A Multimedia Streaming Server Employing Open Standards

Rediet Ashne
HiLCoE School of Computer Science and Technology, Addis Ababa, Ethiopia
redashne@gmail.com

Workshet Lamenew
School of Information Science, Addis Ababa University, Ethiopia
workshet@gmail.com

Abstract

The exponential growth of the Internet, both in terms quality and accessibility, has made it indispensable in our day to day life. This has dramatically changed the large portion of the contents transported over the Internet from simple text messages into multimedia contents of high quality over the years. Streaming applications has played a crucial role in this transition because of their unique characteristics which favors multimedia contents.

However, most of the applications deployed are built based on proprietary technologies which are close to the public. This paper tries to build a streaming server which is built on specifications and technologies which are open to the public and standardized by an internationally recognized body. It has been tried to enhance the quality of service of the application by integrating a rate control and error recovery mechanism with the multimedia streaming server.

Keywords: Real Time Streaming Protocol; Real Time Transport Protocol; Rate Control; Forward Error Control

1. Introduction

There are various ways to transmit multimedia data over a network, like dedicated links, cables, and ATM. However, the idea of running multimedia over the Internet is extremely attractive. Internet is growing exponentially. The well established LAN and WAN technologies based on the IP protocol suite connect bigger and bigger networks all over the world to the Internet. In fact, the Internet has become the platform of most networking activities. This is the primary reason to develop multimedia protocols over the Internet. Another benefit of running multimedia over IP is that users can have integrated data and multimedia service over one single network, without investing on other network hardware and building the interface between two networks.

However, multimedia networking is not a trivial task. We can expect at least three difficulties. First, compared with traditional textual applications, multimedia applications usually require much higher bandwidth. Second, most multimedia applications require real-time traffic. Audio and video data must be played back continuously at the rate they are sampled. If the data does not arrive in time, the playing back process will stop and human ears and eyes can easily pick up the artifact. Third, multimedia data stream is usually bursty. Just increasing the bandwidth will not solve the burstiness problem. For most multimedia applications, the receiver has a limited buffer. If no measure is taken to smooth the data stream, it may overflow or underflow the application buffer. These issues have to be addressed in order to fully deploy a multimedia application in an IP- based network [1].

Conventional downloading applications (e.g., file transfer such as FTP) involve downloading a file before it is viewed or consumed by a user. Downloading is usually a very robust way to deliver media to a user. However, downloading has two potentially important disadvantages for multimedia applications. First, a large buffer is required whenever a large media file is downloaded. Second, the amount of time required for the download can be relatively large, thereby requiring the user to wait minutes or even hours before being able to consume the content.

An alternative to downloading is streaming. Streaming applications split the media bit stream into separate chunks (e.g., packets), which can be transmitted independently. This enables the receiver to decode and play back the parts of the bit stream that are already received. The transmitter continues to send multimedia data chunks while the receiver

decodes and simultaneously plays back other, already received, parts of the bit stream.

2. Related Works

Currently the Internet is playing a key role in transferring diverse information. As the performance of the Internet has been growing, transferred information has the form of multimedia streams such as video and audio. However, in the current computing and Internet environment, multimedia streaming service can't be accomplished smoothly due to the various factors such as Internet bandwidth, computing power, and so on. An efficient system, called SMART (Server for Multimedia Application for Residence Community), was proposed in [9]. A multimedia file system is needed that can utilize the facilities of the NS card and support large volume of data. Therefore, the authors developed a new multimedia file system called EXT3NS file system. EXT3NS provides applications with the standard read and write system call interfaces to use NS card. In addition to fast-path I/O operation with NS card, legacy buffered I/O is supported as well. Metadata is stored and accessed on a 4KB basis, but the unit of allocation for data blocks depends on the configuration of the NS card.

In a research project called Massively-parallel And Real-time Storage (MARS), it has been tried to design and implement a large scale multimedia storage server [10]. They have used such as data striping and Redundant Arrays of Inexpensive Disks (RAID) for parallel I/O operations and fault tolerance. ATM based interconnect has also been used to achieve a scalable architecture that transparently connects storage devices to an ATM-based broadband network. The architecture relies on innovative data striping and real-time scheduling to allow a large number of guaranteed concurrent accesses, and uses separation of metadata from real data to achieve a direct flow of the media streams between the storage devices and the network.

In order to ensure a large throughput from the entire system, the MARS server physically distributes the data for a stream among a subset of the storage nodes. The metadata information associated with the data is also distributed among various storage nodes and the central manager. In addition to providing storage service, each storage facility supports one or more of the following functionalities: File system support, Admission control, Scheduling support, and Compute support.

In a research paper [11], Adaptive Distributed Multimedia Server (ADMS) architecture that builds upon the idea of offensive adaptively was presented. The server proactively controls its layout through replication or migration of server components to recommended hosts. They have finally evaluated their architecture in a real world streaming scenario.

Architecture of modular streaming media server for content delivery networks was proposed in [12]. It describes some of the requirements and architectures for such servers. The design goals for the system were scalable content delivery, responsiveness to changing conditions, efficient resource utilization, and live video support. They have suggested that the following features can help in making the streaming server scalable: caching and segmentation support, modular design, and intelligent scheduling.

3. Proposed Solution

Existing technologies and specification for session description, session control, media transport, rate control, and error correction will be reviewed and assessed. The properties of these mechanisms will make them suitable for some class of applications and not desirable for others.

The effectiveness of the rate control and error recovery mechanism on the quality of service of the application will be evaluated. It will then be compared with quality of service of the system when it does not employ these mechanisms. This will be done by running the client and the server on different machines connected through a network.

The streaming server will serve multiple users at a time simultaneously. It will create a session for each client and then all the communication and data flow will be through it. The server will keep a state for each session it has with a client. The client will have to create a new session every time it wants to access a new multimedia content.

The RSTP server can assume the following states:

- *Init*: The initial state, no valid SETUP has been received yet.
- *Ready*: Last SETUP received was successful, reply sent or after playing, last PAUSE received was successful, reply sent.
- *Playing*: Last PLAY received was successful, reply sent. Data is being sent.

The current RSTP specification defines 11 methods for implementation. From these 11 methods, the following six have been implemented: OPTIONS, DESCRIBE, SETUP, PLAY, PAUSE, and TEARDOWN.

3.1 Media Transport protocol

This is the protocol that sits above the transport level protocols like TCP and UDP. It will encapsulate the raw packet or datagram with some header information so that the real time behavior of the streams will be taken in to account.

Table 1 reviews the characteristics of widely used media transport protocols. It uses the following parameters for comparison.

- Openness of the specification to the public
- Specification standardized by an international body
- Favor timeliness over reliability

Table 1: Comparison of the widely used media transport protocols

Media transport protocols	*Openness of Specification*	*Standardization of specification*	*Favor timeliness over reliability*
Real Time Protocol (RTP) [6]	Yes	Yes	Yes
Real Time Messaging Protocol (RTMP) [7]	No	No	No
HTTP Live Streaming [8]	No	Yes	No

RTP will be used as the media transport protocol for this work as it is more appropriate than any other media transport protocol.

3.2 Error Correction

Error control is an application-specific mechanism that is exclusively responsible for recovery of lost packets. Streaming applications require quality instead of reliability. The server should allocate portion of the available bandwidth for error control.

Table 2 reviews the characteristics of error recovery mechanisms. It uses the following parameters for comparison.

- Favor timeliness over reliability
- Low end to end delay

Table 2: Comparison of the packet recovery mechanism

Error recovery mechanisms	*Low end to end delay*	*Favor timeliness over reliability*
Forward error correction (FEC) [2, 3]	Yes	Yes
Partial checksum [3]	Yes	Yes
Retransmission [3]	No	No

The quality of packet voice on the Internet has been mediocre due, in part, to high packet loss rates. This is especially true on wide area connections. Unfortunately, the strict delay requirements of real time multimedia usually eliminate the possibility of retransmissions. It is for this reason that forward error correction (FEC) has been proposed to compensate for packet loss in this system.

3.3 Rate Control

Typically, for streaming video, congestion control takes the form of rate control. Rate control attempts to minimize the possibility of network congestion by

matching the rate of the video stream to the available network bandwidth [3].

Table 3 reviews the characteristics of two rate control mechanisms. It uses the following parameters for comparison.

- Favor timeliness over reliability
- Low end to end delay
- Low jitter

Table 3: Comparison of rate control mechanisms

Rate control mechanisms	*Low end to end delay*	*Favor timeliness over reliability*	*Low Jitter*
TCP–like Congestion Control [4]	No	No	No
TCP-Friendly Rate Control [TFRC] [3, 5]	Yes	No	Yes

TCP-friendly rate control (TFRC) has been employed by the streaming server in order to reduce the congestion that might be created by using UDP as a transport protocol instead of TCP. This model-based approach is based on a throughput model of a TCP connection.

The server has the following logical subsystems which were grouped based on the similarity of the functionalities of individual operations within each subsystem. The subsystems are: Request message processor, Response message processor, Client management, Session management, Generate packet, Buffers, Timers, Packet sender.

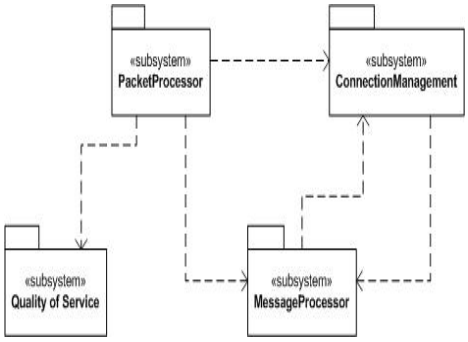

Figure 1: Subsystem Decomposition

4. Prototype

The prototype of the streaming server was built using C# programming language and .NET framework. The testing of the functionalities of the server was done in a personal computer running Windows 7 operating system.

4.1 Creating a Connection

Figure 2: Server's response for connection request

Figure 3: Server's processing of connection request

The server creates a new thread for each client connected. Thread 3 and thread 4 are used to communicate clients. Thread 1 is used by the server to listen to other connection requests from clients.

The execution of this method creates a new session for the connection and then associates it with the unique and randomly generated session id. It also changes the state of the session from Init to Ready. A Status code of '200' tells the client that its request has been processed without any error.

4.2 Client sending PLAY Request

Figure 4: Client's PLAY request

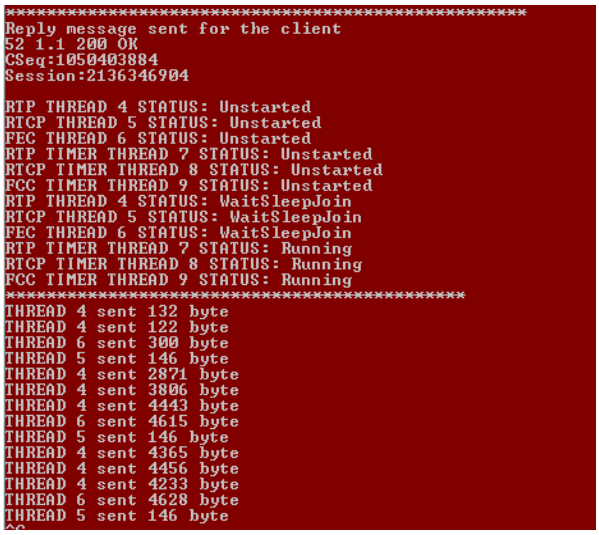

Figure 5: Server's response to PLAY request

This method keeps on sending data until the client issues a pause/teardown request or it reaches the end of the range. Threads 4, 5, and 6 send RTP data, RTCP data, and FEC data, respectively while threads 7, 8, and 9 are used as timers for the sending threads.

4.3 Adjusting Rate of Streaming

Figure 6 shows the effects of employing a rate control mechanism on the streaming server. It measures the number of packets lost in the client when the server uses a rate control and when it is not.

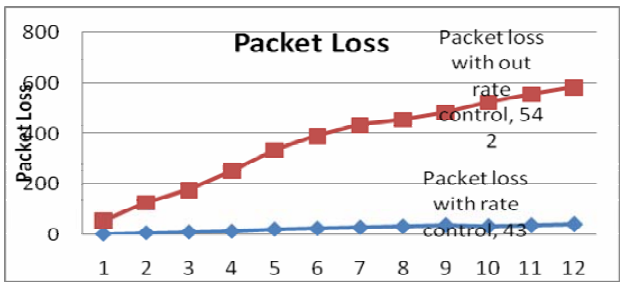

Figure 6: Effect of rate control on packet loss

As it can be seen from Figure 6, the number of packets lost increases in a close to linear fashion when the rate of sending is not controlled. But when the rate control mechanism is employed, the lost fraction becomes almost constant. This constant lost rate means the client will receive a steady flow of streams which improves the viewing experience of the user.

4.4 Recovering Lost Packets

Figure 7 shows the number of lost packets recovered using the lost packet recovery mechanism employed by both the client and the server. It can be seen that the number of recovered packets increases linearly. This will improve the viewing experience of the user by recovering lost packets due to network conditions.

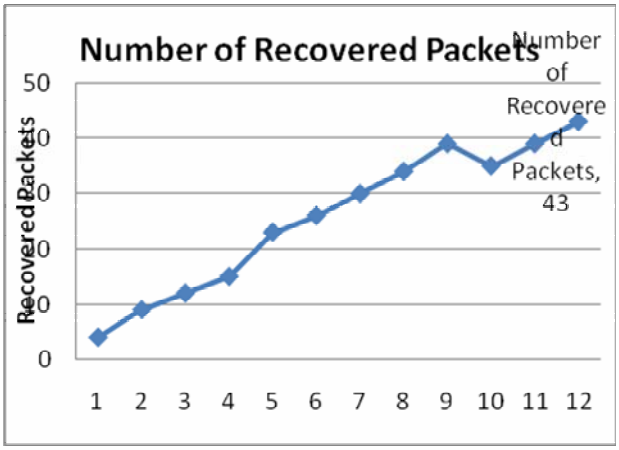

Figure 7: Number of Recovered Packets

5. Discussion

The overall objective of this paper has been producing a media streaming server prototype which is completely based on an open and standard specification. RSTP, the session control mechanism, is implemented. It provides a network remote control like behavior which is suitable for media on demand applications.

RTP has been used for carrying the audio/video data and it provides services that allow the client the correct playback of the multimedia file being streamed by the server. The RTP operation works in conjunction with its control protocol RTCP which provides feedbacks on the quality of distribution for both the client and the server.

Finally the server utilizes threads extensively for serving multiple users at a time as well as executing different operations while communicating with a single user. However, deciding on the number of

users the server can serve at a time was beyond the scope of this work and is left as future work. It will be dependent on the hardware and the available network bandwidth.

References

[1] Feng Wu, Honghui Sun, Guobin Shen, Shipeng, Bruce Lin, Ming-Chieh Lee, and Ya-Qin Zhang, "An Efficient, Scalable, and Robust Streaming Video System," EURASIP Journal on Applied Signal Processing, Vol.2, pp. 192–206, 2004.

[2] J. Rosenberg and H. Schulzrinne, "RFC 2733: An RTP Payload Format for Generic Forward Error Correction," The Internet Society: Network Working Group, December 1999.

[3] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and Jon M. Peha, "Streaming Video over the Internet: Approaches and Directions," IEEE Transactions on Circuit and Systems for Video Technology, Vol. 11, No. 1, February 2001.

[4] Robert Shorten, Fabian Wirth, and Douglas Leith, "A Positive Systems Model of TCP-like Congestion Control: Asymptotic Results," White Paper, April 7, 2004.

[5] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "RFC 3448: TCP Friendly Rate Control (TFRC)," The Internet Society: Network Working Group, January 2003.

[6] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RFC 3550: RTP: A Transport Protocol for Real-Time Applications," The Internet Society: Network Working Group, July 2003.

[7] H. Parmar, and M. Thornburgh, "Real Time Messaging Protocol (RTMP)," Adobe Systems Incorporated, December 2012.

[8] R. Pantos, and W. May, "HTTP Live Streaming," Apple Inc., October, 2012.

[9] Yuhyeon Bak, Kapdong Kim, Youngju Lee, Songwoo Sok, Changsoo Kim, Hagyoung Kim, Myung-Joon Kim, and Kyongsok Kim, "High Performance Internet Server for High Definition Streaming Service," International Journal of Smart Home, Vol. 1, No. 1, January 2007.

[10] Milind M. Buddhikot, Guru M. Parulkar, and Jerome R. Cox, Jr., "Design of a Large Scale Multimedia Server", White Paper, November 14, 1994.

[11] Roland Tusch, Christian Spielvogel, Markus Kröpfl, László Böszörměnyi, "An Adaptive Distributed Multimedia Streaming Server in Internet Settings", Institute of Information Technology, Klagenfurt University, Austria.

[12] Sumit Roy, John Ankcorn, and Susie Wee, "Architecture of a Modular Streaming Media Server for Content Delivery Networks", Hewlett-Packard Laboratories, Palo Alto.