaptation* module. In the case where no QoS adaptation module is present in an intermediate node (i.e., QoS filter operations can not be performed), a QoS filtering request is simply ignored by the RSVP Process. However, resources are still reserved at this node and the reservation request (encapsulating the QoS filter control details) is passed on to the next up-stream node. Figure 5 shows the extended functional model for RSVP++.

4.2. The QoS adaptation service

In addition to the currently specified best-effort, controlled-load and guaranteed service, we propose a *QoS adaptation service* that adapts the structure of a media stream to available resources and user requirements in a sensible manner and reserves the required resources on intermediate links. In order to adapt data streams, QoS filters are activated in network nodes during the reservation process. The QoS adaptation service also uses the TOKEN_BUCKET_TSPEC to describe the characteristics of a flow. In addition, a parameter

Table 2. QoS adaptation data types.

Media type	Data types
Video	MPEG-1, MPEG-2, MPEG-4 JPEG, H.261, H.263
Audio	CD_STEREO, CD_MONO, PCM, ADPCM, GSM, CELP

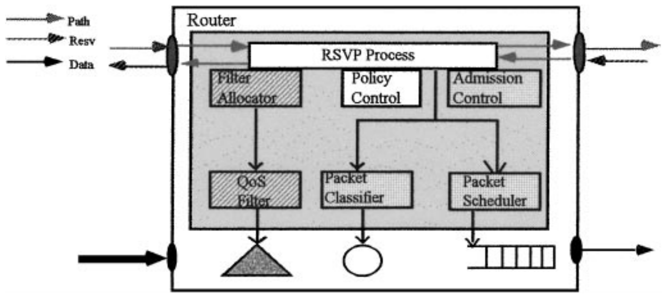


Figure 5. Extended RSVP++ functional model.

DATA_TYPE is used to specify the media type transmitted. Table 2 gives the audio and video data types currently considered.

Further, the *QoSadspec*, a structure that provides information about QoS filters, is employed to control QoS filters. This structure is used in the *Adspec* to provide information on the kind of filters available in intermediate nodes on the path from the sender to the receiver(s). In the *Sender Tspec* it indicates which kind of filter operations the sender allows, and in the *Flow Spec* it is used to instantiate and control QoS filters. The list of filters is given in Table 3.[5]

The currently implemented QoS filters can be controlled using one parameter only. A number of QoS filters can operate in parallel on a data stream to adapt its QoS to user requirements and resource availability. Users are not expected to specify all the different QoS filters and the parameters that are required to achieve a particular QoS. A set of profiles is used to match the available/requested QoS for a data stream and QoS filters. This mapping

Table 3. QoS filter types.

Filter mechanism	Filter type
Frame dropping	Simple frame dropping filter, priority based frame dropping filter
Codec filters	Transcoding filter, compression/decompression filter
Color reduction	Color-to-monochrome filter, DC-color filter, dithering filter
DCT-filters	Low-pass filter, re-quantization filter, limiting filter
Mixing filter	Interleaving frame mixer, intra-frame mixer, video & audio multiplexer, audio mixer
Splitting filter	Individual QoS splitter, hierarchical splitter

is part of the QoS negotiation process and done by the extended RSVP module in the end-system. Thus, the use of QoS filters does not complicate the specification of QoS for the user.

The QoS adaptation service can operate in two modes, viz. *controlled-load* and *guaranteed*. In the controlled-load mode a dynamic adaptation to changes in network QoS is possible, i.e., QoS filters can be instantiated dynamically. However, it is not expected that this occurs frequently since the controlled-load service itself displays the behavior of a “best-effort service under unloaded conditions”. Hence, the achieved QoS should be stable and fluctuation should be rare.

4.3. Resource reservation and filter instantiation

QoS filter instantiation and control is fully integrated into the RSVP resource reservation process using mainly *Path* and *Resv* messages. A *Path* message collects (in addition to information about resource availability stored in the *Adspec*) information about the availability of QoS filters *en route*. Each Filter Allocator keeps information about QoS filters present in network nodes further up-stream. Once the *Path* message reaches the receiver, resource and QoS filter availability are known and can be attuned with the user requirements. In this process the optimal combination of QoS filters to match resource availability and the flowspec are computed and placed in the reservation request.

4.3.1. Filter instantiation and control. The reservation request including QoS filter information is issued by the receiver and travels up-stream towards the sender along the reverse path. Apart from reserving the necessary resources, the Filter Allocator in a network node also puts the required QoS filter(s) in place. The reservation request is forwarded without any change if the Filter Allocator detects that similar QoS filter operations can be also executed further up-stream. If this is not the case, filters are instantiated and the *Flow Spec* and *QoSadspec* are changed accordingly. Any Filter Allocator in up-stream nodes acts in the same way. Hence, a filter is always propagated up-stream to the most optimal point (in terms of resources) in the dissemination tree. When reservation requests are merged this also effects the propagation of QoS filters. The request of a receiver with higher QoS requirements has priority. A QoS filter for the lower quality data stream(s) is installed and the

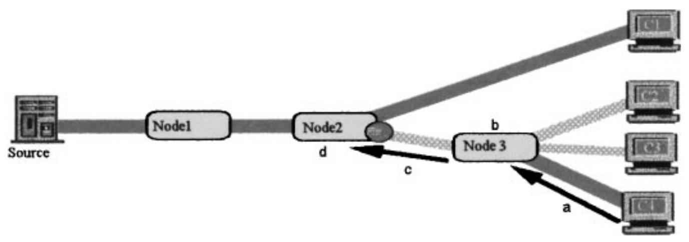


Figure 6. Join of a new participant.

new *QoSadspec* is forwarded in the merged reservation request. This new *QoSadspec* will usually be equivalent to the one of the highest quality data stream at this node. A *ResvConf* messages is sent to the user with the lower QoS requirements if this was requested.

The release of QoS filters is entirely in line with the release of resources in RSVP. Whenever a *ResvTear* or *PathTear* message arrives all QoS filters associated with this message will be removed. If a reservation times out, all QoS filters associated with the data stream will be released along with the resource reservation.

4.3.2. Join and leave of participants. When a new user joins the communication filters might have to be instantiated in nodes where the reservation request of the new participant is merged with already existing reservations. The case where the new participant has lower quality requirements is relatively straight forward. A filter is simply instantiated for the new data stream in this node, no change of the reservation request is required. Figure 6 illustrates the case where the QoS requirements of the joining user are higher than existing reservations. When the reservation request (a) of the joining user reaches Node 3, QoS filters for the existing data streams are installed (b) and the new request is forwarded to the next up-stream router (c). This *Resv* message will release any up-stream filter that adapted the data stream to the requirements of the existing users on this sub-tree and reserves new resources (d). Thus, the filter is effectively propagated down stream. If the new reservation request fails (indicated by a *ResvErr* message) the old reservation and any previously instantiated filter remains in place and the new filter(s) instantiated at the merging point are released.

When a participant leaves, its reservation is either explicitly released (using a *ResvTear* message) or it times out. If, at a merging point, the reservation of the leaving participant was lower than the merged reservation request (previously passed up-stream), then the QoS filter and any state information associated with this data stream are released.

If the reservation of the leaving participant was matching the merged reservation request (as depicted in figure 7) and all other down stream reservations are lower, the ResvTear message (a) causes the generation of a new, lower reservation request and the release of the filter(s) of the now highest quality receivers(s) on this sub-tree (b). This new reservation request is propagated up-stream together with the respective QoS filter (c) which is then installed in the next up-stream node (d).

4.3.3. Non-QoS filtering clouds. Non-RSVP clouds affect the QoS adaptation class in the same way than the controlled-load service and guaranteed service. No additional

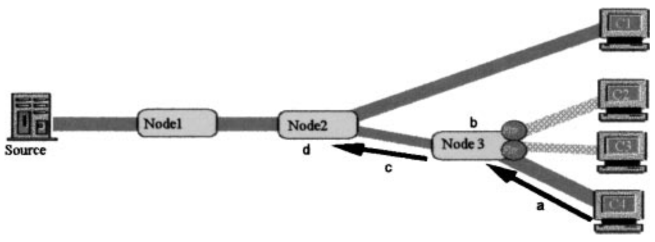


Figure 7. Leave of participant.

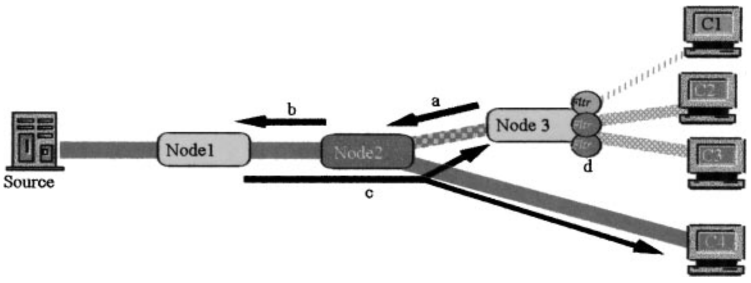


Figure 8. Configuration with non QoS filter node.

mechanisms for the QoS adaptation class in this case are required. However, the case of "non-QoS filtering clouds" (i.e., RSVP routers without the QoS adaptation module) requires special treatment. When not all intermediate nodes are capable of performing QoS filtering it is possible that although the requested QoS filter type is available further up-stream, it cannot be instantiated because the reservation request was merged with a request for higher QoS in a non-QoS filtering capable node. An example for this case is illustrated in figure 8. Node 2 is a non-QoS filtering capable node. The QoS filter type requested by C2 and C3 is available in Node 1. Hence Node 3 will propagate its filter request further up-stream (a). However, since the reservation request was merged with that of C4 in Node 2, the QoS filter has to be instantiated at a down stream node (ideally it would be placed in Node 2). Since Node 2 is a non-QoS filtering capable node, any data related to QoS filtering is meaningless to its RSVP module. Thus, it simply copies and forwards the QoS filtering request (b). From the point of view of the QoS adaptation module in Node 3 this causes an error and a special strategy to deal with this case is required.

In order to enable the next up-stream QoS filtering node to discover that reservation requests have been merged by a non-QoS filtering node, the *QoSadspec* has to contain additional information about the original QoS associated with the filter request. Therefore, the relevant QoS parameters (e.g., token bucket rate and size) are also part of the *QoSadspec*. When the reservation requests are merged in the non-QoS filtering capable node, the *QoSadspec* of the lowest QoS requested (i.e., the one with a definite QoS filter request) is copied in the new *Resv* message. The next up-stream QoS filtering capable node is now able to discover if incongruent requests have been merged by comparing the QoS parameters of the *Flow Spec* with those in the *QoSadspec*. If this is the case a *ResvErr* message is sent indicating the failure of instantiating the requested filter (c). The error message also contains the *Flow Spec* of the merged data stream and the *QoSadspec* of the lowest QoS stream. According to this information the down stream Filter Allocator(s) can determine which QoS filter request was unsuccessful further up-stream and where the respective QoS filter(s) has to be instantiated (d).

Following the instantiation of the QoS filter(s) a new *Resv* message is sent containing the adjusted *QoSadspec* and *Flow Spec*. The node that previously issued the *ResvErr* changes the *QoSadspec* in the periodic *Path* messages indicating that the particular QoS filter operation which failed is not available further up-stream. This avoids any attempt to propagate the same QoS filter request in subsequent *Resv* message which would cause the same error.

The QoS filter can be propagated again further up-stream if the receiver with the higher quality requirements leaves the communication session. In this case the last QoS filtering capable node will place a non empty *QoSadspec* in the *ResvTear* message. Hence, any intermediate non-QoS filtering capable node will forward the *ResvTear* message regardless of whether reservation requests were previously mergeds.[6] However, the fact that there was a merge of reservation requests previously is indicated by setting the *Flow Spec* in the *ResvTear* message to zero. This causes a state change in the next up-stream QoS filter capable node which in its next Path message indicates that a propagation of the QoS filter that initially failed is again possible. The *ResvTear* message will not be forwarded any further by this node.

5. Implementation issues

A prototype of the integrated QoS filter control as part of RSVP has been implemented at Lancaster University within the GCommS project [3] under the label RSVP++. The implementation concentrates on the QoS filter related and integration aspect. It comprises the modified RSVP protocol (i.e., RSVP++), the RSVP Daemons in routers, RSVP Modules in the end-systems and adapted QoS filter components. Resource reservation and traffic control are not part of the prototype. The development environment and experimental testbed consists in 486 66 MHz and Pentium 166 MHz PCs connected via a standard ethernet network. The PCs can be operated as end-systems or routers.

The development environment does not allow the direct access and programming of routers. Hence all RSVP++ components are implemented as user processes and all RSVP messages exchanged between them are UDP encapsulated. Messages to multiple participants are duplicated at this level, no network level multicast is used. IP multicast could be used directly if the respective functions would be implemented in the IP multicast forwarding.

5.1. System components

The major components of the extended functional model of RSVP++ are the *RSVP Module, RSVP Daemon* and the *QoS Adaptation Module*. Figure 9 shows all components of the RSVP++ system and how they are related. The QoS adaptation module consists of the

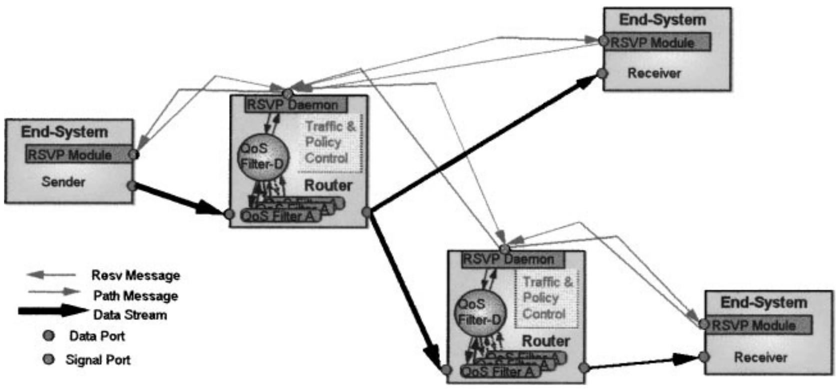


Figure 9. RSVP++ components.

QoS Filter-D (the filter control daemon) and *QoS Filters-A* (the QoS filter agents). The QoS filter mechanisms are not affected by their integration in the RSVP++ system and operate as described in [11]. The main control instance in an intermediate note is the *RSVP Daemon*. The RSVP Module in the end-system is an agent associated with an application process that translates application requirements into RSVP messages.

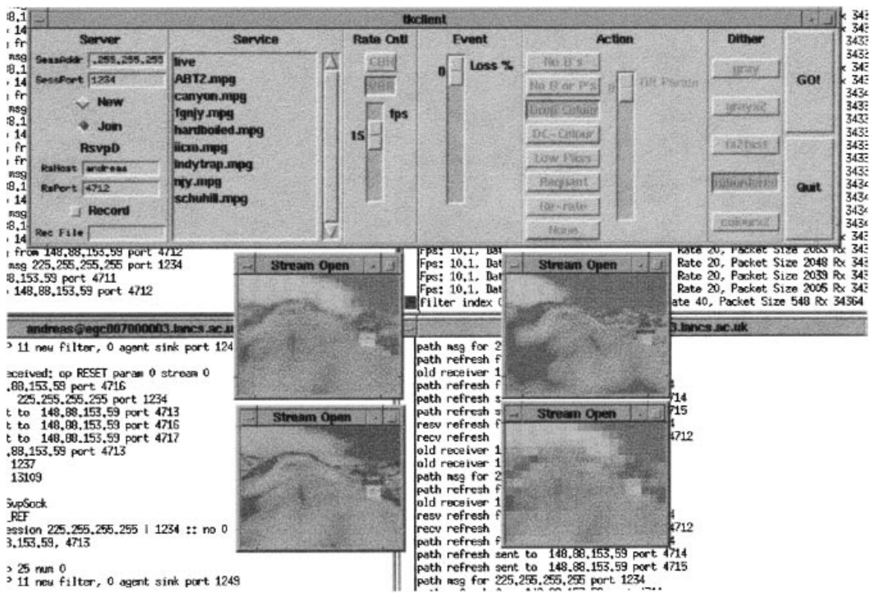
The *RSVP Daemon* is responsible for the management of RSVP sessions, this includes the periodic exchange of *Path* and *Resv* messages to up-date reservation information and maintain the "soft-state" of a session. Whenever a message arrives the RSVP Daemon checks if it is a new request or a refresh message. In the former case it creates a new management entry and informs the other components. When it is a refresh message the information currently held is updated and in the case of changes other components are informed. The RSVP Daemon does, however, not interpret the information contained in the actual messages that is opaque data to it. It only extracts this information and passes it to the respective modules. When a timer expires the RSVP Daemon acts according to the RSVP specification. That is, it send *PathTear* respectively *ResvTear* messages and passes the infomation to the QoS Filter-D. QoS filtering related information can be part of *Resv, Path, ResvConf, ResvErr, PathTear* and *ResvTear* messages.

The QoS *Adaptation Module* consists of one QoS Filter-D and a number of QoS Filter-As. The QoS Filter-D is responsible for the management of the components performing the filtering, viz. the QoS Filter-As. The QoS filter daemon keeps all status information related to QoS filters and their control. With this information it is able to decide whether QoS filters can, and should, be propagated or if it is responsible for instantiating them locally. If a filter should be propagated it asks the RSVP Daemon to send a new *Resv* message with the respective QoS filter request. For each stream the QoS Filter-D instantiates one QoS Filter-A which serves all filter requests for that stream at this particular node. The QoS Filter-A accommodates all requests for a particular stream. Those requests might be for different filters and/or from different clients.

The RSVP Daemon and the components of the QoS Adaptation Modules run as separate processes and communicate using messages. This design makes it possible for the different tasks to run independently. As a result both resource reservation and QoS filtering modules can be replaced without affecting the other component. Even the RSVP protocol could be replaced by another resource reservation protocol with similar features. The QoS filters only rely on the information passed between the RSVP Daemon and the QoS Adaptation Module and the capability of the resource reservation protocol to carry this data but not on the protocol itself.

The *RSVP Module* in the end-system is the application interface to RSVP++. Applications initiate reservations and control filters via this interface. The IETF does not specify an API for RSVP, hence the implemented RSVP Module only conforms to the RSVP specification itself. The actual interface definition was influenced by an existing application framework. There is a distinction between the sender RSVP Module and the receiver RSVP Module. An RSVP Module is associated with every sending and receiving process. The sender can create sessions and send *Path* messages along the established multicast route. The receiver can join an existing session but neither create nor delete it. The RSVP Module handles all incoming RSVP messages and only informs the user in the case of major state changes (e.g., when a session is deleted). It is also responsible for sending periodic refresh messages and dealing with time-outs indicating that there is a problem in an up-stream (or down-stream) router.

The application framework of the experimental systems consists of a file server (that incorporates a daemon and a source agent) and a client constructed around the MPEG 1 software video player [5]. Figure 10 shows an example of the application framework with four clients connected to the server via one respectively two intermediate systems[7]



SessAddr .255.255.255
SessPort 1234
RsHost andreas
RsPort 4712
Rec File

live
ABT2.mpg
canyon.mpg
fgnjy.mpg
hardboiled.mpg
iicm.mpg
indytrap.mpg
njy.mpg
schuhill.mpg

```
 from 148.88.153.59 port 4712
msg 225.255.255.255 port 1234
8.153.59 port 4711
 148.88.153.59 port 4712
```

andreas@egc007000003.lancs.ac.u

```
> 11 new filter, 0 agent sink port 124

eceived: op RESET param 0 stream 0
.88.153.59 port 4716
  225.255.255.255 port 1234
t to  148.88.153.59 port 4713
t to  148.88.153.59 port 4716
t to  148.88.153.59 port 4717
.88.153.59 port 4713
 1237
 13109

SvpSock
_REF
ession 225.255.255.255 | 1234 :: no 0
3.153.59, 4713

> 25 num 0
> 11 new filter, 0 agent sink port 1249
```

```
Fps: 10.1. Dat          Rate 20, Packet Size 2048 Rx 343
Fps: 10.1. Dat          Rate 20, Packet Size 2039 Rx 343
Fps: 10.1. Dat          Rate 20, Packet Size 2005 Rx 343
filter index (          ate 40, Packet Size 548 Rx 34364
```

3.lancs.ac.uk

```
path msg for 2
path refresh f
old receiver 1
path refresh f
path refresh s                    714
path refresh s                    715
resv refresh f
recv refresh                      4712
old receiver 1
old receiver 1
path msg for 2
path refresh f
old receiver 1
resv refresh f
recv refresh                      4712
path refresh f
path refresh sent to  148.88.153.59 port 4714
path refresh sent to  148.88.153.59 port 4715
path msg for 225.255.255.255 port 1234
```

Figure 10. RSVP++ application framework.

(i.e., RSVP++ router modules). The client window on the upper left hand side shows an image after re-quantization filtering. This is filtered at node 4712. Node 4713 is the down-stream node of 4712; the right hand images are filtered at this node. The lower right hand one displays the result of low-pass filtering.

5.2. Message formats and RSVP QoS filtering objects

Continuous media data units are UDP encapsulated using the RTP standard header [6] for the required sequence numbers, timing information, and payload types. The payload type field in RTP is currently 7 bit, i.e., there can be 128 different payload types. This is for the time being sufficient but may prove to be not enough considering the rapidly increasing number of new encoding standards and media formats.

All QoS filtering related information is carried in two structures: `Data_type` which gives the type of the media stream that has to be filtered and corresponds to the payload type carried in the RTP header, and `QoS_ad_spec` containing all relevant information for the instantiation and management of QoS filters. These structures are part of the receiver `Flow_Spec` object of *Resv, ResvTear, ResvErr, ResvConf* messages, and the `Sender_Template` object of *Path* and *PathTear* messages. Further, the `QoS_ad_spec` is also part of the RSVP `Adspec` object in a *Path* message. The `Adspec` carries information about available resources and in RSVP++, also available QoS filters *en route*. No additional objects or RSVP messages are necessary, i.e., the extension of the RSVP messages and objects in RSVP++ is restricted to two additional structures.

The `QoS_ad_spec` contains four structures to control QoS filters and a duplicate of the flow spec which is needed to check if reservations were merged in non-QoS filtering capable RSVP routers. Figure 11 shows the structure of the `QoS_ad_spec`.

The `filter_index` contains all information required for the management of the QoS video filters. It is essentially a 32 bit long bit map that enables a QoS filter agent to instantiate and control different filters. The filters are colour reduction filters, low pass filters, frame dropping filters, re-quantization filters and smoothing filters. As part of the `Adspec` this structure carries information about QoS filters available in any up-stream node. This is indicated by a ‘1’ if the filter is available and ‘0’ otherwise. The sender uses this structure as

```
 typedef struct QoS_ad_spec {
    struct Video_filter filter_index; /*video filter index bit
map*/
    struct Audio_filter audiofil;     /*resrv. for audio
filter*/
    struct Mixing_filter mixfil;      /*resrv. for mixing
filter*/
    struct Ext_filter exfil;          /*resrv. for future use*/
    struct Comp_flowspec flowsp;      /*duplicated flow spec*/
  } QoS_ad_spec;
```

Figure 11. `QoS_ad_spec` structure.

```
typedef struct Video_filter {
   unsigned ColMo       : 2;  /*colour reduction filter*/
   unsigned LowPass     : 6;  /*low pass filter cut-off values*/
   unsigned FraDr       : 2;  /*frame dropper*/
   unsigned FDPer       : 7;  /*percentage of frames to drop*/
   unsigned Quant       : 5;  /*re-quantization filter*/
   unsigned Smoothing   : 10; /*smoothing filter*/
 } filter_index;
```

Figure 12. Data structure for QoS filter control.

part of the `Sender_Template` to inform the receiver which filter operations are permitted. Finally, the `Flow_Spec` in a reserve message carries concrete filter instructions including all required parameters for filters to reduce the quality in discrete steps. The structure is shown in figure 12.

Compared with other RSVP objects the overhead incurred by the `QoS_filter_spec` and `Data_type` is relatively low (i.e., 10*32 bits) and does not require much additional bandwidth.

6. Conclusion and future work

The provision of QoS in a heterogeneous communications environment is still one of the most pressing problems in distributed multimedia systems. Particularly in open systems were a number of participants want to communicate it is necessary to find an efficient way of supporting individualistic QoS for disparate receivers. The Internet resource reservation protocol RSVP is receiver oriented and allows different levels of QoS for different receivers. However, it does not specify how the QoS requirements of a data stream can be reduced to accommodate the specific QoS requirements of a receiver.

The QoS filters developed at Lancaster University provide mechanisms to change the structure of a media stream and hence to adapt its QoS requirements. Our experiences with the use of QoS filters have shown that it is possible, in heterogeneous environments, to meet the distinct requirements of different participants within distributed multimedia applications. In order to provide optimal QoS support it is, however, necessary to tie resource reservation and QoS filter instantiation and control. In this paper we introduce an enhanced version of RSVP which fully integrates QoS filter control in the functional model of RSVP. A special QoS service class, the QoS adaptation service, is defined for the specification of resource and QoS filtering requirements. Additional, a new RSVP object, the *QoSadspec*, is used to convey QoS filter control information. RSVP++ is capable of interoperating with standard RSVP implementation such that system openness is ensured.

The implementation of RSVP++ concentrates on the QoS filter aspects; resource management is not part of the current system. The main components are the RSVP Daemon, the QoS filter daemon (QoS Filter-D) and filter agents (QoS Filter-A). For the control of the video filters a structure of 32 bits suffices though provisions to accommodate more and other QoS filter types have been made. QoS filters are an effective tool to reduce QoS requirements of a continuous media data stream. By integrating QoS filter control and resource reservation, QoS requirements of heterogeneous receivers can be accommodated.

QoS filters can influence the system performance in various ways. However, the additional overhead introduced through QoS filtering operations on the transmission delay is limited. In fact, in some cases the overall end-to-end delay is reduced by applying QoS filters rather than increased. The extra information that has to be carried in RSVP messages is also restricted (i.e., 10^* 32 bits). There are additional management costs for QoS filters mainly during set-up and when a resources reservation changes. However, these processes can be handled in parallel to the resource reservation. Hence, in an environment with active communication and resource management, QoS filters will not heavily increase costs or decrease the performance of the overall system and are therefore suited to provide individualistic QoS to heterogeneous receivers.

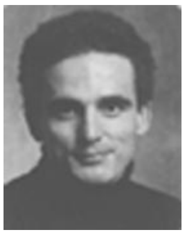
Notes

1. Note, QoS filters change the structural composition of a data stream to adapt its QsP requirements. They are completely different from RSVP filters that determine how reserved bandwidth should be shared between multiple streams. Since the QoS filtering concept was developed concurrently with RSVP and has been known in the research community under this name for some time we do not want to replace it by a new expression. In order to avoid confusion QoS filters are always addressed by their full name throughout this paper.
2. Note, reservations are made for sessions and, together with the concept of filtering, they might apply to a number of flows.
3. Note, RSVP filters and QoS fiters are different concepts. The former specify how resources should be shared between different streams, whereas the latter change the structure of data streams and reduces their resource requirements.
4. In IPv4 the flow ID is replaced by a combination of IP address and port number.
5. Note, most currently specified filters only apply for video streams. Filters that can be used with audio streams are Codec Filters, Audio Mixers and Splitters.
6. Note, a *ResvTear* is usually suppressed in these circumstances.
7. The client windows on the left hand side are connected to node 4712 (i.e., there is only one intermediate system between the sender and the clients) whereas those on the right hand side are connected via node 4713 linked to 4712 (i.e., there are two intermediate systems).

References

1. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation Protocol (RSVP)—Version 1 functional specification,” Internet Draft, IETF, 1996.
2. F. Garcia, D. Hutchison, A. Mauthe, and N. Yeadon, “QoS support for distributed multimedia communications,” in Proc. International Conference in Distributed Processing, Dresden, Germany, Chapman & Hall, 1996, pp. 463–477.
3. A. Mauthe, L. Mathy, and D. Hutchison, “Communication services for multimedia systems,” in Proc. High-Performance Networks for Multimedia Applications, Schloss Dagstuhl, Germany, 1997.
4. J.C. Pasquale, G.C. Polyzos, E.W. Anderson, and V.P. Kompella, “Filter propagation in dissemination trees: Trading off bandwidth and processing in continuous media networks,” in Proc. 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Lancaster, 1993, pp. 269–278.
5. L. Rowe and B. Smith, “A continuous media player,” in Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV’92), San Diego, California, 1992, pp. 334–344.
6. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” Request for Comments, RFC-1889, Category: Standards Track, IETF, 1996.

7. S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," Internet Draft, Draft-ietf-intserv-guaranteed-svc-06.txt, IETF Integrated Services WG, 1996.
8. S. Shenker and J. Wroclawski, "General characterization parameters for integrated service network elements," Internet Draft, Draft-ietf-intserv-charac-02.txt, IETF, Integrated Services WG, 1996a.
9. S. Shenker and J. Wroclawski, "Network element service-specification template," Internet Draft, Draft-ietf-intserv-svc-template-03.txt, IETF, Integrated Services WG, 1996a.
10. J. Wroclawski, "Specification of the controlled-load network element service," Internet-Draft, Draft-ietf-intserv-ctrl-load-svc-03.txt, IETF Integrated Services WG, 1996.
11. N. Yeadon, "Quality of service filtering for multimedia communications," Ph.D., Lancaster University, 1996.
12. N. Yeadon, F. Garcia, A. Campbell, and D. Hutchison, "QoS adaptation and flow filtering in ATM networks," in Proc. Second IWACA '94, Heidelberg, Springer-Verlag, Heidelberg, 1994, pp. 191–202.
13. N. Yeadon, F. Garcia, D. Hutchison, and D. Shepherd, "Filters: QoS support mechanisms for multipeer communications," Journal on Selected Areas in Communications, JSAC, Vol. 14, No. 7, pp. 1245–1262, 1996.
14. N. Yeadon, A. Mauthe, F. Garcia, and D. Hutchison, "QoS filters: Addressing the heterogeneity gap," in Proc. European Workshop IDMS '96, Berlin, Germany, 1996, pp. 227–244.
15. L. Zhang, R. Braden, D. Estrin, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP)—Version 1 functional specification," Internet Draft, 1994.
16. L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP—A New Resource ReSerVation Protocol," IEEE Network Magazine, 1993.



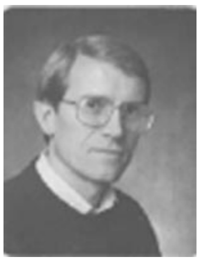
Andreas Mauthe received a Ph.D. in Computer Science from Lancaster University in 1998. His work was concerned with end-to-end support for multimedia multipeer communications. From 1994 to 1997 he was employed as a Research Associate at the Computing Department of Lancaster University, UK. The main areas of his research was multimedia group communication and collaboration, QoS provision and resource management.

In December 1997 Andreas Mauthe joined the Digital Media Division of TECMATH in Kaiserslautern, Germany. Initially he worked on two European funded projects in the area of electronic commerce, collaboration between radio and television broadcasters and digital asset management. Since January 1999 he is Chief Development Officer. In this role he is responsible for the development of media archive (a multimedia content management system mainly deployed in the broadcast industry).

Dr. Nicholas Yeadon completed his Ph.D. entitled "Quality of Service Filtering for Multimedia Communications" in May 1996 at Lancaster University. His Ph.D. centred on continuous media filtering mechanisms to dynamically adjust compressed media data streams in distributed heterogeneous environments.

After his Ph.D. he worked for two years as a Research Associate at Lancaster on the development of multimedia based distributed group applications for very low speed, mobile, networks. He now works at the British Broadcasting Corporation, Research and Development Department. While at BBC R&D he has been involved in quality preservation of cascaded compressed audio streams and in the evaluation and development of the DVB Multimedia Home Platform.

Francisco J. Garcia is a computer science honours graduate from Lancaster University and in 1993 was awarded a Ph.D. in the area of protocol support for distributed multimedia applications. From 1993 till 1995 he worked at Lancaster University as a senior research associate on the Quality of Service Architecture project (QoS-A) looking at the design and implementation of new transport services. Here he took an interest in compression technology and applied it to research on flow filtering mechanisms to provide Quality of Service (QoS) support for heterogeneous groups. During this time he was also actively involved in ISO/IEC JTC1/SC21 WG1 on the development of a QoS Framework. In October 1995 he joined Hewlett Packard Laboratories in Bristol where he worked on highly scalable distributed measurement systems. Currently he is a Senior Member of Technical Staff (SMTS) od Agilent Laboratories, Edinburgh.



David Hutchison is Professor of Computing at Lancaster University, where he has worked for the past fifteen years. He has published numerous papers, and edited several books, on services and support mechanisms for distributed multimedia applications. A prominent theme in his research is Quality of Service. The Distributed Multimedia Research Group in which he works is internationally renowned and is involved in many research collaborations with industry and with other universities, both within and without the UK. He serves on advisory and assessment panels for the EPSRC and the European Commission, and is a technical expert adviser at OFTEL. During a recent sabbatical year, he was an academic visitor at HP Labs in Bristol, UK, at EPFL in Lausanne, Switzerland, and at BT Labs in Ipswich, UK.