Multimedia Systems 6: 299–315 (1998)

Multimedia Systems

Quality-of-service adaptation in distributed multimedia applications

Abdelhakim Hafid, Gregor v. Bochmann

Université de Montréal, Dept. d'Informatique et de Recherche Opérationnelle, Montréal, H3C 3J7 Canada; e-mail: {hafid, bochmann}@iro.umontreal.ca

Abstract. High-speed networks and powerful end-systems enable new types of applications, such as video-on-demand and teleconferencing. Such applications are very demanding on quality of service (QoS) because of the isochronous nature of the media they are using. To support these applications, QoS guarantees are required. However, even with service guarantees, violations may occur because of resources shortage, e.g., network congestion. In this paper we propose new adaptation approaches, which allow the system to recover *automatically*, if possible, from QoS violations (1) by identifying a new configuration of system components that might support the initially agreed QoS and by performing a user-transparent transition from the original configuration to the new one, (2) by redistributing the levels of QoS that should be supported, in the future, by the components, or (3) by redistributing the levels of QoS that should be supported immediately to meet end-to-end requirements based on the principle that (local) QoS violation at one component may be recovered immediately by the other components participating in the support of the requested service. The proposed approaches, together with suitable negotiation mechanisms, allow us (1) to reduce the probability of QoS violations which may be noticed by the user, and thus, to increase the user confidence in the service provider, and (2) to improve the utilization of the system resources, and thus to increase the system availability.

Key words: Quality of service – Adaptation – Violation – Recovery

1 Introduction

Many distributed systems currently in use have been designed around low-speed network technology, such as Ethernet and Token Ring. They have been adequate and useful for text and numerical data applications in which relatively small amounts of data have to be transmitted. For most of these applications, performance and quality of service (QoS) have not been a major problem. This changes when more and more distributed multimedia (DMM) applications are developed, which require transmitting and processing large amou nts of data (in particular video and audio) in real time.

Due to increasing demands of DMM applications, efficient and effective support of QoS has become increasingly important. To support QoS requirements, communication systems and end-systems must provide latency and bandwidth characteristics that allow timely transmission of information. DMM applications require a system that *maintains* the initially agreed QoS, *regardless of the fluctuation* in system load; otherwise DMM applications will not be able to compete against traditional systems, such as television. A number of schemes have been proposed to provide deterministic and/or statistical QoS guarantees spanning end-systems and networks. Such guarantees can be provided through resource reservation in the network and the end-systems for each user request serviced and access control in order to limit the number of serviced user request in case of temporary system overload. Certain network services provide such guarantees; however, the Internet traditionally provides a "best effort" service without QoS guarantees. DMM applications which depend on a certain level of QoS need a mechanism for QoS adaptation in order to deal with temporary changes in the available QoS parameters. Such changes may not only occur in "best effort" environments but also, less frequently, in networks providing resource reservation, for instance when certain partial system failures occur (Hanko et al. 1991).

We note that a renegotiation of the QoS qualities of a DMM application may also be initiated by the user during an ongoing session. The user may wish to increase the quality or reduce it in order to reduce the costs. The internal mechanism for adapting the application to this new situation is similar in this case to the mechanism used for adapting to changing network and server QoS parameters.

1.1 Existing approaches to QoS-related problems

In general, resource overload is solved by application- and user-defined policies. The system must detect (end-to-end) QoS violations by using some monitoring mechanisms, and

Correspondence to: A. Hafid

follow the policies to solve, or react to, overload situations. Most existing approaches (Topolcic 1990; Gilge and Gussella 1991; Yin and Hluchyi 1991; Parris et al. 1994; Tobe et al. 1992) for managing QoS violations have one or more of the following characteristics: (1) they are restricted to the communication sub-system; (2) they react only after the occurrence of end-to-end QoS violations, which means that the problem is passed on to the user or the application; or (3) they react to QoS violation by renegotiating a degraded QoS and are thus restricted to applications that can adapt to varying QoS.

Steinmetz (1990) suggests a partial blocking mechanism which allows the specification of actions to be taken in the face of a loss of synchronization, e.g., for video, redisplay of the previous frame to deal with lost and late frames. However, such a mechanism may be useless if the media stream cannot be delivered for a long period of time; for instance, for an audio stream 10 ms is a long period. Tawbi and Horlait (1994) propose mechanisms to specify, at the application level, the actions to be taken when QoS violations occur. However, those actions consist only of degrading the initially agreed QoS. Gilge and Gussella (1991) make use of rate and flow control, where feedback from the network is used to adapt coding parameters and to vary the output rate. This allows the sources to dynamically adjust their maximum transmission rates to match the available resources.

A significant contribution on QoS adaptation has come from the Tenet Group at the University of California at Berkeley. Parris et al. (1994) propose a network adaptation scheme (called graceful adaptation service) to increase the adaptivity of real-time networks. A graceful adaptation service dynamically manages the QoS of real-time communications by changing the parameters that specify it during the lifetime of the connection. The scheme provides the change from the old QoS to the new one with no or limited disruption. It is implemented using routing, performance monitoring, dynamic rerouting, load balancing and control mechanisms. Dynamic rerouting is needed to establish the alternative route and to perform "transparent" transition from the primary route to the alternative route. Performance monitoring is required to monitor the currently provided QoS; the load-balancing module determines whether to attempt load balancing using an algorithm defined in Parris et al. (1994). If load balancing is possible, the routing mechanism determines an alternative route that has the highest probability of successful channel establishment, and the transition from the primary route to the alternative route is accomplished using the dynamic rerouting mechanism.

Graceful adaptation service is the exception in that it tries to maintain, if possible, the initially agreed QoS instead of degrading it when QoS violations occur. However, the proposed service (1) concerns only communication systems, that is, it reacts to network performance violation by the establishment of an alternate route, and (2) is closely related to the Tenet protocol suite.

1.2 Our proposals

Our work on QoS adaptation has been guided by the following premises: (1) QoS adaptation should be performed automatically when possible; (2) QoS violations should be dealt with locally at the component level; and (3) QoS adaptation should maintain the initially agreed QoS as long as possible, before any QoS degradation is initiated. In this paper, we consider two approaches to QoS adaptation. The first approach involves a reconfiguration of the application infrastructure, replacing the overloaded system components by other alternative components. The second approach does not change the configuration of components, but changes the QoS characteristics allocated to the different components.

Adaptation at the configuration level. When a violation is detected, one or more alternate components are selected, and a transparent transition from the primary components to the alternative ones is performed. The alternative components are selected based on several factors, such as the functional configuration of the requested service, and the current load of system components. The QoS characteristics considered by this approach are delay, jitter, throughput, and reliability (expressed in terms of loss rate); that is, the approach may be used to recover from delay, jitter, throughput, and/or loss rate violations. This may be useful for any application that requires certain guarantees on QoS, such as video-on-demand and teleconferencing systems.

Adaptation at the component level. In order to provide end-to-end QoS guarantees, each system component involved in the QoS provision must contribute its share to the requested end-to-end QoS (Bochmann and Hafid 1997) (each component providing certain guarantees). When a component violates its guarantees (and this is detected by the system), some form of cooperation among the components is started with the aim of reassigning the guarantees to the different components, so that the end-to-end guarantees, as observed by the user, are unaffected. Thus, the non-overloaded components may reserve additional resources, e.g., buffers or CPU slots, in order to provide an improved QoS and thus compensate the violations of the other components. The QoS characteristics considered by this approach are delay, jitter and reliability (expressed in terms of loss rate); that is, the approach may be used to adapt from delay, jitter, and loss rate violations. This may be useful for any application with stringent temporal requirem ents, such as teleconferencing systems.

When the system cannot recover from QoS violations (using either of the approaches), users or applications should be required to intervene. The user should be informed directly at the user interface. If a violation occurs, a special notification is sent to the application/user, who can react accordingly. Generally, the interactions with the user lead to the renegotiation of a degraded QoS or simply to the abortion of the application (as in most existing systems).

In this paper we propose three schemes based on the above approaches. The first scheme, called component reconfiguration scheme (CRS), performs adaptation at the configuration level, as explained above. The second scheme, called resource reconfiguration scheme (RRS), recovers from QoS violations by changing the distribution of QoS levels that each component will support in the near future. The third scheme, called delay recovery scheme (DRS), recovers from transit delay violations immediately, so that the failure of one or more components to meet their commitment does not necessarily lead to an end-to-end QoS violation.

The proposed schemes, together with suitable negotia- mechanisms (Bochmann and Hafid 1997), (1) will make re efficient use of system resources, thus increasing the em availability, and (2) will increase the user acceptance DMM by decreasing the probability of QoS violations no- d by the users. To support our approach, the following umptions are made:

the distributed system supports QoS guarantees, that is, the components are able to reserve resources to support certain levels of QoS, and
internal monitoring mechanisms are available, which can detect local QoS violations by a given component. It is worth noting that facilities for monitoring will likely become *available with certain types of equipment* (Seneviratne and Cho 1995).

Component reconfiguration scheme

consider an overall system configuration consisting of eral (physical or software) components, which provide port for one or several concurrent DMM applications. amples of such configurations are shown in Figs. 1, 4, For many applications, especially presentational applica- ns, where a user views information coming from a re- te multimedia database, a linear configuration as shown Fig. 1 is appropriate. Most of our discussions assume such simple linear system configuration; however, certain appli- tions, such as teleconferencing, require much more com- ex system architectures due to the large number of users at could be involved. Most of our results can be extended more general system configurations.

We assume a QoS management framework where each stem component has its own QoS agent, which provides eans for handling all the information about the perfor- ance and the functional behavior of the given system com- nent. The overall QoS coordination, including QoS adap- tion, is performed by a global QoS manager which interacts ith the QoS agents of the different system components. An gent collects QoS and performance-monitoring information out the associated component and makes this informa- n available, upon request, to the QoS manager. The term component" and "QoS agent" will be used interchangeably the rest of the paper.

1 Streams

he term "stream" is used to represent a flow of informa- on with real-time properties. In DMM applications, data reams carrying audio, video, text or image data, run from e camera to the monitor, from the file server to the TV set, more generally, from the source to the sink. In video-on- emand systems, data is read from the disk, stored in buffers the sender side, transmitted over the network and again ored in receiver buffers. Often, the data will have to be ecoded before it can be presented to the user (see Fig. 1).

A multimedia stream has a number of properties related the multimedia semantics represented by the data, such s the media type, the coding scheme, the resolution, and ie throughput. A stream-processing component creates a stream (e.g., camera), absorbs a stream (e.g., display), or transforms input streams into output streams (e.g., decoder). Often, linear configurations of stream-processing components are used, such as shown in Fig. 1, to describe presentational applications, such as video-on-demand. However, multimedia applications may need several streams, e.g., one for audio and one for video, which may lead to more complex configurations, especially when inter-stream synchronization is required, as in the case of a video sequence with a separately stored voice track. Much more complex stream-processing architectures are required for multi-party conferencing, which includes multi-point delivery and intermediate processing functions, such as bridges.

2.1.1 QoS parameters

The system components, e.g., network or decoder, can be characterized by the following QoS parameters, to be evaluated over a certain measurement interval:

- *transit delay* is the time between the moment some data unit is received (at the input port) to the moment it is sent (at the output port);
- *transit delay jitter* indicates the variation of the delays experienced by different data units in the same stream;
- *loss rate* is the fraction of data units lost during transit.

We note that the throughput is a property of the stream, which is processed by the components of interest, and not a property of the components.

2.1.2 Concatenation properties

If several components are composed in a linear configuration, as shown in Fig. 1, the end-to-end QoS characteristics of such a composition can be calculated based on the QoS characteristics of each individual component. For instance, the delay of the composed system consisting of the network and the decoder is the sum of the delay of the network and the delay of the decoder. In general, the end-to-end QoS parameters P^{ee} of a linear concatenation of n stream-processing components can be calculated from the QoS parameters P^i of the i-th component ($i = 1, \cdots, n$) as follows (Bochmann and Hafid 1997):

$AvailableThroughput^{ee}$= minimum(for all $i = 1, \cdots, n$) of $AvailableThroughput^i$

$Delay^{ee}$ = sum(for all $i = 1, \cdots, n$) of $Delay^i$

$Log\,(1 - LossRate^{ee})$ = sum(for all $i = 1, \cdots, n$) of $Log\,(1 - LossRate^i)$

$Jitter^{ee}$ = sum(for all $i = 1, \cdots, n$) of $Jitter^i$ [assuming that the jitter is defined as the difference between the maximum and minimum delay]

$Jitter^{ee}$ = square-root of sum(for all $i = 1, \cdots, n$) of square of $Jitter^i$ [assuming that the jitter is defined as the average deviation of the delay from the average delay, and the delay is assumed to have a normal distribution]

It is important that these formulas apply to the actual QoS parameters that describe the performance of the system, as observed during its operation. For system management purposes, we are not only interested in these actual QoS

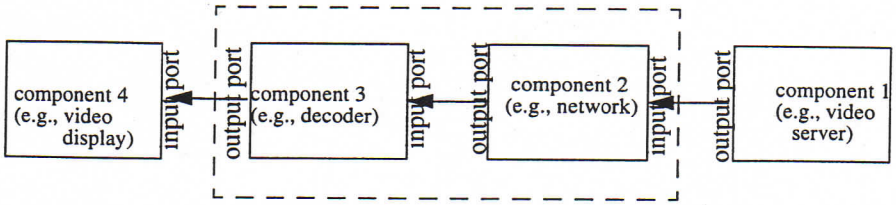

Fig. 1. Stream: linear configuration

parameters, but also in QoS guarantees that may be obtained from the different system components during the planning and initialization of the multimedia application. Different degrees of guarantees may be distinguished (Danthine 1992):

- deterministic guarantee (which means that the communication service is equal or better than the specified QoS parameters),
- statistical guarantee (which means deterministic guarantee for at least a certain fraction, e.g., 95%, of the transmitted data blocks, or for a certain fraction of the connections that are established over a long period),
- target objectives (which means that the component knows the requirements and tries to satisfy them without providing any guarantee), and
- best effort (which means that the component will do as well as it can without considering the particular QoS requirements); past experience may provide some information about how well the component usually performs.

It is clear that the above formulas remain valid for QoS guarantees as long as all components provide guarantees of the same degree. If this is not the case, the degree of guarantee of the resulting end-to-end QoS parameter is the lowest guarantee provided among the different components. For instance, if one network in the configuration only provides a best effort service, the resulting end-to-end service will only be of the same type.

We note that we consider each stream-processing component as a black box providing the specified processing function. We do not consider its internal structure which, in the case of a network, may consist of several layers of communication protocols. Only the QoS of the service provided at the black-box level is considered here.

2.2 *QoS negotiation framework*

When a user starts a DMM application, he/she should specify the desired QoS, otherwise some default QoS will be used. The QoS manager will first determine all configurations of system components that are candidates for supporting this particular instance of the application. Then, it will proceed to select an optimal configuration that has enough available resources to support the requested QoS.

The process that leads to the selection of an optimal configuration consists of the following steps. Figure 2 shows the state machine of the proposed negotiation procedure.

Step 1: The QoS manager identifies the components that may get involved in providing the requested application. The identification is based on the functional behavior of the application, and the static characteristics of the system components, such as the software functions they support and

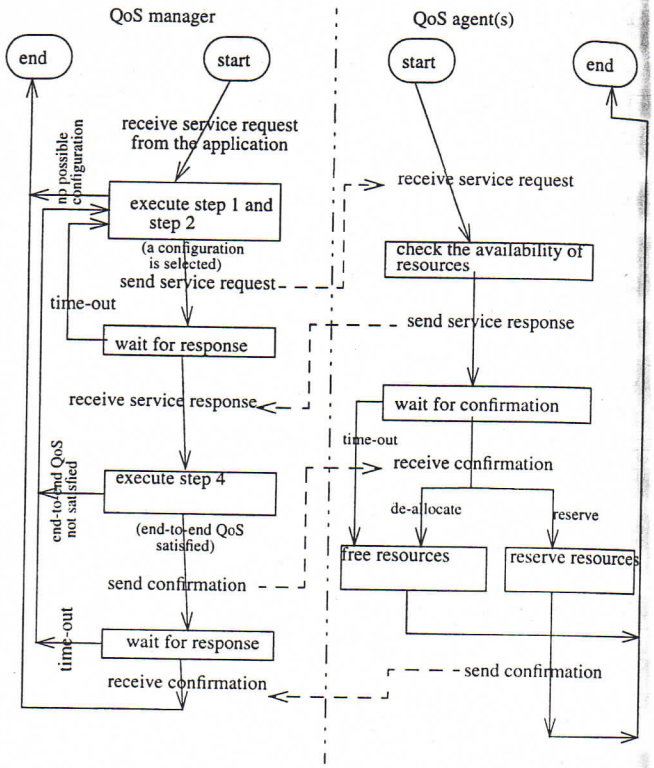

Fig. 2. Negotiation procedure -state machine-

their maximum capacity. For example, to support a video-on-demand application, the following components are considered: those servers that store the movie to be played, the networks which connect these servers to the client machine, the decoders to decode the data retrieved from these servers, and the client machine.

Step 2: The QoS manager orders the configurations produced in Step 1 according to the optimization criteria. The following static information may be used for this purpose: (1) the cost to be charged to the user for using a given configuration; (2) the availability and reliability (statistical values) of the components of a given configuration; and (3) the QoS that might be provided by a given configuration. Several algorithms may be used to classify the configurations depending on the type of applications considered; a detailed description of a classification algorithm for presentational multimedia applications, such as video-on-demand, can be found in Hafid et al. (1996).

Step 3: The QoS manager selects the best configuration and inquires about the available service quality from each of the components, via their QoS agents. On receipt of the service request, each QoS agent makes a resource reservation for the best possible level of QoS and sends this information to the QoS manager. We assume that it is always possible to

compute the maximum level of QoS that can be supported by a given component, given its current load.

Step 4: When receiving all responses, the QoS manager determines whether the configuration meets the end-to-end requirements using the formulas of Sect. 2.1. If the QoS the components are committing to support does not satisfy the requirements, another configuration is considered (go back to Step 3). This proceeds until a configuration supporting the requested QoS is found, or all configurations identified in Step 1 have been checked (or a time-out occurs). If the QoS the components are committed to support is better than the requested QoS, the commitments of certain components can be relaxed. Some policies for performing such relaxation are described in Nagarajan (1993).

Step 5: The QoS manager sends a message to each of the QoS agents in the configuration in order either to request the effective reservation of the resources or to request the deallocation of the resources that have been temporarily reserved.

For decentralized systems, we may consider an approach where the system components are divided into separate domains (Sloman and Moffet 1989). We associate to each domain a QoS manager which is responsible for the management of the components of that domain. To interact with a QoS agent, a QoS manager may use the services of other QoS managers in a hierarchical way (Hafid 1995).

2.3 The component reconfiguration scheme (CRS)

The basic idea of the CRS is to replace the overloaded component(s) by other components with the same functionality, but able to support the initially agreed level of QoS. When a QoS violation is detected for one or more components, alternate components are selected and a user-transparent transition from the primary components to the alternates is performed.

To support CRS, the QoS manager will make use of the negotiation mechanisms described above. Upon receipt of the user request for a given application and associated QoS requirements, the QoS manager produces a set of possible configurations of system components and selects a configuration which best satisfies the user requirements.

When a QoS violation occurs during the execution of the application, the QoS manager is notified by QoS agents or lower QoS managers. When CRS is supported, the QoS manager will consider another configuration, among the set produced in Step 1 and classified in Step 2, and evaluate the capacity of the components to support the user requirements via interactions with the QoS agents (Step 3, Step 4 and Step 5). If this activity succeeds, the QoS manager performs a user-transparent transition from the existing configuration to the new one; otherwise, another configuration is considered. If all the configurations are evaluated without success, the QoS manager initiates a renegotiation with the user. The selection of the alternate configuration depends on several factors such as the cost and the current load of the components. Furthermore, the following rule may be used to minimize the interactions between the QoS managers and the QoS agents. The changes to the configuration should be minimal; only the overloaded components should be replaced (as far as possible).

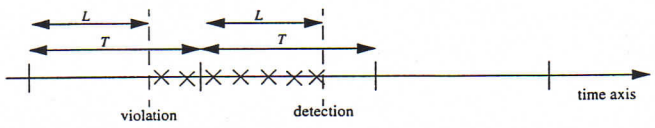

Fig. 3. The frequency and the interval of measurement parameters

2.4 Responsiveness

The response time of the CRS is the time between the moment a QoS violation is detected and the moment the reconfiguration has been completed, either resulting in a new configuration that reestablishes the originally agreed QoS, or leading to the conclusion that the agreed QoS cannot be maintained. In the case that the reconfiguration is requested by the user, this is the resulting response time of the system. However, in the case that the system should automatically detect any internal QoS violation of a component and invoke the CRS before the user will notice any end-to-end QoS violation, we also have to take into account the time required by the QoS monitoring tool for diagnosing the QoS violation. We therefore discuss these two aspects of responsiveness in the following subsections.

2.4.1 Detection of QoS violations

If the QoS violation is due to the failure of a system component, its detection is as fast as the detection of the failure, which could be immediate. However, in most cases, the QoS violation may be due to system overload, and there is no clear point in time when such a QoS violation should be signalled. We assume in the following that a monitoring tool is used to measure the QoS parameters during the operation of the system, that a single measurement of a parameter requires a time interval L, and that such a measurement is performed every T time units. The values of L and T should be selected carefully. It is not recommended to use extremely large or small values. For instance, if L takes a value equal to the duration of the active phase of the configuration, no adaptation is possible, since information about a possible QoS violation will be available only at the end of the session. If, on the contrary, we assume a small value for L, the result of the measurement may include strong statistical fluctuations because of the small number of samples on which the measurement is based. More generally, long intervals provide a more consistent view of the system, while short intervals increase the responsiveness to QoS violations. Ideally, each measurement period should be immediately followed by another one ($T = L$). However, if the method of measurement has much overhead, less frequent measurements may be desirable ($T > L$). On the other hand, T should not be too long, since it represents the upper bound on the delay between the occurrence of a QoS violation and its detection (Fig. 3).

Because of the dynamic fluctuations of the QoS actually provided, certain precautions should be taken to avoid

2.6 Case study: news-on-demand application

The CITR news-on-demand prototype (Wong et al. 1997; Hafid and Bochmann 1996) runs in a fully distributed architecture, where multimedia documents are stored at various sites, which can be accessed from different places throughout the network.

The database is the information provider. It can be supported by several database servers and stores multimedia data as well as meta-data used to facilitate searching, transfer and delivery. The documents stored in the database are composed of several monomedia objects, linked together with spatial and temporal synchronization constraints. Several physical representations may exist for a monomedia object; we here use the term variant, which corresponds to a format variant (Bochmann et al. 1996). For example, two variants of a given video sequence could offer different color qualities. Variant 1 could be a super-color variant, while variant 2 is black and white. A variant is stored on a specific server machine. The server machine is a machine located in the network, on which the objects that compose multimedia news are stored. A server machine can be either a database server, an image server or a continuous-media server. The network physically links the different machines together. Before displaying a multimedia document, the QoS manager has to select, for each monomedia object, one of its variants, since we assume that each monomedia object may exist in different physical representations (variants). A monomedia object is defined in a particular medium: a text, a still image, an audio sequence, a graphic or a video sequence. Its variants are physical objects coded on the same medium, possibly with different format representations. For instance, a monomedia document could be a video sequence and could exist in MPEG2 format as well as in MJPEG format. Variants may be replicated and stored on different servers for availability and financial purposes (Bochmann et al. 1996).

In the news-on-demand prototype, a system configuration consists of three components: the client machine, a server machine and the network connecting the two machines. The role of the QoS manager is to find a configuration which allows the system to deliver the requested document with a presentation quality that corresponds to the user's wishes and his/her financial constraints, as well as within the constraints imposed by the limitations of various system components, such as the available resources at the client's workstation, the bandwidth limitations of the network, and the encoding schemes of the available multimedia documents (Hafid et al. 1996). Several configurations may be potential candidates to support the requested service; that is, several servers store variants of the document that satisfy the service requirements. Before asking for resource commitments from the components, the QoS manager starts by sorting the set of potential configurations to produce an ordered list; a detailed description of the sorting algorithm can be found in Hafid et al. (1996). The sorting algorithm is based on several factors, such as:

- *cost*: the cost includes the transport cost, the server cost, and the copyright cost;
- *reliability and availability*: based on previous experiences and some useful statistics, the QoS manager is able to compute the reliability and availability of the system components. A configuration that contains components with high reliability and availability is preferred.

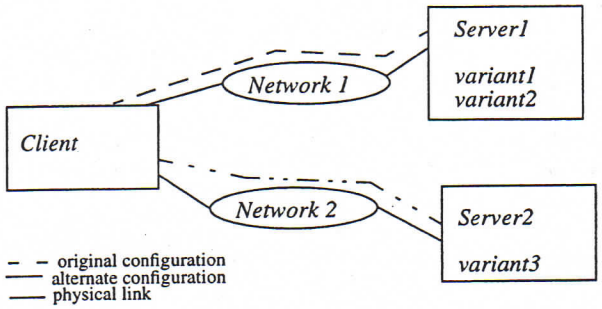

Fig. 4. Example of system component configurations

Let us assume that a configuration is selected and the corresponding resources are reserved. If, during the presentation session, the network or/and the server machine of the current configuration become congested, thus leading to lower presentation quality, the QoS manager considers the ordered list of configurations, *except the current one*, and executes the same QoS negotiation procedure, which was used during the establishment phase. If an alternate configuration is found and the required resources are reserved, the QoS performs a transition from the current configuration to the alternate one. That is, the delivery of the document will continue using the services of the alternate components. To perform the transition, the QoS manager stops the presentation of the document after having determined the current position. It then activates the alternate configuration and restarts the presentation from the current position. The transition procedure is a simple one, more sophisticated procedures may have to be used for other applications.

Example. Let us assume that two variants, *variant1* and *variant2*, of the video "San Francisco Trip" are stored on *server1*, and one variant, *variant3*, is stored on *server2* (Fig. 4), so that:

Variant1: Color = Color, Frame rate = 15 frames/s, Resolution = 320 × 240

Variant2: Color = Grey, Frame rate = 15 frames/s, Resolution = 320 × 240

Variant3: Color = Color, Frame rate = 15 frames/s, Resolution = 320 × 240

Upon receipt of the user request to play "San Francisco Trip" with (color = color, frame rate = 15 frames/s, Resolution = 320 × 240), the QoS manager finds the configuration *Server1-Network1-Client* suitable to support the user requirements.

Let us assume that, during the presentation session, *Network1* experiences congestion. The QoS manager will switch automatically to the configuration *Server2-Network2-Client*, which we assume has enough resources to support the user requirements.

If the configuration *Server2-Network2-Client* is not able to support the delivery of the requested document, the QoS manager will evaluate the capacity of the configuration *Server1-Network1-Client* to support the delivery of *variant2* of the document. If there are enough resources, the QoS man-

ager starts playing *variant2* from the point where *variant1* was stopped. This activity is called graceful degradation.

We conducted a number of experiments on CRS in the context of a news-on-demand prototype (Hafid and Bochmann 1996). To detect QoS violations, we make use of a monitoring tool (Somalingam 1996) at the transport level. Our experimental platform consists of a network employing point-to-point links coupled to a high speed ATM switch (Newbridge) to form an ATM LAN and two IBM RS/6000 workstations, running AIX. An Ethernet is also used to connect the machines.

The response time of CRS in scenarios similar to the scenario described in this example is about 2 s. We believe that this response time can be further improved, since no special code optimization was performed. A more detailed description of the news-on-demand prototype (hardware and software platform, implementation architecture, etc.) can be found in Hafid and Bochmann (1996).

3 The RRS

RRS is an adaptation scheme that tries to maintain, if possible, the initially agreed delay, jitter, and/or loss rate when one or more components of the configuration of interest failed to meet their commitment. While the principle of RRS can be used for any kind of configuration, we consider only linear configurations involving a single stream from the source to the sink.

3.1 General idea

The idea for RRS is to change, in response to a QoS violation, the amount of resources reserved by the components in the configuration in such a way that the end-to-end requirements will still be met. When a QoS violation occurs, the QoS agent of the overloaded component (which does not meet the initially agreed QoS) will ask the other components to reserve additional resources to compensate for the violation. This is done through the sending of a so-called *violation signal*. If the system components do not have enough resources to recover from the violation, a renegotiation is initiated with the applications/users.

The new resource configuration (the new distribution of QoS levels over the components of the configuration) remains in place until

(1) another QoS violation occurs, or
(2) the overloaded component recovers from the problem; if the overloaded component recovers from the problem, it may resume the initially agreed QoS and send a so-called *relaxation signal* to the other components in order to indicate that they may reduce their extra commitments.

The responsiveness of RRS is determined by the monitoring delay and the response time of the RRS algorithm, as explained in Sect. 2.4. The response time of the algorithm depends on whether the centralized, ring or hierarchical architecture is used, as discussed below.

Example of the operation of RRS. A user at *Client machine1* asks to communicate, with a given QoS, with another user at *Client machine2*, e.g., in audio-conferencing;

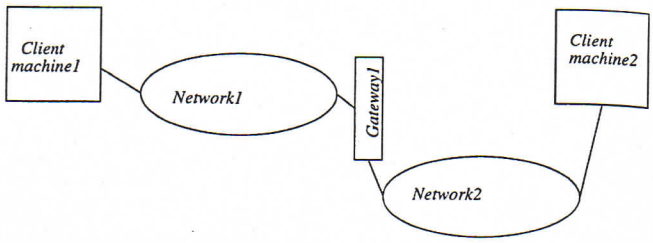

Fig. 5. An example of a distributed system

Table 1. Example of the RRS operation to deal with delay violation

	Network1 delay	*Network2* delay	other components cumulative delay	end-to-end delay
agreed delay	100 ms	100 ms	50 ms	250 ms
delay after violation	150 ms	100 ms	50 ms	300 ms
delay after the operation of RRS	150 ms	50 ms	50 ms	250 ms
delay after relaxation	100 ms	100 ms	50 ms	250 ms

the QoS manager makes use of the negotiation procedure (see Sect. 2.2), and finds a system configuration that might support the requested service. The configuration consists of *Client machine1*, *Network1*, *Gateway1*, *Network2*, and *Client machine2* (Fig. 5); each of these components commits to a certain level of QoS. During the communication session, *Network1* fails to meet the agreed delay, while *Network2* has a certain amount of resources unused.

Using RRS, a possible scenario to recover from the violation caused by *Network1* is that *Network1* asks *Network2* to reserve more resources to compensate for the violation (Table 1). At a later time, if *Network1* has enough resources to meet the initially agreed delay, then it will notify *Network2*, which might free the resources used for the previous compensation.

3.2 Protocols to implement the RRS

RRS allows the system to adapt to longer term load changes than DRS, and thus to reduce the end-to-end QoS violation over the lifetime of the application session without enormous overhead. Only one violation signal is issued for each detected problem. In order to support RRS, each participating component must realize the following three functions.

(1) *Violation detection.* The QoS agent of c_j must monitor the levels of QoS it is providing. Upon the detection of a QoS violation over a certain period, the QoS agent computes (estimates) a QoS level which would be sustainable in the future; traffic-forecasting techniques may be used to compute this QoS level (Somalingam 1996). Then, a *violation signal* is generated. It is worth noting that the violation cannot be recovered from if the amount of violation is higher than the end-to-end QoS (e.g., a component in the configuration failed to meet its agreed delay by 200 ms, and the end-to-end delay is 150 ms).
(2) *Resource renegotiation.* The QoS agent processes the signals it receives, e.g., *violation or relaxation signals.*

This processing consists of (1) committing to a higher level of QoS by some additional resources, if possible, or (2) deallocating additional resources.

(3) *Recovery detection (optional).* The QoS agent of c_j that initially issued a *violation signal* may monitor the current load of the component to check its capability to support the initially agreed level of QoS. Upon the detection of such a capability, a *relaxation signal* is generated; the *relaxation signal* contains the amount of the last violation that occurred at c_j.

We identified three types of protocols for implementing RRS, as described below: a centralized protocol, a ring protocol using an optimal contribution policy, and a ring protocol using an immediate contribution policy.

3.2.1 Centralized RRS protocol

Since it is the task of the QoS managers to manage QoS, the obvious choice would be to use the centralized QoS manager to react to QoS violations by reorganizing the available resources. The operation of the centralized protocol is based on interactions between the QoS manager and the QoS agents; when a QoS agent detects a QoS violation, it sends a *violation signal* to the QoS manager (Fig. 6). The QoS manager then checks the available resources of the components involved in the configuration; this is performed by sending *request_resource signals* to the QoS agents. Based on the results of this operation, the QoS manager then decides on a solution and informs the QoS agents (by sending *confirmation signals*) of the fact that they have to assign more or less resources to this particular application. If after some time the QoS agent determines that it can again support the previously agreed QoS, it sends a *relaxation signal* to the QoS manager. The QoS manager may decide to either continue with the currently operating configuration or to resort to the one previously agreed.

Description of the centralized protocol. For the sake of clarity, the presented protocol description concerns only the delay parameter; however, the same protocol applies for any other QoS parameters, e.g., loss rate and jitter, which have the additive concatenation property (see Sect. 2.1). The protocol described below can be extended to the general case involving several QoS parameters by replacing the scalar variables, such as MaximumQoS, RelaxationDegree, ViolationDegree, by vector variables, where each index of the vector corresponds to a particular QoS parameter. Similarly, the arithmetic operations on the scalar variables should be replaced by corresponding operations on vectors. The same applies for the ring protocols described in Sect. 3.2.2.

The QoS manager associates an identifier, which is called *ServiceId*, to each instance of service to be provided. Then, it associates with this identifier the characteristics of the service instance which are required for QoS management, such as the components involved in the support of the service and the QoS parameters initially negotiated. In a distributed system, a globally unique identifier for a service instance can be obtained by concatenating the identifier of the QoS manager (assumed to be globally unique) with a local identifier selected by the QoS manager.

Signals description

We define the following signals:

- *violation signal* (ServiceId, Id, ViolationDegree). It is sent by the QoS agent, identified by Id, to the QoS manager; it contains three parameters: (1) ServiceId, which is the identifier of the service instance for which the target QoS is violated, (2) Id, and (3) ViolationDegree, which indicates the amount of the violation; ViolationDegree is equal to the difference between the initially agreed QoS and the currently provided QoS.
- *request_resource signal* (ServiceId, Id). It is sent by the QoS manager to the QoS agent identified by Id.
- *QoS_available signal* (ServiceId, Id, MaximumQoS). It is sent by the QoS agent, identified by Id, to the QoS manager in response to a *request_resource signal*; it contains three parameters: (1) ServiceId, (2) Id, and (3) MaximumQoS: the maximum QoS which might be supported by the corresponding component for the configuration in question.
- *relaxation signal* (ServiceId, Id, RelaxationDegree). It is initially sent by the QoS agent, identified by Id, to the QoS manager; it contains three parameters: (1) ServiceId, (2) Id, and (3) RelaxationDegree, which is equal to the difference between the initially agreed QoS and the currently provided QoS.
- *confirmation signal* (ServiceId, Id, QoSToSupport). It is sent by the QoS manager to the QoS agent identified by Id; it contains three parameters: (1) ServiceId, (2) Id, and (3) QoSToSupport, which is the effective QoS which the corresponding component should support.

Description of the operation of a QoS agent

Variables description

We define the following variables:

- QoS is a QoS variable which indicates the level of QoS (target) to which a component c (in the configuration) has committed during the establishment phase.
- qos is a QoS variable which indicates, at a given time, the level of QoS which would be sustainable in the future by the component c.
- $self$ contains the identifier of the component in question.

Operation

- *When a QoS violation is detected:*
  the QoS agent sends a *violation signal (ServiceId, Self, ViolationDegree)* to the QoS manager, where ViolationDegree=$qos - QoS$;
- *When a recovery is detected:*
  the QoS agent performs the following operations:
  - reserves an extra amount of resources to support QoS instead of qos;
  - sends a *relaxation signal (ServiceId, Self, RelaxationDegree)* to the QoS manager, where RelaxationDegree=$qos - QoS$;
- *When a request_resource signal (ServiceId, Id) is received:*
  the QoS agent, identified by Id, performs the following operations:

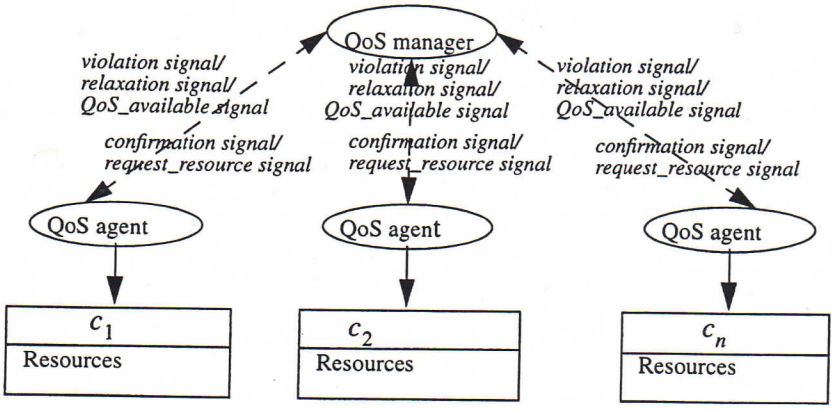

Fig. 6. RRS centralized protocol

- computes *MaximumQoS*, the maximum QoS that could be supported, assuming that all remaining resources could be used to support the requested service; this operation depends on the nature of the component, e.g., gateway or LAN, the management software, e.g., scheduling mechanisms, and the characteristics of the service (stream) in question. An example of the computation of MaximumQoS can be found in Ferrari and Verma (1990);
- sends a *QoS_available signal (ServiceId, Id, MaximumQoS)* to the QoS manager;
- waits for receipt of the corresponding confirmation signal;

– *When a confirmation signal (ServiceId, Id, QoSToSupport) is received:*
the QoS agent, identified by Id, reserves the necessary component resources, such as buffer. CPU slots and bandwidth, in order to support the QoS *QoSToSupport* for the remainder of the session.

– *When a relaxation signal (ServiceId, Id, RelaxationDegree) is received:*
the QoS agent, identified by Id, deallocates a certain amount of resources to relax the currently provided QoS, qos, by *RelaxationDegree*.

Description of the operation of the QoS manager

Variables description

We define the following variables:

- $endtoendQoS$ is a QoS variable which indicates the agreed (target) end-to-end QoS to be supported by the configuration.
- $NewendtoendQoS$ is a QoS variable;

Operation

– *When a violation signal (ServiceId, Id) is received:*
  - the QoS manager sends a *request_resource signal (ServiceId, $Id^{\dagger}$)* to each QoS agent (except the QoS agent identified by Id), identified by $Id^{\dagger}$, in the configuration identified by *ServiceId*;

– *When all QoS_available signals (ServiceId, Id, MaximumQoS) have been received from the QoS agents:*
the QoS manager performs the following operations:

- computes the "sum", $NewendtoendQoS$, of *MaximumQoS* using the formulae described in Sect. 2.1; ($NewendtoendQoS = \sum^{n} MaximumQoS$).
- if $NewendtoendQoS$ is "better" or equal to $endtoendQoS$, then, for each component (except the overloaded component), identified by Id, in the configuration identified by *ServiceId*, the QoS manager performs the following operations:
  - computes the QoS *QoSToSupport* that the component should support; this operation depends on the contribution policies used by the QoS manager (see below for more details);
  - sends a *confirmation signal (ServiceId, Id, QoSToSupport)* to the component;
- if $endtoendQoS$ is "better" than $NewendtoendQoS$, then the QoS violation cannot be recovered and the QoS manager notifies the user/application who may initiate a renegotiation to degrade the initially agreed QoS, abort the current service session, or ignore the violation.

– *When a relaxation signal (ServiceId, Id, RelaxationDegree) is received:*
the QoS manager sends *relaxation signals (ServiceId, Id, RelaxationDegree′)* to the QoS agents that participated to compensate for the previous violation or simply to certain QoS agents that may be in difficulty to meet their commitments; the sum of the values of RelaxationDegree′ is equal to *RelaxationDegree*.

Contributions policies used by the QoS manager

When the QoS manager has received the *available_resource signals (ServiceId, Id, MaximumQoS)* from all the QoS agents in the configuration, it computes for each agent the value of *QoSToSupport* (smaller than *MaximumQoS*) which the component should support in order to meet the end-to-end requirements. This means that the end-to-end QoS requirements are allocated to individual components. For example, given the fact that the end-to-end delay is the sum of the component delays (see Sect. 2.1), Table 1 shows one possible QoS allocation in which Network1 (respectively Network2) is required to support a delay of 100 ms (respectively of 100 ms), and the other components are required to support a delay of 50 ms; however, to support an end-to-end delay of 250 ms, several QoS allocations are possible. The QoS manager may use a certain number of policies, which we call

contribution policies, to select a *particular QoS allocation* for the different components; these policies may be used to maximize the system-supportable load, to minimize the cost to the user, to minimize the time required to compensate for a violation, to favor certain components (with high reliability), and so on. A detailed study of policies which maximize the system-supportable load can be found in Nagarajan (1993).

3.2.2 RRS ring protocol using an optimal contribution policy

The operation of the RRS ring protocol using an optimal contribution policy is similar (to some extent) to the operation of the centralized RRS protocol. They differ mainly in two aspects: (1) the RRS ring protocol is based on direct interactions between the QoS agents of the components of the linear configuration (Fig. 7), in opposition to the centralized protocol, which is based on interactions between the QoS manager and the QoS agents (Fig. 6); and (2) the functions performed by the QoS manager in the c ase of the centralized protocol are performed by the QoS agent of the overloaded component in the case of the RRS ring protocol.

In the following, we present a more detailed description of this RRS ring protocol. When a QoS agent detects a QoS violation, it sends a *violation signal* along with the violation degree to the neighboring component (Fig. 7); the latter computes the maximum-level QoS, MaximumQoS, it is able to provide for the service in question, and sends a *violation signal* along with this information to its neighboring component. When the QoS agent of the overloaded component finally receives the *violation signal*, which has passed around the ring and includes the maximum level of QoS each component is able to support, it checks whether the "sum" of these levels of QoS is equal or "better" than the initially agreed (end-to-end) QoS. If the response is yes, the QoS agent decides on the level of QoS each component should support, based on some optimization factors (Nagarajan 1993), puts this information in a *confirmation signal*, and sends the signal over the ring to the other QoS agents; upon receipt of the signal, the QoS agent should reserve the resources to satisfy the levels of QoS specified in the signal. If the "sum" is worse than the initially agreed QoS, the QoS agent (of the overloaded component) notifies the user/application who may initiate a renegotiation to degrade the agreed QoS, abort the current service session, or ignore the violation. If some time after the adaptation the overloaded QoS agent determines that it can again support the previously agreed QoS, it sends a *relaxation signal* to the other components on the ring. Upon receipt of this signal, the QoS agents (who participated in solving the previous violation) may deallocate the resources which were used to solve the previous violation.

Description of the ring protocol using an optimal contribution policy

Signals description

We define the following signals:

- *violation signal* (ServiceId, Id, ListOfMaximumQoS): it is initially generated by the QoS agent, identified by Id, who detects a QoS violation; ListOfMaximumQoS is a list of tuplets each containing the identifier of a component and the maximum QoS the component is able to support for the service in question.
- *confirmation signal* (ServiceId, Id, ListOfQoSToSupport): it is initially sent by the QoS agent, identified by Id, who detects a QoS violation; ListofQoSToSupport is a list of tuplets each containing the identifier of a component and the QoS the component should support for the service in question.
- *relaxation signal* (ServiceId, Id, RelaxationDegree): it is initially sent by the QoS agent, identified by Id, who previously failed to meet its initially agreed commitments.

Description of the operation of a QoS agent

Variables description

In addition to the variables defined for the centralized protocol we define the following variable:

- $Next$ contains the identifier of the component next in the linear configuration (or of the first component, in the case of the last component in the configuration).

Operation

- *When a QoS violation is detected:*
  the QoS agent who detects the violation sends a *violation signal* (serviceId, $Self$, []) to $Next$; ([] means an empty list).
- *When a recovery is detected:*
  the QoS agent performs the following operations:
  - reserves an extra amount of resources to support QoS instead of qos;
  - sends a *relaxation signal (ServiceId, Self, RelaxationDegree)* to its neighboring component, where RelaxationDegree=$qos - QoS$;
- *When a violation signal (ServiceId, Id, ListOfMaximumQoS) is received:*
  if $Id = Self$ (which means that the QoS agent receives the *violation signal* it initially sent), then the QoS agent performs the following operations:
  - computes the "sum", $NewendtoendQoS$, of the MaximumQoS values contained in *ListOfMaximumQoS*, using the formulae described in Sect. 2.1;
  - if $NewendtoendQoS$ is "better" or equal to QoS, then the QoS agent performs the following operations:
    - for each component, the QoS agent computes the QoS *QoSToSupport* that the component should support, and puts the tuplet (identifier of the component, *QoSTosupport*) into *ListOfQoSToSupport* (which is initially empty); this operation is based on some contribution policy, as described at the end of Sect. 3.2.1;
    - sends a *confirmation signal (ServiceId, Self, ListOfMaximumQoS)* to its neighboring component, $Next$;
  - if QoS is "better" than $NewendtoendQoS$, then the QoS violation cannot be recovered and the QoS agent

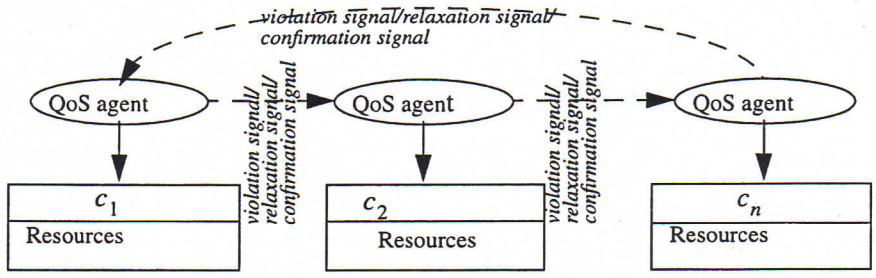

Fig. 7. RRS ring protocol

notifies the user/application who may initiate a renegotiation to degrade the initially agreed QoS, abort the current service session, or ignore the violation;

Else ($Id \neq Self$)

the QoS agent performs the following operations:

- computes the maximum QoS, *MaximumQoS*, assuming that all remaining resources could be used to support the requested service;
- puts the tuplet (*MaximumQoS*, $Self$) at the end of *ListOfMaximumQoS*;
- sends *violation signal (ServiceId, Id, ListOfMaximumQoS)* to its neighboring component, $Next$;
- waits for receipt of the corresponding confirmation signal;

Endif

- *When a confirmation signal (ServiceId, Id, ListOfQoSToSupport) is received:*
  if Id = $Self$, then the QoS agent discards the confirmation signal; otherwise, it reads the tuplet (*QoSToSupport*, $Self$) from *ListOfQoSToSupport*, reserves a certain amount of its resources to support *QoSToSupport* for the remainder of the session, and sends a *confirmation signal (ServiceId, Self, ListOfQoSToSupport)* to its neighboring component, $Next$;
- *When a relaxation signal (ServiceId, Id, RelaxationDegree) is received:*
  if Id = $Self$, then the QoS agent discards the relaxation signal; otherwise, it deallocates a certain amount of its resources to relax the currently provided QoS by (a fraction of) *RelaxationDegree*, and sends a *relaxation signal (ServiceId, Self, remaining of RelaxationDegree)* to its neighboring component, $Next$;

3.2.3 RRS ring protocol using an immediate contribution policy

The RRS ring protocol using an immediate contribution policy is also based on interactions between the QoS agents of the components of the linear configuration in question; when a QoS agent detects a QoS violation, it sends a *violation signal* to the neighboring component (Fig. 7); its QoS agent reserves resources, if available, to completely compensate the violation. The result of this operation may be (1) a success: the violation problem is solved, (2) a failure: the component cannot reserve any additional resources, that is, the component is at its maximum utilization, or (3) a failure with an offer: the component can reserve resources to compensate only a fraction of the violation. In any case, the component should send a *violation signal* that contains the violation degree (e.g., equal to zero in case (1) above) that remains to be absorbed to its neighboring component. When the QoS agent of the overloaded component receives the *violation signal* it initially sent, it checks the violation degree the signal contains. If it is different from zero, the QoS agent notifies the user/application; otherwise, it sends a *confirmation signal* towards the components to make the reservation of the extra-allocated resources effective.

Description of ring protocol using an immediate contribution policy

Signals description

We define the following signals:

- *violation signal* (ServiceId, Id, ViolationDegree): it is initially sent by the QoS agent, identified by Id, who detects a QoS violation;
- *confirmation signal* (ServiceId, Id): it is initially sent by the QoS agent, identified by Id, who detects a QoS violation;
- *relaxation signal* (ServiceId, Id, RelaxationDegree): it is initially sent by the QoS agent, identified by Id, who previously failed to meet its initially agreed commitments;

Description of the operation of a QoS agent

Variables description

Same as for the RRS ring protocol using optimal contribution policy.

Operation

- *When a QoS violation is detected:*
  the QoS agent (who detects the violation) sends a *violation signal (ServiceId, Self, ViolationDegree)*, where *ViolationDegree*=$qos - QoS$, to its neighboring component, $Next$;
- *When a recovery is detected:*
  the operation of the QoS agent is similar to the corresponding operation described in RRS ring protocol using an optimal contribution policy.
- *When a violation signal (ServiceId, Id, ViolationDegree) is received:*
  if Id = $Self$, then the QoS agent checks whether *ViolationDegree* is equal to zero; if this is the case, then it will send a *confirmation signal (ServiceId, Self)* to its neighboring component, $Next$; otherwise the QoS agent notifies the user/application;

  Else ($Id \neq Self$)

  if (*ViolationDegree*=0) then

  the QoS sends a *violation signal (ServiceId, Id,*

0) to its neighboring component, $Next$;
else (*ViolationDegree*$\neq$ 0)
the QoS agent performs the following operations:
- computes the maximum QoS, MaximumQoS, assuming that all remaining resources could be used to support the requested service;
- *ViolationDegree*=*ViolationDegree*- (*qos*-MaximumQoS);
- if (*ViolationDegree*¡0) then
  - *ViolationDegree:*=0;
  - reserves temporarily the resources required to support *qos*-ViolationDegree;

  else
  - reserves temporarily the resources required to support MaximumQoS;

  endif
  - sends a *violation signal (ServiceId, Id, ViolationDegree)* to $Next$;
  - waits for receipt of the corresponding confirmation signal;

  Endif

– *When a confirmation signal (ServiceId, Id) is received:*
if $Id = Self$, then the QoS agent discards the confirmation signal; otherwise it makes effective reservation of the resources (reserved temporarily to compensate for the corresponding violation) for the remaining duration of the service session, and sends a *confirmation signal (ServiceId, Id)* to $Next$;
– *When a relaxation signal (ServiceId, Id, RelaxationDegree) is received:*
the operation of the QoS agent is similar to the corresponding operation described in the RRS ring protocol using an optimal contribution policy.

It is worth noting that the RRS ring protocol using an immediate contribution policy may be realized without the *confirmation signal*. In this case, when a QoS agent receives a *violation signal*, it reserves the resources immediately and uses them for the service in question without waiting for the confirmation signal.

We note that, to each violation signal, there corresponds exactly one confirmation signal (for all protocols described in this section); the two signals are identified by the identifier of the QoS agent which generates the violation signal and by the identifier of the service instance in question. The operation of the RRS protocol can therefore be executed independently for each violation signal that may be generated within the system. The only interference that may occur is the following. When a given agent receives a second violation signal when an earlier violation signal has not yet been confirmed, it may have allocated most of its free resources to the first violation signal and have no additional resources available for the second signal. In this case, it will pass the second violation signal on to the next neighbour without any additional contribution, although most of the temporarily allocated resources will probably be freed when the confirmation for the first violation signal arrives, indicating how many resources should actually be reserved. In order to improve the adaptation performance of the system, it may be advantageous to limit the amount of resources temporarily allocated for a single violation signal. Also, the performance of the algorithm could be optimized in order to reduce the chance that a second violation signal is received during the processing of an earlier one.

3.3 Operation of the RRS in a hierarchical environment

RRS can easily operate in a hierarchical multi-domain environment as shown in Fig. 8; a combination of the protocols described above may be used to recover from QoS violations in such an environment. Examples of these combinations are: ring protocols (or the centralized protocol) at lower levels of the hierarchy and a centralized protocol (or ring protocols) at higher levels, ring protocols at all levels, and the centralized protocol at all levels.

As an example, we describe in the following the operation of RRS in a three-level hierarchical environment which consists of four domains; we use the centralized protocol at the lower levels and the ring protocol using the immediate contribution policy at the higher level (Fig. 8). Let us assume that (1) the components involved in the provision for the requested service are c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, and c_{10} (Fig. 8), and (2) during the lifetime of the session, c_2 detects a local QoS violation. Using the RRS, the following operations are executed: (1) c_2 sends a *violation signal* to QoS manager-1, (2) QoS manager-1 uses the centralized protocol to ask for the help of c_1 to compensate for the violation (a fraction or the whole violation may be compensated by c_1), (3) if the violation cannot be solved, instead of notifying the user/application, QoS manager-1 sends a violation signal, with the remaining violation degree, to QoS manager-2. The latter uses the centralized protocol to recover, if possible, from the violation (the violation degree is included in the *violation signal*), and then sends a *violation signal*, with the remaining violation degree, to QoS manager-3. When QoS manager-1 receives the corresponding *violation signal*, it notifies the user/application if the violation degree (contained in the signal) is different from zero (Fig. 8).

More generally, a domain is seen by our scheme as a single component, and its QoS manager as a QoS agent. At the higher level, the operation of the RRS is as shown in Fig. 9. When a QoS manager receives a *violation signal*, it must compute the maximum contribution it can support, puts this information in the *violation signal* and sends it to its neighboring QoS manager. The computation of the maximum contribution is delegated to the QoS agents of the domain's components of interest using the centralized protocol.

Since RRS is a *high-level scheme* (Fig. 9), it may also integrate existing adaptation schemes, such as the GAS (Parris et al. 1994). In fact, different domains may implement different adaptation schemes, and RRS may still be useful for inter-domain coordination. For example, upon receipt of a *violation signal*, QoS manager-2 may use the GAS to establish a better route within its domain, in order to recover from the violation.

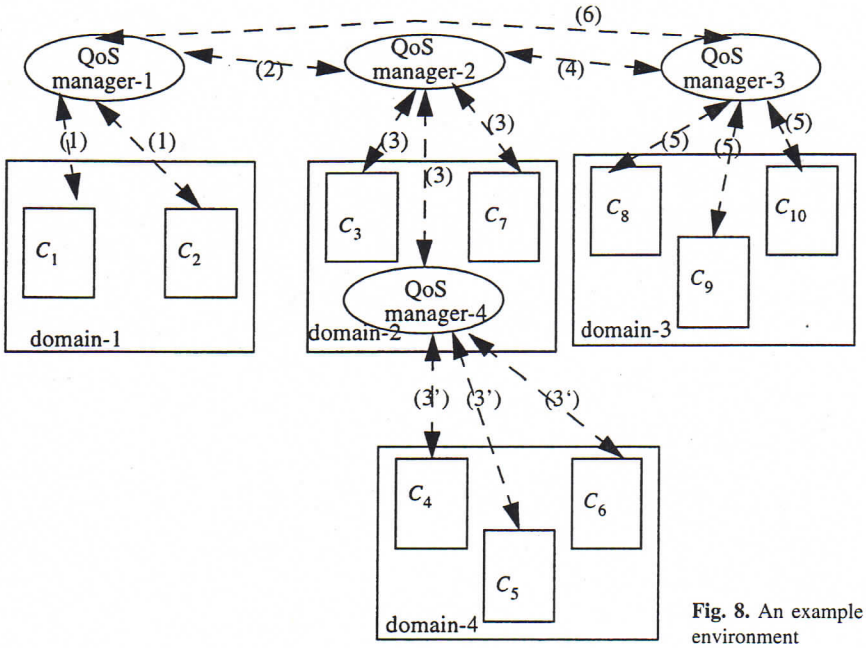

Fig. 8. An example illustrating the operation of the RRS in a hierarchical environment

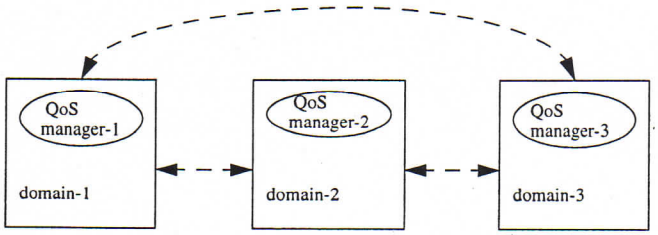

Fig. 9. An example illustrating the operation, at a higher level, of the RRS ring protocol in a hierarchical system

4 DRS

The basic idea of RRS, as explained above, is to request increased QoS commitments from the other system components when a given component has some difficulty and cannot meet its agreed commitments. RRS is initiated by the component in difficulty. Usually, the difficulty will be detected only after a certain measurement period (see Sect. 3), during which the actually provided QoS was found to be inadequate. During this period, before RRS has been activated, there is therefore a good chance that the end-to-end QoS, as seen by the user, could be below the agreed value.

If the inadequacy of the QoS provided by a component could be detected immediately, it would be possible to avoid any QoS degradation as seen by the user. This is the case for delay. The purpose of the DRS, discussed in this section, is to compensate for a delay violation for each transmitted data unit. If the measured delay value of a given data unit is above the commitment, a *violation signal* is sent together with the data unit in question, and the next component in the chain may apply a higher QoS, if available, to the same data unit, in order to compensate the problem encountered earlier. A similar approach involving only three components (the source, the sink, and the transport system) was proposed in Khoumsi et al. (1995) in the context of protocol synthesis for real-time applications.

DRS can be used to react only to delay violations; it is suitable for real-time and multimedia applications with stringent temporal requirements. An example is the telerobotics application described in Nahrstedt (1995), where the sensory data has strict constraints on end-to-end delay, but relatively low throughput demands.

4.1 Example of operation

Let us consider an established configuration CF=$c_1 - c_2 - c_3 - c_4$ to support a requested service (Fig. 10). The end-to-end delay required to support the requested service is QoS. The component c_1 (c_2, c_3, and c_4) committed during the establishment phase to support QoS_1 (QoS_2, QoS_3 and QoS_4). Let us assume that c_1 did not meet its commitment for a given data unit by violating its agreed level of delay by RC=VD1. This information is sent to c_2, together with the data unit, in order to try to compensate for the violation. Unfortunately, c_2 has not enough resources to recover from the violation. However, it reduces the violation by *contribution*=VD1-VD2 using its available resources. RC – *contribution*=VD2 is sent to c_3. Fortunately, c_3 has enough resources and recovers from the delay violation. c_4 meets its commitment, and hence the end-to-end delay guarantee is provided, even though c_1 failed to support its agreed level of delay.

4.2 A DRS protocol

The protocol defined below for DRS is similar, to some extent, to the RRS ring protocol using the immediate contribution policy. More specifically, the DRS protocol has the following characteristics: (1) the exchanges of messages between the components are more frequent; more precisely, information is exchanged between the components for each service data unit; (2) only "downstream" components may

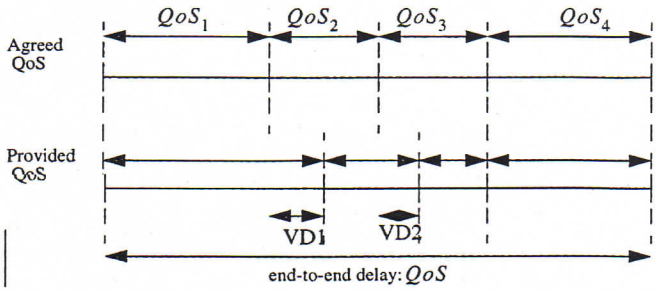

Fig. 10. An example of the operation of the DRS

contribute to solve the violation, (c_n does not send *violation* or *relaxation signals* to c_1); and (3) the existence of synchronized clocks is assumed for easily determining, through time stamps, the delay of each data unit.

Description of the operation of the protocol

Signals description

We extend the *data unit*, e.g., Network Protocol Data Unit (NPDU), to include the following field:

- ViolationDegree: indicates the difference between the delay encountered by the data unit and the agreed delay for a given component;
- InputTimeStamp: indicates the time when a given component receives the data unit;

Description of the behavior of a component

Variables description

We define the following variables:

- $AgreedDelay$ corresponds to the (target) delay the component committed to provide, during the establishment phase.
- $Next$ contains the identifier of the neighboring component in the configuration in question.

Operation

Each time the component, receives a data unit, it performs the following operations:

(1) reads the current time, CurrentTime, from its clock;
(2) sets InputTimeStamp to CurrentTime;
(3) if ViolationDegree > 0 then
  - The QoS agent processes the data unit, e.g., a decoder decodes the data unit and a network transports the data unit, while doing its best (depending on its available resources) to satisfy a delay of $AgreedDelay$ = $AgrreedDelay$ - ViolationDegree;

  else (ViolationDegree ≤ 0)
  - The QoS agent processes the data unit; in this case, the QoS agent may relax its commitment by ViolationDegree (or fraction of ViolationDegree), that is, it has only to satisfy a delay of $AgreedDelay$ = $AgrreedDelay$ -ViolationDegree; (this operation is recommended when the QoS agent has difficulties to meet its initially agreed delay)

  endif
(4) reads the current time, CurrentTime, from its clock;
(5) ViolationDegree=(CurrentTime-InputTimeStamp)-$AgreedDelay$;
(6) sends the (processed) data unit, along with the updated ViolationDegree, to its neighboring component, $Next$;

If we assume that each component in the configuration knows the allocated partial delay, the operation of the protocol will be considerably simplified. The partial delay, called PartialDelay, of a component relative to a given stream is defined as the elapsed time from the moment a data unit of the stream is generated (at the source) to the moment the data unit leaves the component. In this case, the protocol can be changed as follows: for each data unit the time stamp, TimeStamp, is set only once at the source; thus, the operations (1) and (2) of the protocol are not needed, and the operation (5) will be replaced by (5′) ViolationDegree=CurrentTime-TimeStamp-PartialDelay. In the example presented in Sect. 4.1, the allocated partial delays for c_1, c_2, c_3 and c_4 are QoS_1, QoS_1 + QoS_2, QoS_1+QoS_2+QoS_3, and QoS_1+QoS_2+QoS_3+QoS_4 (which is equal to the end-to-end delay), respectively.

Concerning the performance of DRS, it is related to the performance of reading the real-time clock twice for each data unit. We assume that the system components provide an efficient clock-reading operation.

5 Conclusion

The issues of QoS adaptation have only been partially addressed in the literature. Most of the existing approaches have one or several of the following characteristics: (1) they are restricted to the communication sub-system; (2) they react only after the occurrence of an end-to-end QoS violation; and (3) they react to a QoS violation by renegotiating a degraded QoS, and thus they are restricted to applications that can accept a varying QoS. In this paper, we presented an approach which allows to recover *automatically* from QoS violations, and *only require user/application intervention* (as existing approaches) when the system has not enough resources to support the current load. The proposed approach consists of *high-level interactions* between management entities, and thus may be applied to an arbitrary system (several domains) with different adaptation and reservation schemes. A number of instantiations of the proposed schemes may be developed based on the characteristics of the environment (systems and applications).

We have presented three schemes for QoS adaptation: the component resource reconfiguration (CRS), the resource reconfiguration scheme (RRS), and the delay recovery scheme (DRS). CRS tries to recover from any QoS violation by selecting one or more alternate components and by performing a user-transparent transition from the primary components to the alternative ones. RRS tries to recover from QoS violations, specifically delay, jitter, and loss rate violations, by changing the distribution of QoS levels that each component will support. It integrates, in a suitable way, the two possible approaches that can be used to deal with QoS violation: to maintain, if possible, the initially agreed QoS, and otherwise to initiate a renegotiation with the user/application. DRS tries to recover from delay violations of specific components for each individual data unit that is transmitted. The

role of DRS is to avoid, if possible, end-to-end delay violation, but it introduces substantial overhead.

The proposed schemes, especially CRS, are able to support a graceful degradation. Indeed, when the QoS manager fails to find an alternate configuration that supports the initially agreed QoS, it may select a configuration that supports a lower QoS in a graceful way, e.g., continue playing black/white video instead of the initial color video.

A scheme that supports the characteristics of all three schemes RDS, RRS and CRS may be more efficient. As an example, for the delay parameter, the following scenario may be used:

(1) DRS is used to maintain end-to-end delay of each data unit.
(2) If a component is in difficulty to supports its delay requirements, RRS is used to reconfigure the resource distribution of the components in a way to satisfy end-to-end delay requirements. The delay considered is the delay averaged over some measurement interval. This operation is executed in parallel with operation (1).
(3) If operation (2) does not succeed, CRS may be used to find an alternate configuration able to support end-to-end requirements. Then, the QoS manager performs a transition from the current configuration to the new one.

Acknowledgements. We would like to thank R. Velthuys from IBM Toronto, and A. Khoumsi from University of Montreal for fruitful discussions on a draft of this paper. This work was supported by a grant from the Canadian Institute for Telecommunication Research (CITR), under the Networks of Centres of Excellence Program of the Canadian Government.

References

Bochmann G von, Kerherve B, Hafid A, Dini, Pons A (1996) Architectural Design of Adaptive Distributed Multimedia Systems. In: Proceedings of the IEEE International Workshop in Distributed Multimedia Systems Design, Berlin, Germany

Bochmann G von, Hafid A (1997) Some Principles for QoS Management. Distributed Syst Eng J 4(1): 16–27

Danthine A (1992) OSI95: High-Performance Protocol with Multimedia Support on HSLANs and B-ISDN. In: Proceedings of the 3rd Joint European Networking Conference, Innsbruck, Austria

Ferrari D, Verma D (1990) A scheme for Real-Time Channel Establishment in Wide-Area Networks. IEEE J 8 (3)

Ferrari D, Banerjea A, Zhang H (1992) Network Support for Multimedia. Technical report 92–072, International Computer Science Institute, Berkeley, Calif.

Gilge M, Gussella R (1991) Motion Video Coding for Packet Switching Networks – An Integrated Approach. Proc SPIE Visual Commun Image Process

Hafid A (1995) A Hierarchical Negotiation for Distributed Multimedia Applications in a Multi-Domain Environment. In: Proceedings of the Second International Workshop on Protocols for Multimedia Systems, Salzburg, Austria, pp 325–337

Hafid A, Bochmann G von (1996) Quality-of-Service Negotiation in News-on-Demand Systems: An Implementation. In: Proceedings of the Third International Workshop on Protocols for Multimedia Systems, Madrid, Spain, pp 221–240

Hafid A, Bochmann G von, Kerherve B (1996) A QoS Negotiation Procedure for Distributed multimedia Presentational Applications. In: Proceedings of the Fifth IEEE International High-Speed Distributed Computing (HPDC-5), Syracuse, New York

Hanko J, Kuerner E, Northcut D, Wall G (1991) Workstation Support for Time-Critical Applications. In: Proceedings of the Second International Workshop Heidelberg, Germany

Khoumsi A, Bochmann G von, Dssouli R (1994) On Specifying Services and Synthesizing Protocols for Real-Time Applications. In: Proceedings of the Conference on Protocol Specification, Testing and Verification (PSTV), Vancouver, Canada, pp 185–200

Nagarajan R (1993) Quality-of-Service Issues in High-Speed Networks. Ph.D. Thesis, University of Massachusetts, Cambridge, Mass.

Nahrstedt K (1995) An Architecture for End-to-End Quality-of-Service Provision and its Experimental Validation. Ph.D. Thesis, University of Pennsylvania

Parris C, Ventre G, Zhang H (1994) Dynamic Management of Guaranteed-Performance Multimedia Connections. Multimedia Syst

Seneviratne A, Cho H (1995) Quality of Service in Distributed Multimedia Systems. In: Proceedings of International Conference on Multimedia Networking (IEEE Computer Society), Aizu-Wakamatsu, Fukushima, Japan

Sloman S, Moffet J (1989) Domain Management for Distributed Systems. IFIP TC 6/WQ 6.6 (Integrated Network Management I, IFIP 1989)

Somalingam R (1996) Network Performance Monitoring for Multimedia Networks. Masters Thesis, School of Computer Science, McGill University, Montreal, Canada

Steinmetz R (1990) Synchronization properties on Multimedia Systems. IEEE J

Tawbi W, Horlait E (1994) Expression and Management of QoS in Multimedia Communication Systems. Ann Telecommun

Tobe Y, Tokuda H, Chou S, Moura J (1992) QoS control in ARTS FDDI continuous Media Communications. In: Proceedings of ACM SIGCOMM 92

Topolcic C (1990) Experimental Internet Stream Protocol: Version 2 (ST-II), Internet RFC 1190

Wong J, Lyons K, Velthuys R, Bochmann G von, Dubois E, Georganas N, Neufeld G, Ozsu T, Brinskelle J, Evans D, Hafid A, Hutchinson N, Inglinski P, Kerherve B, Lamont L, Makaroff D, Szafron D (1997) Enabling Technology for Distributed Multimedia Applications. IBM Syst J 36(4): 489–507

Yin N, Hluchyi M (1991) A Dynamic Rate Control Mechanism for Integrated Networks. In: Proceedings of INFOCOM'91

DR. ABDELHAKIM HAFID is Assistant professor at the Electrical & Computer engineering/Computer science Depts. (a joint appointment), University of western Ontario, and Research Director of the Advanced Communication Engineering Centre (venture established by UWO, Bay Networks, Bell Canada); he is also an Adjunct Professor at University of Montreal, Department of computer Science. He received his Masters and Ph.D. degrees in computer science from University of Montreal on quality of service management for distributed multimedia applications in 1993 and 1996, respectively. From 1996 to 1997 he was a Researcher Staff Member at the Computer Research Institute of Montreal (CRIM), Telecommunications and Distributed Systems Division, working in the area of distributed multimedia applications. From 1993 to 1994 he was visiting scientist at GMD-FOKUS, Systems Engineering and Methods group, Berlin, Germany working in the area of high speed protocols testing. His current research interests are in Internet and multimedia networking.

GREGOR V. BOCHMANN has been a professor at the University of Montreal since 1972 and holds the Hewlett-Packard-NSERC-CITI chair of industrial research on communication protocols. He is also one of the scientific directors of the Centre de Recherche Informatique de Montreal (CRIM). He is a Fellow of the IEEE and ACM. He has worked in the areas of programming languages, compiler design, communication protocols, and software engineering, and has published many papers and some books in these areas. He has also been ac tively involved in the standardization of formal description techniques for OSI communication protocols and services. From 1977 to 1978 he was a visiting professor at the Ecole Polytechnique Fédérale, Lausanne, Switzerland. From 1979 to 1980 he was a visiting professor in the Computer Systems Laboratory, Stanford University, California. From 1986 to 1987 he was a visiting researcher at Siemens, Munich. His present work is aimed at methodologies for the design, implementation and testing of communication protocols and distributed systems. Ongoing projects include applications to high-speed protocols, distributed systems management and quality-of-service negotiation for distributed multimedia applications.