ARCHITECTURE AND PERFORMANCE EVALUATION FOR REDUNDANT MULTICAST TRANSMISSION SUPPORTING ADAPTIVE QOS

A. Gkamas Ch. Bouras An. Karaliotas K. Stamos
Computer Technology Institute - Computer Eng. and Informatics Dept., Univ. of Patras, Greece
Riga Feraiou 61, GR-26221 Patras, Greece. E-mail: {bouras, gkamas, karaliot, stamos}@cti.gr

Abstract: *In this paper we describe the architecture of an application that was developed for the transmission of multimedia data, using the multicast mechanism, over the Internet. There are two major issues that have to be considered when designing and implementing such a service, the fairness and the adaptation schemes. In our application we use a mechanism that categorizes the receivers into a number of groups according to each receiver's capabilities and (the mechanism) serves each group of users with a different multicast stream. We have also implemented an additional mechanism for the intra-stream bit rate adaptation. The proposed mechanism uses a "friendly" to the network users congestion control policy to control the transmission of the data. We evaluate the adaptive multicast transmission mechanism through a number of experiments in order to examine its behaviour to a heterogeneous group of receivers and its behaviour against TCP and UDP data streams.*

KEYWORDS: IP Based Networks and Services, Multimedia Systems and Services, Multicast, Quality of Service, Adaptation mechanisms, CORBA.

INTRODUCTION

The heterogeneous network environment that Internet provides to the real time applications as well as the lack of sufficient Quality of Service (QoS) guarantees, many times forces applications to embody adaptation elements in order to work efficiently. The main goal of such an approach is to adapt the data rate that is sent to the network every time that network conditions change. The decision whether the rate will increase or decrease is based on feedback information that the receivers send back to the sender. Many researchers believe that this end-to-end control scheme must be implemented in the application layer because today's Internet architecture does not provide such a mechanism in the network layer. In addition any application that sends data (mostly multimedia) over the Internet should have a friendly behaviour towards the other flows that coexist in today's Internet and especially towards the TCP flows that comprise the majority of flows ([5]).

The system we propose is based on multicast video transmission with the use of RTP/RTCP ([2]). The main perspectives we tried to fulfil are the followings:

1. Each receiver should receive the best video quality that it is capable of
2. Generated multicast data flow should not be a constraint for the other flows.

In order to achieve the first goal we create *n* different streams, each one within certain bandwidth limits. All the streams carry the same video information, each one of them having a different quality. Receivers join in the appropriate stream depending on the condition of the network path towards them and the processing power of each one. If meanwhile the receiver detects that the stream it has joined is not suitable for it any more another implemented mechanism is used in order to provide the receivers with the capability of moving into another stream. In order to achieve the second goal (friendliness towards other flows), we deploy the Additive Increase Multiple Decrease (AIMD) scheme in the inter-stream adaptation algorithm.

The subject of adaptive streaming of multimedia data over networks has engaged researchers all over the world. The methods proposed for the multicast transmission of time sensitive data in the Internet can be generally divided in three main categories, depending on the number of multicast sessions used:

3. The source uses a single multicast session for all receivers ([1], [3], [6]).
4. Simulcast: The source transmits versions of the same video encoded in varying degrees of quality. This results to the creation of a small number of multicast sessions with different rates, responsible for a range of receivers with similar capabilities ([7]).
5. The source uses layered encoded video, which is video that can be reconstructed from a number of discrete data streams and transmit each layer into different multicast session ([4]). The receivers subscribe to one or more multicast sessions depending on the available bandwidth into the network path to the source. The video is divided into one basic stream and more additional streams. The basic stream provides the basic quality and the quality improves with each layer added.

This work is based on the simulcast approach and it is an extension of the work, which has been presented in [8] and [7]. In this paper we evaluate the implemented prototype, which has been presented in [9].

1 SYSTEM ARCHITECTURE

Our system is based on the multicast transmission of video. On the session layer, according to the OSI reference model, we use the Real Time Protocol (RTP). The RTP-RTCP protocol was also used because of the feedback capabilities that it offers (RTCP reports). At the same time, a CORBA (Common Object Request Broker Architecture) connection is established between each Client and the Server. This connection provides the necessary information to the Client concerning the multicast IP address as well as the port that will be used. Because of the variations on the quality of video that various Clients can handle, the source transmits a small number of (in our implemented prototype we use three) different multicast video streams, each one with its own bandwidth limits, with no overlapping. The transmission rate within each stream is adapting within its limits according to the capabilities and the state of the Clients participating in it. The Server is unique and responsible of:

1. Creating the *n* different multicast streams
2. Setting each one's bandwidth limits,
3. Tracking if there are any Clients that are not handled with fairness and
4. Providing the mechanisms to the Clients to change stream whenever they consider that they should be in another stream closer to their capabilities.

The Server generates *n* different Stream Managers. In each Stream Manager an arbitrary number of Client Managers is assigned. Each Client Manager corresponds to a unique receiver that has joined the stream controlled by this Stream Manager. The Server uses a Synchronisation Server, which is responsible for the management, synchronization and intercommunication between Stream Managers. The Stream Manager is responsible for the maintenance and the monitoring of one of the *n* different multicast streams that are generated in the beginning of the application. Also the Stream Manager entity has all the intra-stream adaptation mechanisms for the adjustment of the transmission rate. The Client Manager corresponds to a unique Client. It processes the RTCP reports generated by the Client and can be considered as a representative of the Client at the side of the Server. Client Manager receives the RTCP reports from the Client and processes them based on packet loss rate and delay jitter information. It then makes an estimation of the state of the Client, based on the current and a few previous reports that it stores in a buffer.

2 DESCRIPTION OF SYSTEM OPERATION AND ALGORITHMS

The source initially constructs a number of streams. When a Client wants to start receiving video, it requests from the Server the address of a multicast session belonging to a transmitting stream (through CORBA communication). After the Client joins the multicast session, a dedicated Client Manager is created to represent the Client at the side of the Server. RTCP reports are sent back to the stream and in particular to the appropriate Client Manager. Information in RTCP reports contains two values that describe the quality of the transmission: packet loss rate and delay jitter. These values are passed through the following filters used to avoid wrong estimations and determine the aggressiveness of the feedback analysis protocol: For the packet loss rate:

$$LR_{new} = a * LR_{old} + (1-a) * LR_{net} \qquad (1)$$

Where:

- LR_{new}: The new filtered value of packet loss rate.
- LR_{old}: The previous filtered value of packet loss rate.
- LR_{net}: The packet loss value that was contained in the RTCP report received from the Client.
- a: a parameter that determines the aggressiveness of the adaptation concerning the packet loss value. Its value ranges from 0 to 1.

For delay jitter:

$$J_{new} = b * J_{old} + (1-b) * J_{net} \qquad (2)$$

Where:

- J_{new}: The new filtered value of delay jitter.
- J_{old}: The previous filtered value of delay jitter.
- J_{net}: The delay jitter that was contained in the RTCP report received from the Client.
- b: a parameter that determines the aggressiveness of the adaptation concerning the delay jitter value. Its value ranges from 0 to 1.

For the sake of clarity, a distinction has to be made between two kinds of states, that both can take the values of UNLOADED, LOADED or CONGESTED: we call the first one the "unprocessed state" and the second the "processed state". The unprocessed state is derived directly from the filtered values of packet loss rate and delay jitter, according to the following rules:

$$\begin{gathered} \textit{if } (LR_{new} >= LR_c) \textit{ unprocessed state} = \textit{CONGESTED} \\ \textit{if } (LR_u < LR_{new} < LR_c) \textit{ unprocessed state} = \textit{LOADED} \\ \textit{if } (LR_{new} <= LR_u) \textit{ unprocessed state} = \textit{UNLOADED} \\ \textit{if } (J_{new} > \gamma * J_{old}) \textit{ unprocessed state} = \textit{CONGESTED} \end{gathered} \qquad (3)$$

We have defined LR_U as the maximum value of the unloaded packet loss rate and LR_C as the minimum value of the congested packet loss rate. Where γ is a parameter, which specifies how aggressive the network condition estimation component will be to the increase of delay jitter.

The state that will be reported to the Stream Manager is called the processed state. It is computed by taking into account the last *n* unprocessed states, which are held in an n-sized buffer in the Client Manager. A CONGESTED unprocessed state does not necessarily impose that the processed state will also be congested, especially if the majority of the previous "unprocessed states" were UNLOADED. The way the processed state is computed as presented below: We first introduce a new variable, USV (Unprocessed State Variable), that takes a new value for each unprocessed state as shown:

$$\begin{array}{l} \textit{if (unprocessed state}_i == \textit{CONGESTED) then } USV_i = -1 \\ \textit{if (unprocessed state}_i == \textit{LOADED) then } USV_i = 0 \\ \textit{if (unprocessed state}_i == \textit{UNLOADED) then } USV_i = 1 \end{array} \qquad (4)$$

The processed state is then determined by the value of

$$f(i) = state_i * w_i + state_{i-1} * w_{i-1} + \ldots + state_{i-n+2} * w_{i-n+2} + state_{i-n+1} * w_{i-n+1}$$

where w_i, …, w_{i-n+1} are weights used to quantify the decreasing importance of old unprocessed states.

$$\begin{array}{l} \textit{if (f(i) < 0) then processed state}_i = \textit{CONGESTED} \\ \textit{if (f(i) == 0) then processed state}_i = \textit{LOADED} \\ \textit{if (f(i) > 0) then processed state}_i = \textit{UNLOADED} \end{array} \qquad (5)$$

We have chosen to completely ignore the first RTCP report since the moment a Client joins a new stream, because we observed that this report usually contains a very high packet loss rate value. Stream Managers update their rates synchronously and therefore time in system operation is divided in epochs of certain length. At the end of an epoch, each Stream Manager polls the states of all the Client Managers that correspond to a Client receiving this stream and then determines the improvement or degradation in this stream's video quality. Whether there will be an improvement or degradation is determined as follows: If all receivers[1] are in the UNLOADED state, video quality is improved. If more than a certain threshold of receivers is CONGESTED, video quality is degraded. The threshold used for our experiments was one-third of all receivers listening to the stream.

The new bit rate is estimated using an Additive Increase, Multiplicative Decrease (AIMD) algorithm, just like TCP. Increase is achieved by adding a standard small value to the previous bit rate, and is therefore quite conservative in bandwidth consumption, while decrease is achieved by multiplying the previous bit rate with a number in the range of 0…1 (typically around 0.5) and so the algorithm is more aggressive when trying to react to congestion.

There are three cases in this phase that will lead to a Client's transition towards another stream: (1) If the stream from which the Client is currently receiving video has already reached its lowest transmitting rate and the Client is still in CONGESTED state then the Client stops listening to this stream and joins the session of a lower quality stream (if such a stream exists). (2) If the stream from which the Client is currently receiving video has already reached its highest transmitting rate and the Client is still in UNLOADED state then the Client stops listening to this stream and joins the session of a higher quality stream (if such a stream exists). (3) The third case applies to a Client that co-exists in a stream with low capacity receivers but is capable of handling better quality video, so it has been unable to improve the video quality of the current stream. The mechanism used aims at making the protocol more conservative and operates by counting the number of consecutive times the receiver was UNLOADED but failed to improve the video quality. When this number exceeds a certain limit, we assume that the receiver has indeed higher capabilities and move it to a better quality stream. For a Client transition to occur, an additional rule is imposed in all cases: the Client must have sent a minimum number of reports (for our experiments this number was set to 5) since it joined the stream it now wants to leave.

3 PERFORMANCE EVALUATION

In order to evaluate the performance of the implemented prototype, we run three different experiments, each with a different configuration, over a controlled networking test-bed, which we have implemented over the campus network of University of Patras in Greece. During our

[1] The number n of the receivers can easily computed by the RTCP protocol

experiments we use one Server and six Clients. We connect each participant (the Server and the six Clients) with connections of different capacity to our network test-bed with the use of traffic policy on the access router of each participant in the test-bed. The Server transmitted three streams with the following limits: stream 1:10Kbps-100Kbps, stream 2: 100Kbps-200Kbps and stream 3: 200Kbps-300Kbps. During the experiments the Server was using the following parameters in order to control the operation of the implemented mechanism: a=0.5, b=0.8, γ=2, LR_u=0.02, LR_c=0.05. The AIMD algorithm of the Server was increasing the transmission rate of a stream by 25Kbps, during network unloaded periods and was decreasing the transmission rate by 50% during network congestion periods. In the beginning of the all the experiments the Server transmits only stream 1 with transmission rate of 10Kbps and all the Clients are connected to stream 1.

3.1 First experiment: Transmission into a heterogeneous group of receivers

During this experiment we investigate the behaviour of the implemented prototype with a heterogeneous group of receivers. In order to implement a test-bed with a heterogeneous group of receivers the Server's outgoing link and each Client's incoming link is restricted to a specific capacity, as Figure 1 shows. We run this experiment for 360 seconds.

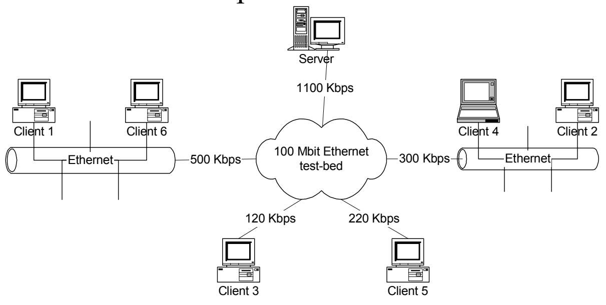

Figure 1 Experiment topology

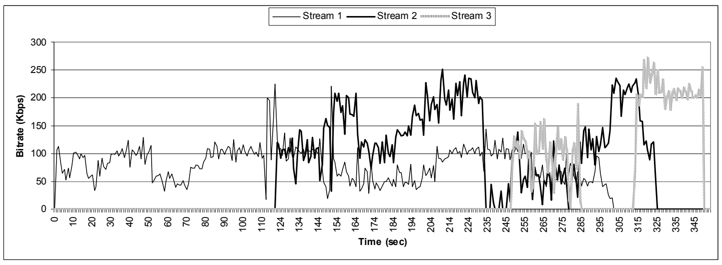

Figure 2 Transmission rates of Server streams during first experiment

Figure 2 shows the transmission rates of Server streams and Figure 3 shows the reception rates of all the Clients. The straight lines in Figure 3 show the Server stream number from which the Client receives video. As Figure 2 shows, when the experiment starts, the Server transmits only the stream one and the streams two and three are inactive because all the Clients initially join stream one. The transmission of stream two and stream three start at the 122nd and 248th second respectively, when some Clients prefer transmission rates more than stream one can deliver. Each Client starts receiving video at a different time point. As Figure 3 shows, Client 3 joins stream one of the Server and keeps receiving this stream until the end of the experiment. This behaviour of Client 3 is as expected because Client 3 is connected to the test-bed with a 120Kbps link that discourages it to join a stream with a higher transmission rate. Client 2 and Client 4, which share the same link with 300Kbps capacity, join stream one, one after the other. After some period of time during which both Client 2 and Client 4 are receiving stream one, Client 4 (100th second) joins stream two. The capacity of the link (300 Kbps) allows the simultaneous reception of stream one (max 100 Kbps) and stream two (max 200 Kbps). Therefore Client 2 joins stream two at the 135th second. At the 223rd second, Client 4 tries to join stream three. This causes congestion to the link and therefore both Clients move to a lower stream (Client 4 returns to stream two and Client 2 moves to stream one). Client 6 and Client 1, which share the same link with 500Kbps capacity, join stream one, one after the other. After some time Client 6 and Client 1 join stream two almost at the same time point (145th second). Finally at the 321st second Client 6 joins the third stream because of the high capacity of the link (500 Kbps) that connects Client 6 to the test-bed. Client 5 is the last Client, which joins the video transmission. Initially Client 5 joins stream one and at the 257th second it joins stream two. The capacity of link (220 Kbps) that connects Client 5 to the test-bed does not allow Client 5 to receive stream three, so this is why Client 5 does not attempt to receive stream three.

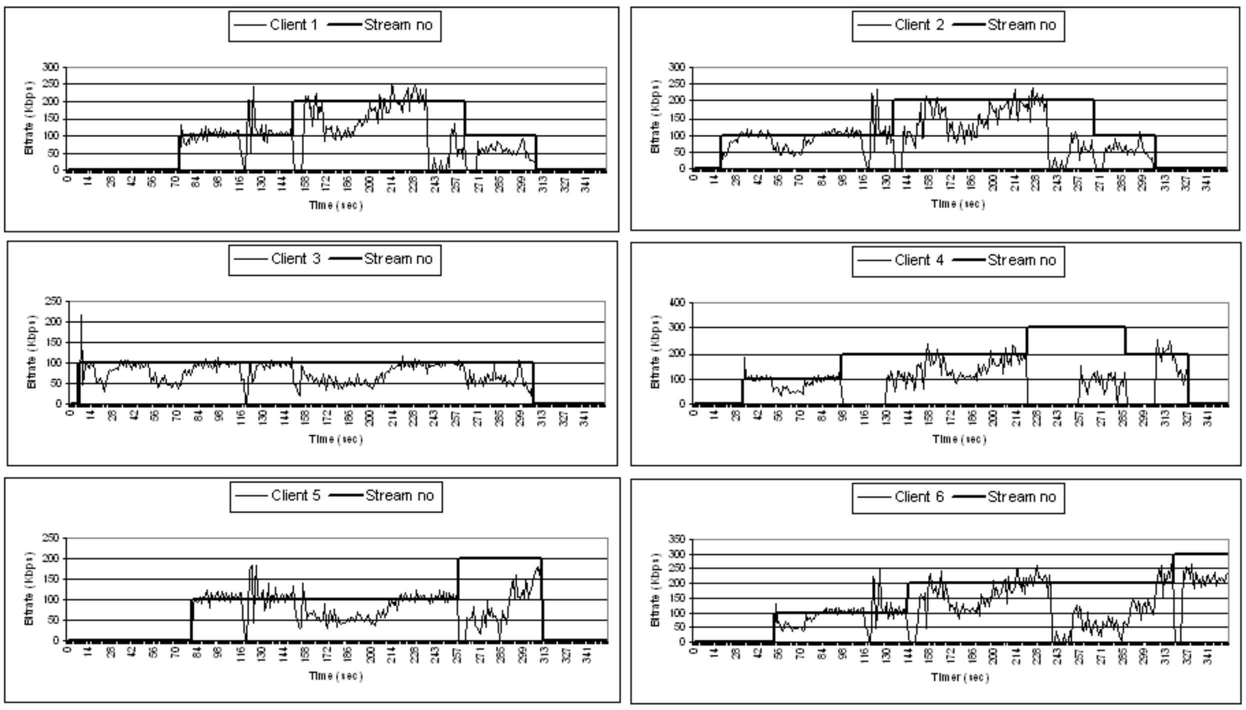

Figure 3 Clients reception rate during first experiment

The behaviour of the implemented prototype was as expected. All the Clients, depending on the capacity of the link that connects them to the network test-bed, join the appropriate stream and the Server treats all Clients with fairness. An exception is the behaviour of Client 1 between the 264th second and the 307th second. During the above period, we expected that Client 1 would join stream three but Client 1 moved to stream one. We believe that the above behaviour of Client 1 is a result of low resources in the workstation that Client 1 was running on. The allocation of the Clients to the appropriate stream takes some time. This is because of the conservative operation of the implemented prototype in order to be TCP-friendly as the following experiment shows.

3.2 Second experiment: Transmission with background TCP traffic

In this experiment, we transmit at the same time multimedia data with the use of the implemented prototype and TCP traffic in the same link. The topology of this experiment is the same with the topology of first experiment except for the capacity of the link, which connects Client 5 with the network test-bed. The capacity of that link has been increased to 300 Kbps. We simultaneously transmit to the link of Client 5 TCP traffic at an initial rate of 280Kbps. TCP traffic was generated by the LanTrafficV2[2] traffic generator, configured to send packets of 1436

[2] http://www.zti.fr

bytes each every 40 ms. Figure 4 shows the transmission rate of TCP traffic and the reception rate of Client 5 during the second experiment. We run this experiment for 600 seconds.
As Figure 4 shows the implemented prototype has friendly behaviour towards TCP traffic: Initially, before the transmission of video with the implemented prototype, TCP traffic consumes all the available bandwidth. When the transmission of video starts, Client 5 joins stream one of the Server and TCP traffic reduces its transmission rate due to the congestion, which takes place to the link of Client 5. The transmission rate of stream one increases, and Client 5 consumes bandwidth to the link. After some time Client 5 tries to join stream two with a higher transmission rate. This action of Client 5 produces congestion to the link and Client 5 backs off and returns to stream one in order to release bandwidth for the TCP traffic. The above described behaviour continues until the end of the experiment. We stop the transmission of video at the 560th second and TCP traffic consumes all the available bandwidth. During this experiment the TCP traffic has transmission rate of more than 100 Kbps and maximum transmission rate more than 200 Kbps, which is good performance for TCP transmission.

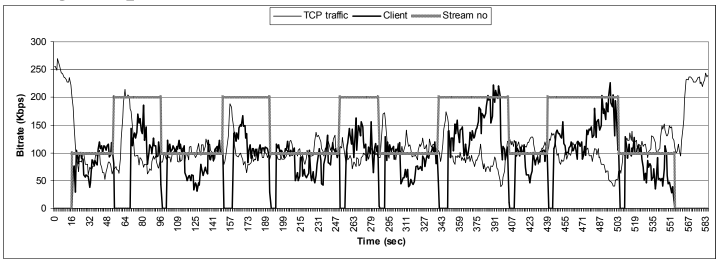

Figure 4 Transmission rate of TCP traffic and the implemented prototype

3.3 Third experiment: Transmission with background UDP traffic

In this experiment, we transmit at the same time multimedia data with the use of the implemented prototype and UDP traffic in the same link. The topology of this experiment is the same with the topology of second experiment. We have again one Server that multicasts multimedia data to the group of Clients. We simultaneously transmit to the link of Client 5 UDP traffic at an initial rate of 280Kbps. At certain time points during the experiment we decreased the UDP transmitting rate in order to test whether our application would be able to use the available bandwidth. UDP traffic began at a rate of 280Kbps, and we later decreased it to 250, 200 and 175Kbps. UDP traffic was generated by the LanTrafficV2 traffic generator, configured to send packets of 1436 bytes each (the Ethernet MTU) every 40, 45, 55 and 60 ms, according to the sending rate we wanted to achieve. Figure 5 shows the transmission rate of UDP traffic and the reception rate of Client 5 during the third experiment. We run this experiment for 420 seconds.

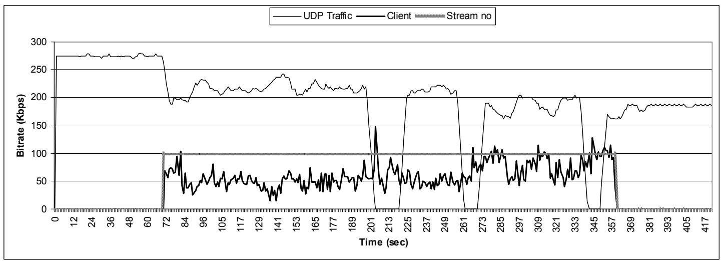

Figure 5 Transmission rate of UDP traffic and the implemented prototype

As Figure 5 shows, when the experiment starts, UDP traffic occupies all the available bandwidth. When the implemented prototype starts video transmission (70th second), Client 5 joins stream one of the Server and the UDP traffic reduces its transmission rate. Although UDP traffic reduces its transmission rate, this reduction is not sufficient and the UDP traffic continues to dominate the available bandwidth. The video transmission consumes only around 50 Kbps of link capacity. At the 204th, 262nd and 342nd second we changed the parameters of the traffic generator in order to reduce the transmission rate of UDP traffic (at the above time periods we briefly stopped the transmission rate of UDP traffic in order to change the parameters of the traffic generator). Gradually the video transmission consumes more bandwidth in the link but again the UDP traffic dominates the link capacity. We stop the transmission of video at the 360th second. The above described behaviour of the implemented prototype is as expected because during the design of the implemented prototype we focused on implementing a TCP friendly application.

4 CONCLUSION - FUTURE WORK

In this paper, we present the architecture of a prototype for multicast transmission of adaptive multimedia data in a heterogeneous group of receivers with the use of replicated streams. We concentrate on the design of a mechanism for monitoring the network condition and estimate the appropriate rate for the transmission of the multimedia data in each stream in order to allocate each receiver to the appropriate stream and treat the receivers with fairness. Moreover we implement a TCP-friendly application. We investigate the behaviour of the implemented prototype through a number of experiments. Our future work includes the validation of the implemented prototype by using it for the multicast transmission of multimedia data in a heterogeneous group of receivers in the Internet. In addition we have planed to examine the behaviour of the proposed mechanism in very large multicast groups through simulation.

REFERENCES

[1] I. Busse, B. Deffner, and H. Schulzrinne, "Dynamic QoS control of multimedia applications based on RTP", Computer Communications, Jan. 1996.

[2] Shculzrinne, Casner, Frederick, Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, IETF, January 1996.

[3] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet", ACM SIGCOMM 1994, pp. 139--146, London, England, August 1994.

[4] S. McCanne and V. Jacobson, "Receiver-driven layered multicast", 1996 ACM SIGCOMM Conference, pp. 117--130, August 1996.

[5] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," IEEE/ACM Transactions on Networking, 1998.

[6] D. Sisalem, "Fairness of adaptive multimedia applications". ICC '98. 1998 IEEE International Conference on Communications. Conference Record, Affiliated with SUPERCOMM'98 IEEE, 1998.

[7] S. Y. Cheung, M. Ammar, and X. Li, "On the Use of Destination Set Grouping to Improve Fariness in Multicast Video Distribution", INFOCOM 96, March 1996, San Fransisco.

[8] Ch. Bouras, A. Gkamas, "Streaming Multimedia Data With Adaptive QoS Characteristics", Protocols for Multimedia Systems 2000, Cracow, Poland, October 22-25, 2000, pp 129-139.

[9] Ch. Bouras, A. Gkamas, An. Karaliotas, K. Stamos, "An Architecture for Redundant Multicast Transmission Supporting Adaptive QoS" Workshop on Multimedia Information Systems, November 7-9, 2001 - Capri, Italy (to appear).