An Architecture for Redundant Multicast Transmission Supporting Adaptive QoS

Ch. Bouras[1,2] **A. Gkamas**[1,2] **An. Karaliotas**[1,2] **K. Stamos**[1,2]

[1]***Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece***

[2]***Computer Engineering and Informatics Dept., Univ. of Patras, GR-26500 Patras, Greece***

e-mail: {bouras, gkamas, karaliot, stamos}@cti.gr

Abstract

In this paper we describe the architecture and the implementation 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. The fairness problem results from the fact that receivers with different capabilities have to be served. 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. With the term "capabilities" we do not only mean the processing power of the client, but also the capacity and the condition of the network path towards that client. Because of today's Internet heterogeneity and the lack of Quality of Service (QoS) support, the sender cannot assume that the receivers will permanently be able to handle a specific bit rate. We have therefore implemented an additional mechanism for the intra-stream bit rate adaptation.

Keywords: Networking protocols and media delivery, Multimedia distribution and transport, Multicast, Quality of Service, Adaptation mechanisms, CORBA

1 Introduction – Related Work

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 ([20], [13]).
The implementation of adaptation mechanisms in the applications is often criticized. The main arguments that rise against it are that the technologies that are used today for the implementation of the core networks, typically based on ATM (Asynchronous Transfer Mode) technology, provide capabilities to support QoS; as a result the network should offer to the applications QoS guarantees. This is generally true but there is a big problem about it: Today's Internet is divided into thousands of different administration domains. The QoS strategies that are implemented on each one are certainly different, and in many cases no QoS strategy is implemented at all. So the multimedia data flows that have to traverse many of these different domains in order to reach to the end user don't have a sufficient QoS support.
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. Due to the sliding window algorithm that TCP deploys, such flows are the most sensitive in network congestion conditions. Therefore the sliding

window algorithm forces applications to meet some special characteristics in order to be characterized as TCP friendly ([26]).
The system we propose is based on multicast video transmission with the use of RTP/RTCP ([10]). The main perspectives we tried to fulfil are 1) each receiver should receive the best video quality that it is capable of and 2) the 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 (in most network conditions a small number of different streams is enough -typically 3 or 4 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 isn't suitable for it any more another implemented mechanism is used in order to provide the receivers the capability of moving into another stream.
In order to achieve the second goal, we deploy the Additive Increase Multiple Decrease (AIMD) scheme in the inter-stream adaptation algorithm. The adaptation mechanism adapts the rate of each stream taking into account the number of the receivers that are congested or unloaded. In addition, if the capabilities of a receiver aren't suitable for the stream it has joined, it moves to another stream with lower or higher bandwidth limits.
The subject of adaptive streaming of multimedia data over networks has engaged researchers all over the world. The simplest approach for the sender is to use a unicast connection towards each receiver ([1], [2], [3], [5], [6], [14]). This approach is best adapted to each receiver's receiving capability, but has the drawback of making unnecessary use of network resources and cannot therefore scale well into a large number of receivers. In order to overcome the above drawback, someone must use multicast transmission ([23]). 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:

1. The source uses a single multicast session for all receivers ([4], [16],[15], [27]). This results to the most effective use of the network resources, but on the other hand the fairness problem arises.
2. 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 ([17], [21], [23]). The different streams carry the same video information but in each one the video is encoded with different bit rates, and even different video formats (MPEG, H263, JPEG). So each receiver joins in the session that carries the video quality, in terms of bit rate, that is capable of receiving. The main disadvantage in this case is that the same video information is replicated over the network.
3. 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 ([18], [19]). 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 in to 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 [24] and [23]. The rest of this paper is organised as follows: Section 2 presents the architecture of the implemented prototype. In section 3, we give a detailed description of the operation of our prototype application. Section 4 presents some implementation issues. In section 5, we present some initial results in performance evaluation of the implemented prototype. Finally, section 6 concludes the paper and discusses some of our future work.

2 System Architecture

Our system is based on the multicast transmission of video. The advantages that the IP multicast service has, makes it the only choice for transmitting multimedia data, especially when the

application is emulating some kind of broadcasting over the Internet. On the session layer, according to the OSI reference model, the Real Time Protocol (RTP) - Real Time Control Protocol (RTCP) - is used both by H.323 application and MBONE and is broadly considered as the de facto standard for multimedia transmission over the Internet. The RTP-RTCP protocol was also used because of the feedback capabilities that it offers (RTCP reports). At the same time, a unicast session is established between each client and the Server. This session provides the necessary information to the client concerning the multicast IP address as well as the port that is used. So the main multicast data session can be joined. 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.
The main goals that we tried to fulfil are: 1) A quite flexible and fast application, which can react fast to the network changes, 2) adjust its rate in such a way that keeps a friendly behaviour against other TCP or UDP network applications, and 3) achieve the best possible fairness for each receiver based on the idea that fairness for each receiver means to give a slightly worse video quality compared to that the receiver might have if it was alone in the multicast stream. The system architecture consists of two major entities: The server and the client.

2.1 Server Architecture

The server is unique and responsible: 1) to create of the *n* different multicast streams, 2) to set each one's bandwidth limits, 3) to track if there are any clients that are not handled with fairness and 4) to provide the mechanisms to the clients to change stream whenever they consider that they should be in another stream closer to their capabilities. Before the beginning of the transmission the Server entity interacts with the system administrator, with a quite simple graphical user interface, in order for the user to set the desirable values to the main variables of the application. These variables are the number of the streams that will be created, the bandwidth limits of each one, the multicast IP address and the ports, data and control and the video that will be transmitted.

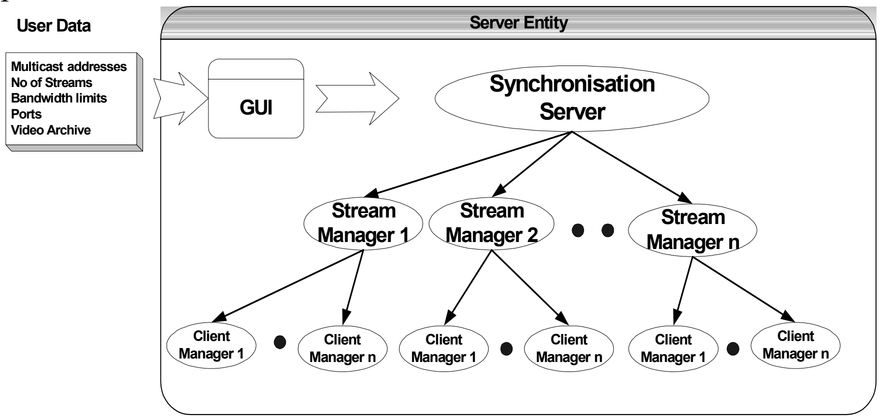

Figure 1 The architecture and the data flow of the Server.

In the above figure the organisation and the architecture of the Server entity is shown. 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 architecture can be further decomposed in the following modules:

- **Synchronisation Server**: Responsible for the management, synchronization and intercommunication between Stream Managers. When a Stream Manager wants to resume the video transmission that was stopped because either all of its clients have left towards another stream or have stopped receiving video, it has to know where the transmission of the video from the other streams currently is. It therefore requests this information from the Synchronisation Server.

- **Video archive:** The hard disks where the video files are stored. Our implementation supports three video formats, namely H.263, MPEG and JPEG.
- **Stream Manager:** The Stream Manager entity is created by the server entity. It 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. These mechanisms will be described later on. A Stream Manager uses the information provided by Client Managers to adjust its data rate in order to achieve optimum overall results. A Stream Manager can only transmit when it has at least one client participating in its multicast session. The Stream Manager entity contains the following modules.
  - <u>Optimal Rate Estimation</u>: The Feedback Analysis module (described later in detail) contained in every Client Manager processes the RTCP reports from the Client and declares the Client to be in one of three states: CONGESTED, LOADED or UNLOADED. The Optimal Rate Estimation module periodically gathers the states reported by all Client Managers belonging to its Stream Manager at the end of a specific, fixed time period (from now on called an epoch). It then uses an algorithm described in a following paragraph that tries to improve fairness between clients by determining whether a lower or a higher bit rate is more appropriate. Whenever a client cannot be satisfied by a stream due to the fact that most of the other receivers have much higher or much lower reception capabilities, the Optimal Rate Estimation module informs it that it has to move to a lower or higher quality stream.
  - <u>Quality Adaptation</u>: Responsible for the adaptation of the video transmitting quality. Its behaviour (whether to adapt the transmitting quality upwards or downwards) is determined by the Optimal Rate Estimation module. Due to the fact that the new bit rate depends on the value of the current one, the Quality Adaptation module computes the current bit rate by dividing the amount of bytes sent from the end of the last epoch until the end of this epoch, by the time period that is represented by an epoch. The other alternative would be to make no manual computation and to assume that the current bit rate is the one that the module tried to establish at the end of the previous epoch. This second approach has the decisive drawback however, that there is no guarantee that the previously intended bit rate was actually achieved by the encoder and sent to the network. In fact, practice showed that sometimes the actual bit rate achieved varies significantly compared to the bit rate the Quality Adaptation module intended to achieve. Whether there is a big difference between the bit rate requested by the Quality Adaptation module and the actual bit rate, mainly depends on the encoder and the encoding format, as well as the capabilities of the server host.
  - <u>Packet scheduler/ Server buffer</u>: This module sends all outgoing information to the network. Its responsibility is to encapsulate data in RTP packets. There is also a buffer used to smooth accidental problems during the network transmission.
- **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. It can interact only with one Stream Manager at a given time, the Stream Manager controlling the stream from which the client is receiving the video. It contains the following component:
  - <u>Feedback analysis/Report buffer</u>: This module 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. The exact operation of the algorithm is described in a following paragraph.

2.2 Client Architecture

The client architecture consists of the following modules:

- **Client buffer**: Multimedia data received are first stored in this module, and presentation does not begin unless there is a necessary amount of data stored in the client buffer. In order to achieve smooth media presentation to the user, this buffer's capacity has to exceed the maximum delay jitter during data transmission.

- **Feedback**: This is the module that produces the information necessary for the Feedback Analysis Module at the server to estimate the client's state. Control information is transmitted with RTCP reports, which include, as mentioned earlier, information about packet loss rate and delay jitter.
- **Decoder**: This module takes the data packets from the client buffer as input, decodes them and outputs them suitable for presentation. The quality of the presented video is higher when the receiving data rate is high. Video quality can also be affected by packet loss and delay. Presentation can come to a complete stop if data in the client buffer drops below the required minimum
- **User Display**: The module responsible for the presentation of the video to the user, which can be a computer monitor.

3 Description of System Operation and Algorithms

The source initially constructs a number of streams. This number depends on the number of receivers with different reception capabilities that expected to request video from the source and the processing capabilities of the machine where the server runs. Since in this implementation this number is determined at the initialisation of the server, an estimation of the actual number of receivers is useful. A small pre-determined number can, however, work well in most cases, because it offers the possibility of a reasonable categorization of all receivers according to their capabilities.
Each stream has to do its own processing on the video so as to transmit it at its prescribed rate, therefore many streams mean heavy processing load for the server. A solution to this problem, apart from using a small number of streams, could be storing the video in various (or all) needed encoding types so that each stream can use its own locally stored file. This, of course, has the drawback of taking up more disk space. Since a small number of streams were considered satisfying for our experiments, our implementation performs encoding on the fly. A stream without any clients in its session is ready for transmission but remains inactive.
When a client wants to start receiving video, it requests from the server the address of a multicast session belonging to a transmitting stream. This and all other Client – Server communication is made through the use of CORBA (Common Object Request Broker Architecture). Since the client user can have some local knowledge over its connection quality, he can make an initial estimation of the stream in which the client might best fit and is allowed to enter any stream the user chooses. If no choice is made, then by default the client enters the lowest quality stream. An overestimation can be expected to have small negative effects, since the client will be moved shortly to a more suitable stream.
By joining a multicast session the client informs the stream to start transmitting, if it is not doing so already. 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's Feedback Analysis module. 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:

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

Where: LR_{new}: The new filtered value of packet loss rate, LR_{old}: The previous filtered value of packet loss rate. For the first report after the start of transmission, this value is 0, 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, with a=0 meaning that only the current report is taken into account, and a = 1 meaning that all new RTCP reports are ignored.
For delay jitter:

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

Where: J_{new}: The new filtered value of delay jitter, J_{old}: The previous filtered value of delay jitter. For the first report after the start of transmission, this value is 0, 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, with b=0 meaning that only the current report is taken into account, and b = 1 meaning that all new RTCP reports are ignored.

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:

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

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 Optimal Rate Estimation module of 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. This buffering mechanism contributes to the conservative behaviour of the Optimal Rate Estimation module. 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 is presented below:

We first introduce a new variable, USV (Unprocessed State Variable), that takes a new value for each unprocessed state as shown:

$$\textit{if } (\textit{unprocessed state}_i = = CONGESTED) \textit{ then } USV_i = -1$$
$$\textit{if } (\textit{unprocessed state}_i = = LOADED) \textit{ then } USV_i = 0$$
$$\textit{if } (\textit{unprocessed state}_i = = UNLOADED) \textit{ then } USV_i = 1$$

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.

$$\textit{if } (f(i) < 0) \textit{ then processed state}_i = CONGESTED$$
$$\textit{if } (f(i) = = 0) \textit{ then processed state}_i = LOADED$$
$$\textit{if } (f(i) > 0) \textit{ then processed state}_i = UNLOADED$$

Information update in Client Managers is made asynchronously, every time an RTCP report arrives. However, 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 the Quality Adaptation module 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 0.75) and so the algorithm is more aggressive when trying to react to congestion.

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

There are three cases in this phase that will lead to a client's transition towards another stream:

- 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).
- 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).
- 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 in 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. Transition from one stream to another also means that the client's corresponding Client Manager module will now interact with the new Stream Manager.

The conservatism our protocol exhibits has two advantages: (1) We successfully ignore RTCP reports that are the result of temporary factors, for example congested reports that appear almost certainly when a stream transition occurs and are due to system load but have a very short effect on the reception capability. (2) Our protocol is TCP-friendly, because it only consumes excessive bandwidth when it is absolutely certain that this bandwidth can be handled, and furthermore uses the conservative AIMD algorithm.

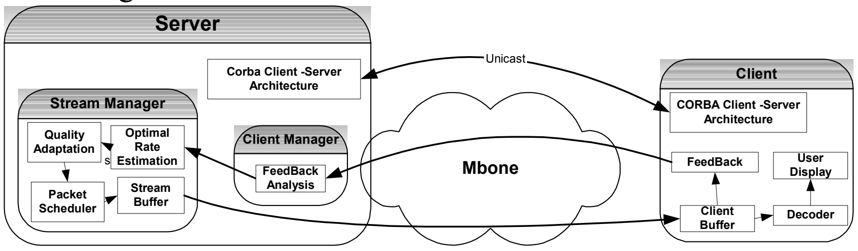

Figure 2 The operation of the application.

4 Implementation Issues

For the implementation of our system we used the Java Programming Language, and in particular the Java Media Framework API ([12]). Java's object-oriented model fits our design and JMF offers a convenient level of abstraction, which allows the developer to concentrate on high-level issues, thus making it an ideal platform for experimental research. In particular, JMF provides support for RTP transmission and reception of real-time media streams across the network. It offers some very useful classes and interfaces, like the Session Manager, that encapsulates the creation, maintenance and closing of an RTP session, the Processor that encapsulates processing and control of time-based media data and the DataSource that encapsulates media protocol-handlers. Our JMF-based implementation is represented by the Figure 3 and Figure 4. The Figure 5 shows the Graphical User Interface of the Client.

All communication between the server and the clients is achieved using CORBA. This technology allows a module written in any language that supports CORBA to be integrated seamlessly in our system, as long as it implements a small number of functions necessary for remote communication.

CORBA communication between the Server and the Clients also requires a third entity, the Naming Service. It can be located on the same host as the Server or on any other host in the network. All clients and the server, however, must know its location. When the Server is initialised, it registers itself and all the Stream Managers it creates to the Naming Service using a hierarchical representation similar to an operating system's file structure. When a client is started it uses the Naming Service to request a reference to the Stream Manager it wishes to receive data from. Every time the client makes a transition to a different stream, it uses the Naming Service to get a reference to the new Stream Manager. Since communication may also be directed from the Client to the Server, during initialisation every Client also registers itself to the Naming Service. This way the

Client Manager module (which is part of the Server entity) can locate its corresponding Client and order it to move to a different stream whenever necessary.
These choices generally indicate our purpose for this implementation to be experiment- and flexibility- oriented, rather than performance-oriented and therefore it can be improved in terms of resource optimisation.

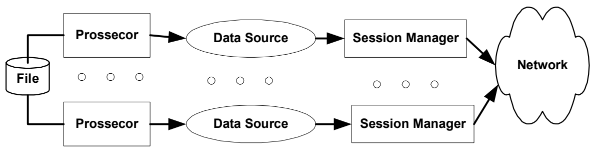

Figure 3 The Server operation

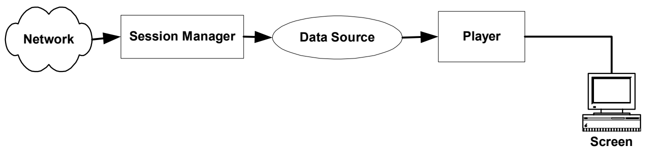

Figure 4 The Client operation

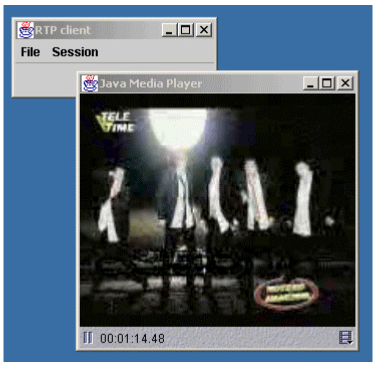

Figure 5 Client GUI

5 Performance Evaluation – Initial results

In order to evaluate the performance of the implemented prototype, we run an experiment over a controlled networking test-bed, which, we have implemented over the campus network of University of Patras in Greece. The test-bed consists of one Server and six Clients. We connect each participant with connections of different capacity to our network test-bed with the use of traffic policy on the access router of each participant (the Server and the six Client) in the test-bed. More particularly, the Server is connected to our test-bed with 2 Mbps connection and the Clients are connected with connections, which vary from 200-700 Kbps. The Server was transmitted three streams with the following limits: Stream 1: 10Kbps-184Kbps, Stream 2: 184Kbps-368Kbps and Stream 3: 368Kbps-600Kbps.During the experiment the Server was using the following parameters in order to control the operation of the implemented mechanism: $a=0.5$, $b=0.8$, $\gamma=2$, $LR_u=0.02$, $LR_c=0.05$ (the values of the above parameters base on experimental results). The AIMD algorithm of the Server was increasing the transmission rate of a Stream by 50Kbps during network unloaded periods and was decreasing the transmission rate to 75% during network congestion periods. We run the experiment for 360 sec and in the begging of the experiment the Server transmits only the Stream 1 with transmission rate of 10Kbps and all the Clients are connected to Stream 1.
Figure 6 shows the transmission rate of the Server Streams during the experiment and the Figure 7 shows the receive rate of a representable Client which is connected to the network test-bed with 400 Kbps connection.
As the Figure 6 shows, in the beginning of the experiment all the Clients are connected to Stream 1 and the transmission rate of Stream 1 increase until it reach its upper limit. Then some Clients switch to Stream 2 and the Server starts transmit the Stream 2 (at 60th second) except of the Stream 1. After some time some Clients switch to Stream 3 (at 100th second) and the Server transmits all the three streams. During the experiment, the Server stops a stream if the stream does not have any

receivers. As the Figure 7 shows, the representable Client starts receiving Stream 1 from the Server and when this Stream reach its maximum capacity, the Client switch to Stream 2 in order to exploit the available bandwidth. The switching from Stream 1 to Stream 2 takes some time to complete because the join action and especially the leave action of a multicast group take some time to complete.
The above described experiment can only be consider as an initial performance evaluation of the implemented prototype and shows predictable operation of the implemented prototype and has encouraging results. We plan to investigate in more detail the behaviour the implemented prototype in our future work.

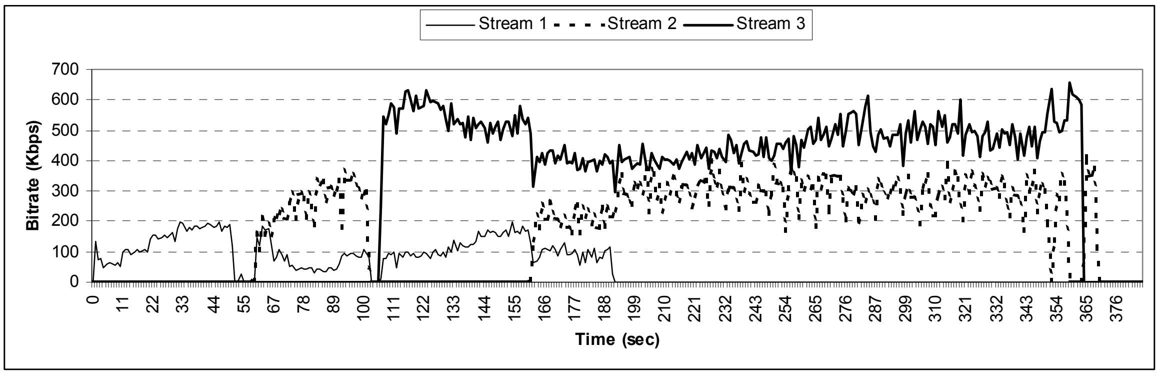

Figure 6 Transmission Rate of Server Streams

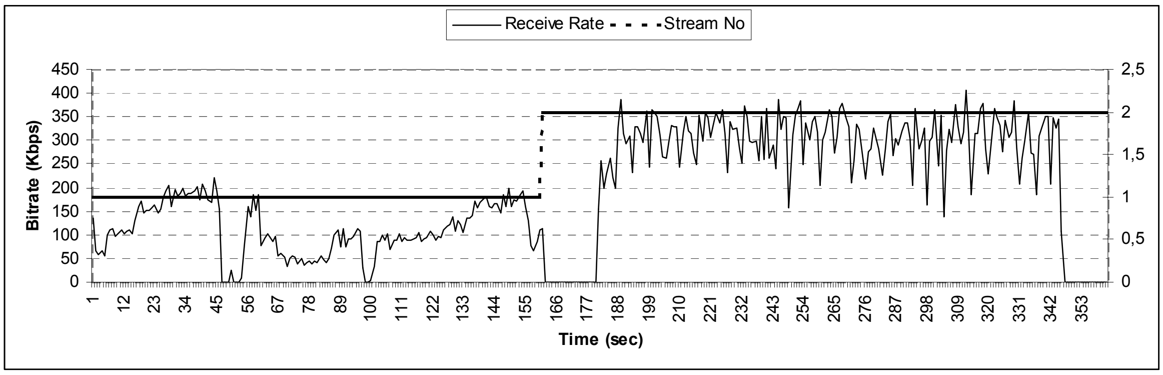

Figure 7 Receive Rate of representable Client

6 Conclusion - Future Work

In this paper, we present the implementation 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 streams and treat the receivers with fairness. The implemented prototype uses RTP/RTCP protocols for the transmission of multimedia data.
Our future work includes the validation of implemented prototype by using it for the multicast transmission of multimedia data in a heterogeneous group of receivers both in a controlled network test-bed and in the Internet in order to examine the behaviour of the implemented prototype against TCP and UDP traffic. In addition we plan to examine the behaviour of the proposed mechanism in very large multicast group through simulation. In addition we will investigate the benefits of dynamically adding more streams instead of the static number of streams that the implemented prototype supports now.

7 Bibliography

[1] C. Cowan, S. Cen, J. Walpole, C. Pu. "Adaptive Methods for Distributed Video Presentation", ACM Computing Surveys, 27(4), pp. 580-583, December 1995. Symposium on Multimedia.
[2] R. Rejaie, D. Estrin, and M. Handley, "Quality Adaptation for Congestion Controlled Video Playback over the Internet" in Proc. of ACM SIGCOMM '99, Cambridge, Sept. 1999.

[3] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, L. Yu, "A player for adaptive mpeg video streaming over the internet," in Proceedings of the 26th Applied Imagery Pattern Recognition Workshop AIPR-97, SPIE, (Washington DC), Oct. 1997.

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

[5] S. Jacobs, A. Eleftheriadis, "Adaptive Video Applications for Non-QoS Networks", Proc. 5 th International Workshop on Quality of Service (IWQoS'97), Columbia University, New York, USA, pp.161-165.

[6] R. Ramanujan, J. Newhouse, M. Kaddoura, A. Ahamad, E. Chartier, K. Thurber, "Adaptive Streaming of MPEG Video over IP Networks", Proceedings of the 22nd IEEE Conference on Computer Networks (LCN'97), November 1997.

[7] P. Mundur, A. Sood, R. Simon, "Network Delay Jitter and Client Buffer Requirements in Distributed Video-on-Demand Systems", Department of Computer Science George Mason University Fairfax, VA 22030.

[8] S. Cen, C. Pu, J. Walpole, "Flow and Congestion Control for Internet Media Streaming Applications", In Proceedings of Multimedia Computing and Networking, 1998.

[9] B. Vandalore, W. Feng, R. Jain, S. Fahmy, "A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia", Journal of Real Time Systems (Special issue on Adaptive Multimedia), April 99.

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

[11] Shculzrinne, Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, IETF, January 1996.

[12] Java Media Framework: http://java.sun.com/products/java-media/jmf/index.html

[13] R. Rejaie, M. Handley, D. Estrin. "Architectural considerations for playback of quality adaptive video over the Internet", Technical Report 98-686, USC-CS, November 1998.

[14] R. Rejaie, M. Handley, D. Estrin. "RAP: An end-to-end rate-based congestion control mechanism for real time streams in the Internet", Proc. IEEE Infocom, March 1999.

[15] H. Smith, M. Mutka, D. Rover, "A Feedback based Rate Control Algorithm for Multicast Transmitted Video Conferencing", Accepted for publication in the Journal of High Speed Networks.

[16] J.-C. Bolot, T. Turletti, I. Wakeman. "Scalable feedback control for multicast video distribution in the Internet" In Proceedings of SIGCOMM 1994, pp. 139-146, London, England, August 1994. ACM SIGCOMM.

[17] T. Jiang, E. W. Zegura, M. Ammar, "Inter-receiver fair multicast communication over the Internet". In Proceedings of the 9th International Workshopon Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), pp. 103-114, June 1999.

[18] B. J. Vickers, C. V. N. Albuquerque T. Suda, "Adaptive Multicast of Multi-Layered Video: Rate-Based and CreditBased Approaches," Proc. of IEEE Infocom, March 1998

[19] S. McCanne, V. Jacobson. Receiver-driven layered multicast. 1996 ACM Sigcomm Conference, pp. 117-130, August 1996.

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

[21] T. Jiang, M. Ammar, E. W. Zegura, "Inter-Receiver Fairness: A Novel Performance Measure for Multicast ABR" Sessions. pp. 202-211 SIGMETRICS 1998

[22] X. Li, M. Ammar, S. Paul, "Video Multicast over the Internet", IEEE Network Magazine, April 1999

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

[24] 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.

[25] C. Diot - Sprint Labs "On QoS & Traffic Engineering and SLS-related Work by Sprint", Workshop on Internet Design for SLS Delivery, Tulip Inn Tropen, Amsterdam, The Netherlands, 25 - 26 January 2001.

[26] J. Pandhye, J. Kurose, D. Towsley, R. Koodli, "A model based TCP-friendly rate control protocol", Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Basking Ridge, NJ, June 1999.

[27] D. Sisalem, A. Wolisz, "LDA+ TCP-Friendly Adaptation: A Measurement and Comparison Study," in the 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'2000), June 25-28, 2000, Chapel Hill, NC, USA