An Architecture for Dynamic Multimedia QoS Control

João Bom[2], Paulo Marques[2], Miguel Correia[3], Paulo Pinto[1,2]

[1]Instituto Superior Técnico, Lisboa
[2]INESC, R. Alves Redol, 9 P-1000 Lisboa, Portugal
[3]Universidade de Lisboa, Faculdade de Ciências, Campo Grande, Bloco C5, Piso 1, 1700 Lisboa
{joao.bom,paulo.marques,paulo.pinto}@inesc.pt mpc@di.fc.ul.pt

Abstract: The execution of repetitive tasks related with the handling of continuous media is something that should be detached as much as possible from multimedia applications. However, applications should have control over the most significant aspects of the process. This paper presents a framework to enable applications to exercise this control in terms of QoS concepts. Instead of defining low-level QoS parameters that might be meaningless to users, a sequence of QoS scenarios is provided by the applications to a QoS Manager to enable it to decide which is the best option at any moment. Based on this information the QoS Manager dynamically adapts the system to the current conditions on the network and at the end machines. Some experimental results of the algorithm are presented.

Keywords: Dynamic QoS Control; User Application; Practical Experiences

1. Introduction

The subject of quality of service (QoS) for distributed multimedia applications is rather complex due to various aspects involved. The main idea is that multimedia applications are very resource demanding and ideally could grab as much machine power or network bandwidth as the user could get. In practice, technological limitations or economic reasons impose a limit at some level. The sensible way to approach the challenge of building these applications is to try to get the most out of the realistic conditions where they operate. This best effort approach should be, as much as possible, independent from each specific media and technology. In this way, applications will evolve gracefully with the improvement of the technology. This approach can also be used as an adaptation mechanism to different conditions of operation.

This paper is concerned with multimedia applications over broadband networks, and with the control of the QoS when the dynamic conditions of the connection can vary substantially. A basic approach to QoS control when the network conditions are fairly stable, taking into account intramedia and intermedia synchronization for continuous streams, was performed earlier [6].

The framework proposed in this paper is based on two main prerequisites:

a) the distinction between media specific adaptation techniques to change QoS and a general QoS control algorithm; and
b) the mapping from a general QoS description at user level to lower level QoS concepts (dependent on the media) in order to simplify user control over the connections.

Consequently, the paper focus largely on the interaction between the application and the QoS algorithm, defining the main entities involved and the media independent data structures they both work with.

The algorithm interacts with the application in terms of QoS parameters meaningful to the programmer. It allows the application to define its own policies of QoS control, and whether it wants to have higher or lower control over the QoS. Applications provide their range of acceptable operating conditions together with their encoding dependent solutions to the variations of these conditions, in order to enable the algorithm to exercise its control autonomously (as long as it stays within the range).

The framework makes a clear distinction between data and control (as RTP also does) - multimedia data is captured and sent (or received and presented) directly to (or from) the transport protocol by a "media device manager", freeing the application from the complexity of such tasks.

One factor that can introduce dynamic operating conditions is the use of ATM technology. Instead of being limited to a connection traffic contract, better quality of service could be obtained by exploiting the statistical nature of the network. This is true as long as the application could handle most of the situations it can incur by using non-guaranteed conditions. Most of the Call Admission Control algorithms are conservative [1] and some network resources can be used at user discretion. The QoS algorithm works in an integrated way with the transport management algorithm to adapt the application and take advantage of these spare resources.

Another novel part in our proposal is that the algorithm takes into account the operating conditions on both machines in its control loop, and not only the network situation. This is relevant as the increase of bandwidth available in the network many times puts the processing restrictions in the machines.

2. Network Considerations

Traditional transport protocols, such as TCP, have already too many assumptions in terms of the semantics of the data communication or network behaviour. For instance, the flow control is not suitable for continuous media because missing data can be tolerated but retransmissions or extra delays cannot; the congestion control assumes congestion rather than errors when a segment does not arrive. The use of abstract portions of data (segments) with little relevance to the application makes the overall situation worse.

New protocols, such as RTP [13], overcome most of the previous limitations by the use of application level framing (ALF), the integration of their processing with the application's own processing (integrated layer processing, ILP) [5], and the use of profiles for the control algorithms that are more suitable to the application's requirements. Moreover, RTP has an associated control protocol, RTCP, to assess network conditions but does not impose any pre-defined algorithm to work on control data. In this paper, it is not so important to highlight the concrete case of our choice of RTP but to consider these abstract features of the new protocols (some extensions were even felt as being needed in the RTCP packets).

A natural way to use ATM networks with a best-effort service is to choose the Available Bit Rate (ABR) class of service. However, there are some reasons why this is not suitable for multimedia: the ABR was designed to serve data applications that are able to control spare bandwidth. They can adapt to varying conditions on the network but it is important that cell loss must be as low as possible (data applications are rather sensitive to cell loss). This is not a requirement to multimedia applications. ABR has its own control cycle (to prevent losses) which is based on multimedia abstract concepts, such as cells (the same problem pointed out above for traditional transport protocols).

The sensible way is then to use the Variable Bit Rate (VBR) class of service, and use priorities to work over contract values. The application sets a lower limit of bandwidth for the connection, below which it is useless to maintain the interaction. Then tries to raise the quality by using CLP=1 cells as much as the network can handle. We assume that the UPC/NPC mechanism uses cell tagging until the Peak Cell Rate (PCR) value is reached and discards cells above this rate. Therefore, a video connection, for instance, should set the PCR in accordance to the frame length, but can set the Sustainable Cell Rate (SCR) to a value lower than the actual value it intends to use (just above the minimum QoS level, for instance).

3. System Architecture

QoS control is a complex issue that should not have to be handled explicitly every time an application is programmed. The architecture presented here performs most of the QoS related operations. The objective is to create an environment where applications using media already installed in the system are programmed in a very straightforward way. The introduction of new media in the system must also be a relatively simple task.

The architecture is composed of a set of interacting components for each host: a *QoS Manager*, a *Media Device Manager*, a *Protocol Component*, *Media Device Drivers* and the applications themselves (figure 1). The first two components are specific for this architecture. The Protocol Component provides an interface to the protocols, which can be implemented by the component or in the operating system (OS). The Media Device Drivers components are interfaces to device drivers of continuous media (audio and video).

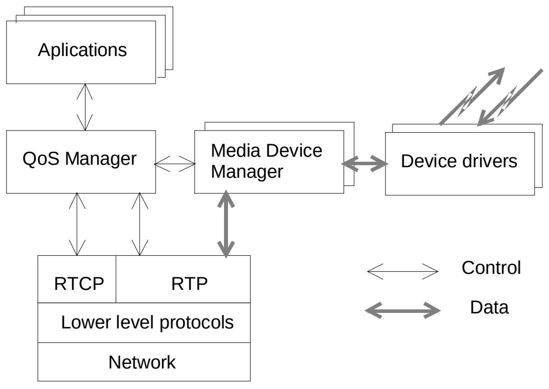

Figure 1. Architecture components in one of the system’s hosts.

A major simplification in the architecture is the facility to handle most of the tasks related to the multimedia data directly, outside of the applications. The *Media Device Manager* (MDM) performs whatever a medium needs to be processed and offers a control interface to enable external components to change its internal behaviour. The external components can be the QoS Manager for issues related with QoS, or the application itself for other kind of actions. As this paper is concerned with QoS we will only focus on the interaction with the QoS Manager. Basically, the MDM gets data to and from the protocols component and the device drivers (and from there to and from the peripherals such as disks, video windows, speakers, etc.). So, the MDM hides the media device drivers complexity from the remainder of the system.

The *QoS Manager* is the component that adjusts the QoS of the system to the current conditions of the network and hosts. These conditions are measured by monitoring, using RTCP packets (section 4.1). It also manages RTP sessions on behalf of the application. Monitoring is highly dependent on the specific media being handled, so little generalization can be achieved. However, if the interactions are normalized it is easy to reuse algorithms and ways of acting. A three part structure was devised: 1) the monitoring activity which produces an output value; 2) a decision algorithm to control the QoS, which uses 3) a set of scenarios to act upon the media to change some conditions. The introduction of a new medium needs the linking of a monitoring algorithm to the QoS Manager and the provision of the related actions in the Media Device Manager.

QoS management, monitoring and control, is performed only at the sender using the information returned by RTCP receiver report packets.

3.1. QoS parameters

QoS is adapted by changing the values of certain *QoS parameters*. Examples of QoS parameters are frame rate (video), colour depth (video), sample rate (audio), packet loss rate and latency.

QoS in multimedia has a slightly different meaning than the traditional concept in the OSI model [9]. Although both end up to values and rates of cell loss, throughput, delay, etc., it is difficult to relate the relative importance of these entities to the overall multimedia quality. A multimedia application user has a subjective assessment of the data (s)he is receiving, and, for instance, a greater delay of a frame than its presentation time is equivalent to a loss. Therefore, it is very difficult to get concrete values for the lower level concepts listed above from the user or programmer. He wants to express his intentions in terms of video frame rates, quality factors, audio sampling rates and encoding quality, etc. So, QoS parameters can be divided in two classes:

- *Programming level QoS parameters*: the QoS parameters that make sense to the programmer: frame rate, frame loss rate, Q factor, etc. These are the parameters used by the application to negotiate with the QoS Manager.
- *System (OS and network) level QoS parameters*: lower level QoS parameters that make sense to the system: packet size, bandwidth, packet loss rate, frame loss rate, etc. These are the parameters used by the Media Device Manager to interact with the network and the operating system. This component has to make translations between the two classes of QoS parameters. The idea is to give the programmer an abstraction that makes sense to him and not just to the network or OS.

3.2. Application -- QoS Manager interaction

The application has to define for each *media stream* a *QoS scale* composed by a sequence of *QoS levels* (the scenarios) that define the working states for that stream. Each level of the sequence represents a lower QoS than its predecessor.

A QoS level has a set of QoS parameter values that are media dependent and meaningful to the monitoring part of the QoS Manager and the Media Device Manager[1]. For example, a five level QoS scale for a M-JPEG video stream can have values for frame rate and Q factor at

[1] It is also plausible to think that some parameters are common to all the media as, for example, latency.

each level: [(25, 30), (15, 30), (15, 100), (5, 100), (5, 150)]. The bandwidth and the need for resources on hosts decrease as the system goes through the scale.

A *composed media stream* is a set of media streams. The application uses this concept to define order relations between them that are used by the QoS Manager to decide in which stream should perform actions first (for better or worse). These order relations are defined in terms of priorities. I.e., each stream has a priority in its composed stream and each composed stream has a priority. This permits, for example, to have a composed media stream of one audio stream and one video stream for each way of a video-conference; the audio stream should have a higher priority because in such a service audio quality is more important than video. If problems arrive, the QoS manager will decrease the video's QoS before the audio's.

The application - QoS Manager interaction has two relevant phases: initialization and run-time.

In the initialization phase of a stream the application requests the manager to open one (or more) channel and gives it the corresponding QoS scale. It also selects the desired QoS initial level (bandwidth reservations can be obtained using this initial level or the lowest level of the scale). The application defines this initial point in terms of Programming Level QoS parameters. A translation to System Level QoS parameters used to negotiate with the network must be performed. To prevent the QoS Manager to become too media specific the Media Device Manager must be invoked to do it.

The QoS Manager either succeeds in establishing the connection or not and informs the application. The application can also register its interest to receive some events from the QoS Manager. Events are associated with QoS level changes: change to the minimum QoS level, any change, etc.

In the run-time phase there are two kinds of actions: 1) The QoS Manager can send events to the application; 2) The application can invoke operations on the QoS Manager, such as terminate a stream, change the present QoS level, ask for status (QoS levels) of all the streams, add/delete a QoS level, change the events registration, change priorities, and change the composition of a composed stream.

Given these mechanisms the programmer can define different policies of QoS control for the application according to its needs.

3.3. QoS Manager -- Media Device Manager interaction

The interaction between the QoS Manager and the Media Device Manager starts with the indication of the parameters for the initial level, in order to initialize the Device Drivers and to get the System Level QoS parameters back.

The other kind of interaction is the request to change something on the data stream. There are less severe actions performed to avoid changing the QoS level which are media dependent (one operation per medium). Examples of a less severe action for audio is the enlargement of packet sizes and for MPEG is the discard of a B frame. Severe actions are the change of QoS levels and the set of parameters of the new level must be given to the Media Device Manager.

4. End-to-End Control Algorithm

QoS management consists of QoS monitoring and QoS control algorithms. The aim of the first is to observe the working conditions, whereas the second acts to adjust the system towards a new stable condition when problems happen or when resources become available.

The basic idea is the following: when network congestion is detected or when machine loads prevent the application from guaranteeing the current QoS level some action on one or more streams have to be done. Depending on the severity of the problem, the QoS control invokes a less severe action on the Media Device Manager, or forces it to decrease a QoS level on its stream. The streams are selected based on the priorities referred in section 3.2. On the other hand, when the QoS level is lower than it could be and the system is not at its top level, the QoS level is increased. The objective is to have the better QoS that the system can provide. Most of the times, if more is requested less will be obtained. An intuitive example that illustrates this is when the network is congested: it will only get worse if the same amount of data keeps being sent.

4.1. QoS Monitoring

The QoS monitor works with an end-to-end closed control loop. The algorithm is cyclic and reacts each time an RCTP receiver report packet is received (figure 2) [13]. The monitor looks at the behaviour of both network level (e.g., cell loss, jitter) and application level events (e.g., number of discarded frames at the receiver due to machine overload). In order to have the latter information, the control packets had to be extended with the following two fields (the extension feature is considered in RTCP by the inclusion of *profile-specific extensions*):

- **too_late** – indicates the number of frames lost or delayed (in relation to its presentation time) in the network or at the sender.
- **nshown** – indicates the amount of frames discarded by the receiver due to machine overload. It is mainly related with the scheduling procedure at the receiver and gives a good indication of the application's local behaviour and machine load.

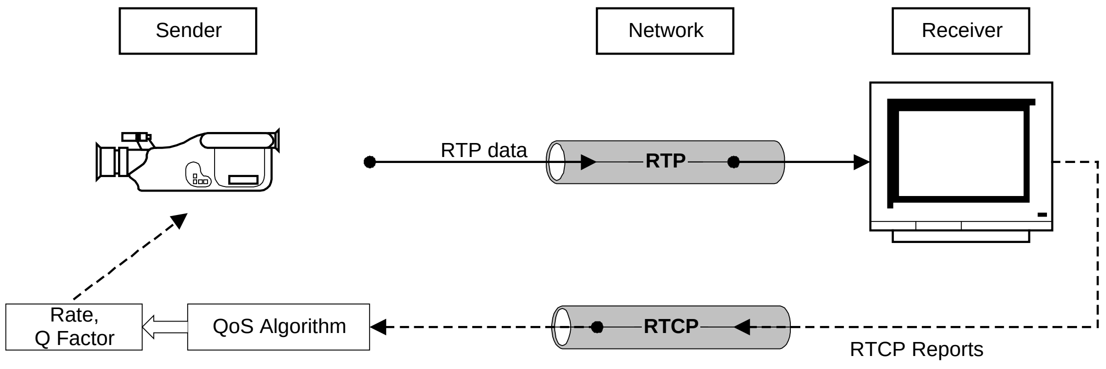

Figure 2. QoS monitoring scheme.

The number of *frames lost* is the sum of: 1) the number of frames lost in the network (obtained from the numbers of packets lost given by the receiver report packet), 2) too_late and 3) nshown. In order to avoid peaks on the output value and to limit the oscillations on the control algorithm the average of the last sums is performed. We called this value *filtered loss*. In the experiments we used the three past reported values for this average.

The output of the QoS monitoring is the filtered loss value that is analyzed by the QoS control mechanism in order to decide the action to take.

4.2. QoS Control

The QoS control algorithm was adapted from [6], [2] and [3]. It must be tuned to achieve a viable compromise between the need to optimize the use of resources and the stability required by the application. The algorithm should not change the QoS parameters due to small variations of the input value because the changes will introduce overload causing even more changes. On the other hand, the algorithm should not become too insensitive that it can not adapt to network changes, losing the required dynamic behaviour. Moreover, it should identify situations when a less severe action could be tried to solve the problem, and if it fails then a change of QoS level should be performed.

The concrete algorithm tested has three different zones (figure 3) in which the input value can fall: *degradation zone* (between λ_s and 100%); *working zone* (between λ_i and λ_s) and *improvement zone* (between 0 % and λ_i). The values λ_i and λ_s are calculated experimentally and must be given by the QoS Manager for each data type.

Our experiments used MJPEG and the less severe actions consisted on dropping an entire frame. If the input value rises so quickly that would pass the working zone, entering in the degradation zone, a QoS change would be triggered immediately. If not, there would be space to invoke a less severe action assuming that the problem is a transient one.

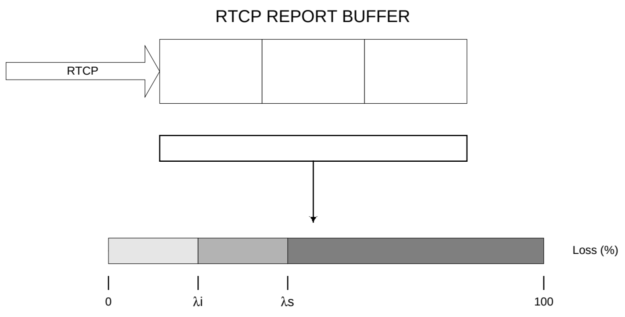

Figure 3. QoS Control Algorithm

Whenever a change in the QoS parameters is necessary, the algorithm moves through the QoS scale. It moves downwards if the conditions of the network do not allow the application to continue using the current QoS, or upwards otherwise, i.e., if the network conditions allow the application to improve its QoS. More precisely, the control algorithm works as follows:

- When the input value falls into the working zone, the algorithm acts autonomously discarding a frame once on a while, if necessary. These losses are not noticeable to the users and constitutes the stable stage of the algorithm.
- When the input value falls into the degradation zone, the algorithm moves downwards on the scale provided by the application to reduce the volume of data (changing parameters of the encoding sequence like the quality factors, frame rate, etc.).
- When the input value falls into the improvement zone, the algorithm moves upwards on the scale to try to use the resources that the network seems to have available at the moment.

Degradation and improvement are not symmetrical. A degradation is performed in principle when a real problem occur. On the other hand, an improvement of the QoS is performed when the current working conditions produce almost no errors; so the algorithm will try to ask for more. This last solution may cause oscillation of the QoS values, but it is not a disturbing factor for the users. Another solution would be to measure the real bandwidth available but that is not simple to do and would require extra load in the machines and network.

5. Experiments

Some experiments were performed in order to assess the algorithm's behaviour in different network and workstation conditions.

The experiments consisted of video images, with a fair amount of movement and some detail, encoded in JPEG. The experimental network consisted on an ATM switch, Sun stations and Parallax JPEG video compression/decompression boards. The QoS scale had the following nine levels (frame rate, quality factor): [(25, 50), (22, 50), (19, 50), (25, 75), (22, 75), (19, 75), (25, 100), (22, 100), (19, 100)]. The threshold values chosen for both experiments were $\lambda_i = 5\%$ and $\lambda_s = 15\%$.

5.1. First Experiment

The goal of this experiment was to monitor the performance of the algorithm in normal working conditions (both of the network and the workstations). The network was not a critical resource and the size of the images were enlarged too much as to produce overload on the workstation. Each frame had to be transported by, at least, two transport segments and a rate of 25 fps produced a load higher than the receiving machine could cope with (the receiving machine is less powerful than the sending machine). Figure 4 illustrates the results. During the first part of the experiment, artificial load was introduced to the receiving machine.

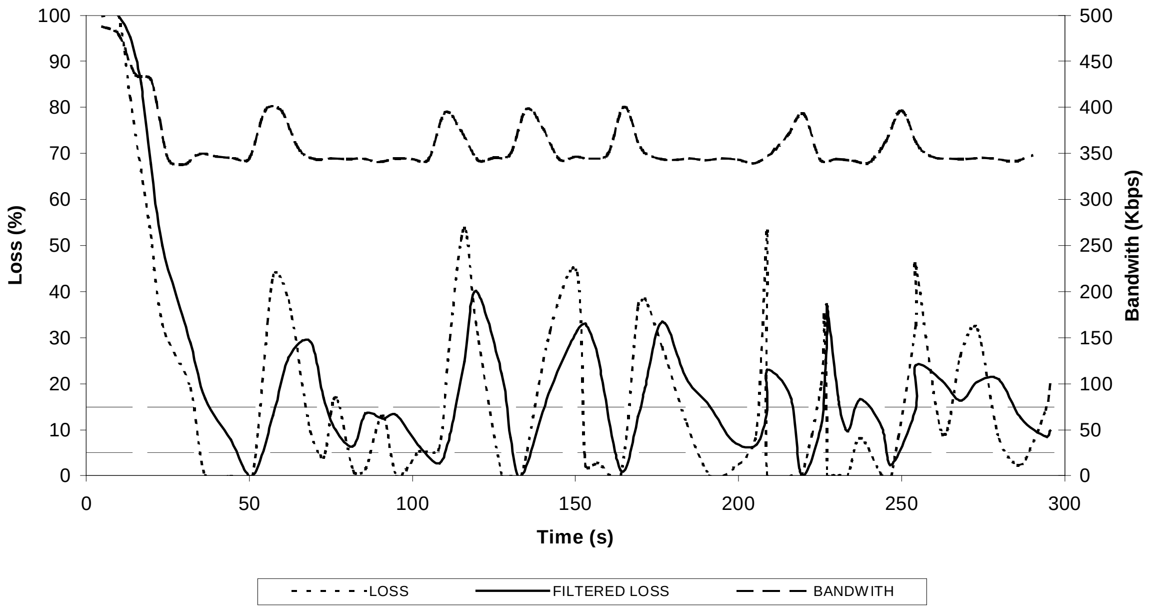

Figure 4. First Experiment

The initial QoS level chosen was too high and the system moved very quickly to the lowest level (level nine). At t=50s it improved a little bit but the losses start to rise and the system moved downwards again. At t=105s it made a second attempt to go to level eight but it happen the same thing as before. Typically, the stable point would be between level eight and nine, very close to level nine. This behaviour happened again some more times and around t=195s the extra load on the workstation ceased to exist. The system moved to level one at t=260 but not continuously (i.e., there were some oscillations around level five and again around level 3). These oscillations were due to the effects of changing the parameters on such a heavy system (due to the image size). Level one was clearly to high and the system got stable around level five. The figure shows the attenuating effect of the average losses, and the increase of the bandwidth each time a new level is tried. The total amount of the bandwidth is not so great, as the system's bottleneck are the machines. It is also important to refer that the RTP was implemented as a user process so some inefficiencies exist.

5.2. Second Experiment

To assess the algorithm's behaviour for different network conditions, artificial losses were introduced on the network (at the sender's side). The experiment started with a bandwidth limitation of 350 Kbps. At t=120s it was reduced to 277 Kbps, and again to 229Kbps at t=180s. There was a final reduction to 182 Kbps at t=260s and, after that, two increases: one to 230Kbps at t=350s and another to 350Kbps at t=450s. The video image size was smaller than the one used for the first experiment.

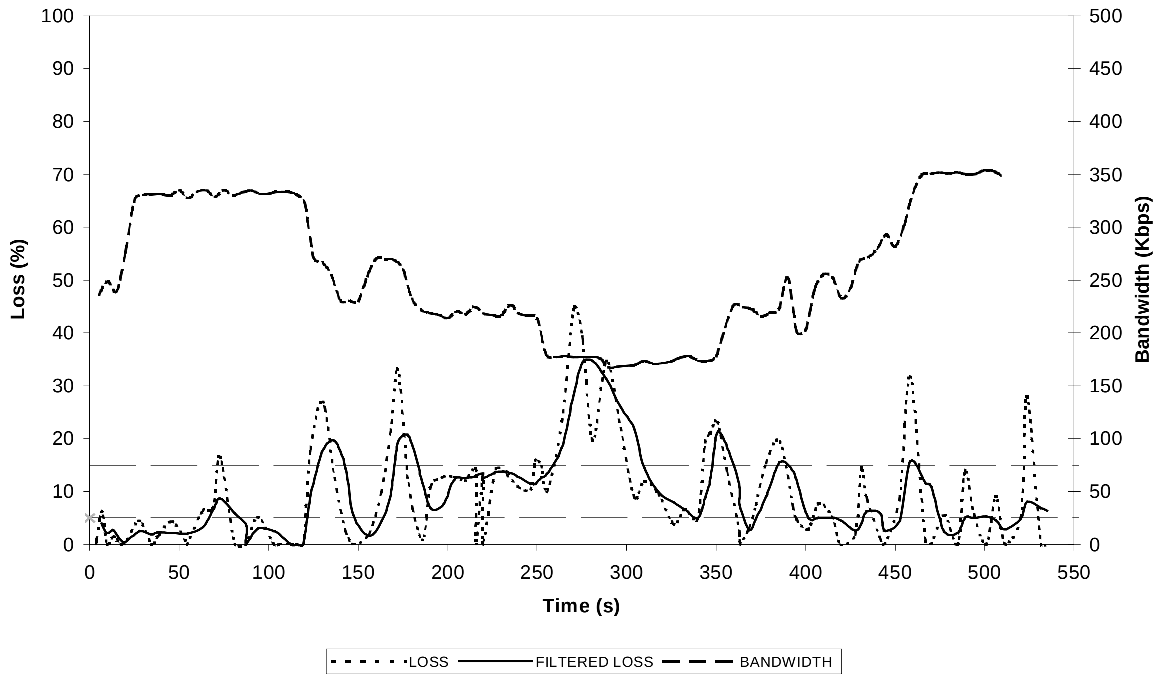

Figure 5. Second Experiment

Figure 5 shows the results of the experiment. This time, the initial level was too low and a series of improvements were performed. The system moved quickly to level one (t=30s) and could have gone higher if the scale allowed it. When the bandwidth was reduced the losses started to rise and the system went down to level three. It remained more or less at this level after the reduction at t=180 (although the losses started to rise slowly). When the final reduction was performed the system went down all the way to level nine. At this point, it would have gone lower if it could. The application should have been notified that the lowest level was reached and should have reacted by terminating the connection or providing a new scale. This part of the algorithm was not implemented so the system had to remain at level nine. The consequence is that the algorithm is virtually not working and the losses start to rise without control. This is clear in the figure. When the bandwidth was increased, the system moved to level five, and after the second increase to level one, increase the used bandwidth.

6. Related Work

[15] provides a survey about QoS on distributed multimedia systems. They single out a proposal [10] to handle our Programming level QoS parameters based on “Quality Query by Example” to make the user choose the QoS without caring for the meaningless (to him) System level QoS parameters.

The dynamic QoS adjustment mechanism described in [2] has strong similarities to our own but no QoS management architecture is proposed (the algorithm is part of the application). The sending application uses RTCP receiver report packets to compute packet losses. Based on this measure the available bandwidth is determined and the QoS is adjusted to that value. The algorithm is similar to the one proposed in this paper although ours takes not only packet loss in consideration but packet delays as well in order to measure the available bandwidth. This means that we not only consider the network conditions as [2] does, but also the end machines load conditions.

[3] proposes an abstract architecture for QoS handling called QoS Architecture (QoS-A). [4] proposes a layer that is part of this architecture, called "multimedia enhanced transport system" (METS), in order to manage QoS for the application. This approach is similar to ours although our architecture is more complex and can handle completely multimedia data for the application and not just to adjust QoS. [4] also proposes an algorithm similar to ours but that does not use RTP/RTCP.

A specific scheme to control network congestion based on bit and packet rate scaling, i.e., QoS degradation, in a system without bandwidth reservation is focused in [14]. The key problem treated is to find sustainable acceptable values for the QoS during a congestion. The algorithm is similar to the one we studied although the end hosts situation is not considered. Also no architecture is proposed.

In [11] a flexible architecture for QoS establishment is proposed. The system takes into account application, network and operating system concepts and a QoS broker calculates their best balance to provide the desired QoS. It is mainly a configuration solution than an on-going control schema, and requires too much from the environment (real-time OS, scheduler policies, etc). There is also a mapping from application QoS parameters to network ones, but it has little change of meaning between the two.

The subject of QoS has also been a topic of standardization. The ISO standard [9] defines a "QoS framework". It describes how QoS can be characterized and QoS restrictions specified. A set of mechanisms for QoS management are identified. A list of QoS parameters is also given. Our algorithm can be thought as an implementation of the generic mechanism they propose for QoS control. No architecture is proposed.

A question related to the QoS control of this paper is receiver-dependent QoS for multicast. The problem, treated in [7], is how to deliver different levels of QoS to different receivers using multicast. Our algorithm can not be used in that case because it is the sender that adjusts the sending rate to the overall system situation. For multicast the QoS would have to be imposed by the receiver in the worst situation (for example, connected to the sender by the lower bandwidth network path), what is not acceptable. [7] proposes a solution based in filters that adjust the QoS of the media being sent to each receiver or group of receivers situation. This question is orthogonal to the problem we studied.

Another related aspect, more media specific, is the challenge to maximize the QoS of an MPEG stream. The proposals presented in [8] and [12] perform that task by scheduling the MPEG frames according to their relevance for the receiver (higher priority to I frames, and lower to B). The user has a lower perception when a lower priority frame is discarded so they are discarded first, when some action is needed. We did not consider these questions (nor even MPEG) but some of the ideas of these papers could be used to extend our algorithms for that media format.

7. Conclusions

The QoS control for distributed multimedia applications has to take into consideration several aspects and applications should be shielded as much as possible from this complexity. The issue is not that simple because multimedia data has very different requirements depending on the specific medium. A flexible framework is thus the best approach in order to be extended with different modules as signal processing improves. The framework should handle the problem the best it can without imposing too many restriction on the environment (operating system, device drivers) to avoid becoming useless. The proposal presented here has the advantage of working by monitoring, and not by strict interaction with the environment (bandwidth reservation), making it suitable to work within many existing environments.

With respect to the algorithms we believe that the best approach is an integrated one that includes the sender and the receiver machines as well as the network. This is visible in this paper in the integration of the transport entity with the application and in the fact that the closed loop control mechanism takes the end machines load into consideration. Another important feature is the ability of the applications to express its requirements using concepts that are more meaningful than the usual low-level classical parameters.

Experiments have shown that the mapping between these concepts and the low-level ones can be achieved successfully and the subjective evaluation have shown that users notice jumps on the quality of the session but it is not a major distracting factor.

As a topic for further work the use of this kind of algorithms in multicast configurations will be studied.

8. Bibliography

1. Antunes N., Rocha R., Pinto P. (1997). “Analysis and Simulation of a Traffic Management Control Scheme for ATM Switches with Loose Commitments”, Int. Conf. On Networks and Distributed Systems Modeling and Simulation, Phoenix, 1997, ftp://mariel.inesc.pt/pub/papers/cndsmsc97.ps.gz

2. Busse I., Deffner B., Schulzrinne H. (1995). “Dynamic QoS Control of Multimedia Applications based on RTP”, Computer Communications, Vol. 19, Number 1, Jan. 96

3. Campbell A., Coulson G. (1996). “A QoS Adaptive Transport System: Design, Implementation and Experience”, ACM Multimedia´96, Boston, 1996, 117-127

4. Campbell A., Coulson G., Hutchison D. (1994). “A Quality of Service Architecture”, ACM SIGCOMM 94, Computer Communication Review, Vol.24, April 1994, 6-27

5. Clark D., Tennenhouse D. (1990). “Architectural Considerations for a New Generation of Protocols”, ACM SIGCOMM 90, Philadelphia, 1990, 200-208

6. Correia M., Pinto P. (1995). “Low-Level Multimedia Synchronization Algorithms on Broadband Networks”, ACM Multimedia´95, San Francisco, 1995, 423-434, ftp://mariel.inesc.pt/pub/papers/mm95.ps.gz

7. Garcia F., Hutchison D., Mauthe A., Yeadon N. (1996). “QoS Support for Distributed Multimedia Applications”, Proceed Int. Conf. in Distributed Processing (ICDP´96), Dresden 1996

8. Han C., Shin K. (1995). “Scheduling MPEG-Compressed Video Streams with Firm Deadline Constraints”, ACM Multimedia´95, San Francisco, 1995, 411-422

9. ISO/IEC JTC1/SC21, (1995).“Information Technology – Quality of Service Framework – Final CD”, July 1995

10. Kalkbrenner G. et al. (1994). ″Quality of Service (QOS) in Distributed Hypermedia Systems″, Proc. 2nd Int´l Workshop on Principles of Document Processing, 1994.

11. Nahrstedt K., Smith J. (1995). ″The QoS Broker″, IEEE Multimedia, Spring 1995

12. Riley M., Richardson E. (1994). “Minimizing the Effect of Cell Losses on MPEG Video”, BRIS´94, Hamburg 1994, 491-494

13. Schulzrinne H., Casner S., Frederick R., Jacobson V. (1996). “RTP: A Transport Protocol for Real-Time Application”, (RFC 1889) January 1996

14. Talley L., Jeffay K. (1994). ″Two-Dimensional Scaling Techniques for Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams″, ACM Multimedia´94, S. Francisco, 1994

15. Vogel A., Kerhervé B., Bochmann G., Gecsei J. (1995). “Distributed Multimedia and QoS: A Survey”, IEEE Multimedia, Vol.2, Numb2, 1995