A Monitoring Module for a Streaming Server Transmission Architecture

Sadick Jorge Nahuz¹, Mario Meireles Teixeira², and Zair Abdelouahab¹

¹ Graduate Program in Electrical Engineering, Federal University of Maranhao,
Campus do Bacanga, 65085-580 Sao Luis, MA, Brazil

² Department of Informatics, Federal University of Maranhao,
Campus do Bacanga, 65085-580 Sao Luis, MA, Brazil

sadicknahuz@gmail.com, mario@deinf.ufma.br, zair@dee.ufma.br

Abstract. The Internet has experienced a considerable increase in the use of audio and video applications, which provoke a large consumption of the resources available in the network and servers. Therefore, the monitoring and analysis of those resources has become an essential task in order to enhance the service delivered to users. This work depicts a monitoring module implemented in a video server architecture, which is used to track the transmission of some popular video formats. Our experiments have demonstrated that one of the formats delivers a performance considerably better than the other, regarding the bandwidth allocated to each user session, what reassures the importance of having such a monitoring module available in a server's architecture.

Keywords: monitoring, qos, streaming server, mpeg-4, mov.

1 Introduction

In recent years there has been an amazing development and wide spreading of network applications that transmit and receive audio and video data through the Internet. Those applications have service requirements different from those of traditional data oriented applications, based in text/images, e-mail, FTP and DNS [1].

Today, it is observed a great demand for a class of Internet applications known as Multimedia Systems. This class of systems differs from others because they need to transmit multimedia data such as video and audio, generating a large data stream, provoking an accentuated consuming of the Internet resources.

Multimedia systems supply services oriented towards the execution of applications that need resources assurance so as not to hinder its performance.

Multimedia systems are very sensitive to end-to-end delays and to delay variation, but they can tolerate occasional data losses [1]. Some current networks can not get adapted to multimedia services because they were not designed to stand such tasks. There are many attempts for expanding the networks current architecture in order to enhance the services quality and to provide support for multimedia applications.

Currently, almost all news sites have a video format for their subscribers. In the next years, multimedia services will dominate a great part of the Internet flow, mainly the live transmission ones, but there is much to be researched and defined in that area [2].

As high speed Internet access reaches an increasing number of households, multi-media services will become part of most users' routine [9]. Everybody will access their favorite programs, irrespective of their transmission scheduling, once there will always be a file stored in a streaming server, which uses a real time transmission protocol.

This paper details a monitoring module designed and implemented in the Darwin Streaming Server, from Apple. This module is used to evaluate this architecture serving both MOV and MP4 files. Transmission peculiarities of both file formats are discussed and the monitoring demonstrates that one of them is able to reach considerable higher throughput than the other.

In Sec. 2 are discussed the protocols used for streaming media transmission in the Internet. Sec. 3 details the streaming server chosen for this work, namely the Darwin Streaming Server. Sec. 4 deals with the monitoring implemented in the streaming server and also the changes performed in it. In Sec. 5 are discussed the undergone experiments and their results. Sec. 6 discusses this work main conclusions and their possible unfolding.

2 Streaming Media Protocols

Real-time applications do have some peculiarities, as commented above. Thus one might expect that they demand protocols specifically designed to attend their characteristics. This section analyzes the Internet protocols used for streaming media transmission: RTP, RTCP and RTSP.

2.1 RTP and RTCP Protocols

The real-time transport protocol, known as RTP, was designed to provide support to applications traffic that broadcasts real-time data. It is normally integrated inside the application (in user mode), so it is not implemented as part of the operating system kernel. Defined in RFC 1889 [3], RTP protocol is a product of the Audio/Video Transport Working Group, and it was formally introduced in January, 1996, by the Networking Working Group) from IETF (Internet Engineering Task Force), aiming at supplying a standardization of functionalities for real-time data transmission applications. [22].

The RTP offers end-to-end transport functions for devices that transmit real-time data, such as audio and video, on unicast or multicast service networks, being characterized as a connectionless protocol. These functions include the identification of the type of data to be handled, sequence number, timestamps and data transmission monitoring. Although the RTP offers end-to-end transfers, it does not offer all functionalities usually found in a transport protocol. Besides that, it does not reserve resources from the network and does not guarantee Quality of Service (QoS) [10] for the applications, neither promotes reordering or re-transmission in case of packet loss, assigning these responsibilities to applications [11].

The RTP and the RTCP protocols were designed as not to depend on the underlying network and transport layers. While RTP is in charge of transportation of the streaming medias (audio and video), the RTCP controls the information returned by

users that received the data, informing about the quality of reception and data transfer, the support and synchronization of different media flows.

2.2 RTSP Protocol

The RTSP (Real-Time Streaming Protocol) is based upon the class “Stream Stored audio and video”, operates in the application layer and was conceived by Real Networks, Netscape Communications and Columbia University. Its first RFC was published by the IETF under the number 2326 [4].

RTSP is a public domain protocol that allows client-server interaction between the constant rate media stream source (server) and the user (transducer) [23]. That interactivity arises from the user’s necessity of having a higher control on the media reproduction in the transducer. The RTSP functionalities can be summarized as handling of the file execution, similar to the functionalities that a DVD player turns available to their users, the RTSP allows a transducer to control the media stream by means of commands.

2.3 Streaming Media Formats

MPEG-4 is the global multimedia standard that broadcasts professional quality audio and video through various types of bandwidth, from cellular phone networks to WANs and others. MPEG-4 was defined by the Moving Picture Experts Group (MPEG) [12] [14], which actively participates in the International Organization for Standardization (ISO), which specified the well known MPEG-1 and MPEG-2 standards. Hundreds of researchers worldwide contributed in the MPEG-4 building [15], which was concluded in 1998, just to become an international standard in 2000.

MOV, another quite popular multimedia format, is used to store video sequences and was created by Apple Computers. The QuickTime Player was presented by Apple as an alternative to the Windows Media platform, from Microsoft, betting in favor of the variety of formats available for content distribution. This video format became still more popular when their specifications were chosen by the MPEG consortium.

In this work, the supplying of both types of files, MP4 and MOV are monitored and analyzed from the perspective of a streaming server, the Darwin Streaming Server, properly enhanced with our Monitor Module.

3 The Darwin Server

The Darwin Streaming Server from Apple is an open source video server that can serve 3GP, QuickTime (MOV) and MPEG4 videos for clients via Internet using standard RTP and RTSP protocols. It is based on the QuickTime Streaming Server code.

Darwin also supports live broadcasting once it uses a video codifier called mp4live, which is part of the MPEG4IP [5]. The latter performs real time video capturing generating a stream to be broadcasted via unicast to the Darwin server, which will distribute it using the RTP and RTSP protocols [16].

The Darwin is an event-driven server working as a set of processes that execute the RTSP, RTP and RTCP streaming media standard broadcasting protocols. The server

supports various compatible streaming file formats and stands up to two thousand or more on-line users [6] [18].

To perform streaming media broadcasting via RTP, it is necessary to attach a hint track at the beginning of each file. A hint track specifies, for example, RTP time scheduling and the packet's maximum size (Maximum Transmission Unit – MTU).

3.1 Server Architecture

The Darwin server is composed of a parent process that creates a child process, which is the core of the server. The parent waits for the child to exit. If in the exit of the child process an error occurs, the parent process creates a new child process.

The core server acts as an interface between the server modules and the clients in the network that use RTP and RTSP protocols to send requests and receive responses (Fig. 1). The server modules process requests and deliver packs to the client.

The server core does its work by creating four threads:

- A Main thread manages the server checking whether it needs to terminate, generating a status log or printing statistics;
- An Idle Task Thread manages a queue of tasks that occur periodically. There are two kinds of task queues: the timeout tasks and the sockets;
- An Event Thread listens to the sockets to receive requests of RTSP events or RTP packets and deliver them to the Task Threads; and
- The Task Threads receive the RTSP and RTP requests from the Event Thread. Then, the Task Thread asks the proper server module to process and send packets to the client.

The Darwin server is complex, works asynchronously and needs an event based communication mechanism. For example, when a socket is used in an RTP connection to acquire data, someone must be warned so that the data come to be processed [19].

Each Task object has two main methods: Signal and Run. The Signal method is called by the server to deliver an event to a Task object. Then, the Run is requested in order to schedule a Task to process the event.

The goal of each Task object is to implement the server's functionality by using small time slices that are not blocked. The Run method is in fact a virtual function that is requested whenever a Task object has events to be processed. Within the Run function, the Task object can call GetEvents to automatically receive all the requests that remain in the queue and previously signal events. The Run function is never re-entered if a Task object calls GetEvents within the Run function and signals before the function is completed. The Run function will only be called by a new event after it is finished. In fact, the Run function will be repeatedly invoked until all events in the Task object are cleared up from GetEvents.

3.2 Server Functioning

The Darwin Streaming Server works by performing a broadcast of its content to all clients that requested the file or by serving on demand. When the client requests the file, the number of packs emitted to each client will depend on the time concerning the beginning of the transmission (broadcast) and on the moment in which the client

Fig. 1. Darwin Server Architecture

established the connection with the server [17]. Each client’s request is served with a complete broadcast of the file [8].

The server has a modular architecture so as to facilitate the building of new functionalities through independent modules. The server uses a main process called Core Server, which truly is an interface between the clients’ requisitions and the modules. Modules must be constituted by a specific structure, with two mandatory methods, one requested by the server to initialize it and the other to execute a certain task. As to know which modules must be requested for a certain event, each module must explicit its role. Thus, each module must have a list of actions called “Roles”.

When the server initiates, it first loads modules that are not compiled in the server (dynamic modules) and next, the compiled modules (static modules). Then, the server invokes each module from the QTSS (Core) with the Registry functionality, a role every module must support. Next, the module requests the QTSS_AddRole to specify the other roles it can have.

After this, the server invokes the role ‘Initialize’ for each module registered in that role. The ‘Initialize’ executes any initialization task the module requires, allocating memory and global data structures.

When the server is deactivated, it invokes a ‘Shutdown’ role for each module registered in that role. When it executes the Shutdown, the module must perform a clearing up and release global structures.

4 The Monitor Module

In this work, we developed a monitoring module which is responsible for obtaining bandwidth use and packet loss rate for each user session. The Monitor Module directly

Fig. 2. Monitor Module Block Diagram

interacts with the Darwin Server kernel (Fig. 2.). It collects data from the server core and these data are processed on-line within the attributes of a server object.

Those attributes work directly with the server's structure, constantly computing the different information needed to assess its performance [17] [21]. This information remains hidden within the Darwin server itself and the monitor module was implemented to collect the information and create an interface to visualize the data.

The Monitor Module implements two basic routines: the Main Routine and the Dispatch Routine. In the Main Routine, all QTSS modules must supply a Main routine. The server requests the Main routine to be initialized and uses the role 'Initialize' for accessing the QTSS stub library (which loads the libraries), thus the server can later invoke the module.

In the Dispatch Routine, every QTSS module must provide a Dispatch routine. The server requests a dispatch routine when it invokes a module for a specific task, passing to the dispatch routine the task name and a specific parameter block for the task.

The Monitor module implements three rules:

- QTSS_Register_Role;
- QTSS_Initialize_Role;
- QTSS_RTSPFilter_Role.

When the QTSS_Register_Role rule is invoked by the server, the module records the rules that it wishes to notify, which are QTSS_Initialize_Role and QTSS_RTSPFilter_Role.

When the QTSS_Initialize_Role is invoked, the module performs the global object's initializations, as, for example, to obtain a reference to the QTSS_ServerObject (object that represents the server). And when the server invokes the QTSS_RTSPFilter_Role rule, the module processes the request.

When the Monitor module is notified through the QTSS_RTSPFilter_Role rule, it obtains the values of the server's total transfer rate in the attribute qtssRTSPvrCurBandwidth, of the QTSS_ServerObject object. Next, for each client's session, it gets

the transfer rate in the qtssCliSesCurrentBitRate attribute as well as the client's address in the attribute qtssCliRTSPSess RemoteAddrStr and the amount of lost packets in the object's qtssCliSesPacketLossPercent attribute which represents the session of each client. After this, it performs the calculation of bandwidth use and stores the results in a file.

We also developed a program in Java to read the results file and graphically present them. There are two visualization modes: one shows the bandwidth and the other shows the packet losses.

5 Experiments and Results

With the Monitor Module developed in this work, we conducted experiments with the Darwin server in a 100 MBps LAN network using more than 20 computers. All machines accessed a single Darwin video server. In this experimental stage, the implemented Monitor Module was successfully tested, once it collected all information within the core server generating the connections graphs. The experiments were conducted in two steps.

In the first step, MP4 video files were used, with 50 MB size. It was noticed that the Darwin server tries to always keep the files within the same data rate, but by doing so, the server ended up with a huge unused band that could be scheduled to speed up the broadcasting [20]. In the video played by clients, excellent performance and optimal image quality was observed when using the QuickTime Player [7] and the RTP protocol to access the server.

Fig. 3 shows the monitoring graph of the Darwin server with MP4 video files. It is important to notice that the server is only using 10% of the available bandwidth, in

Fig. 3. Monitoring of MPEG-4 broadcasting

such a way that its total data output rate is in the order of 10 MBps. The Darwin server, then, leaves 90% of its band totally free, unused.

In the second experimental step, we used MOV video files with up to 60 MB. It was noticed that the Darwin server treats the MOV files differently, since not all files presented the same data output rate. According to the quality of each video file, the server provides more or less bandwidth for the session.

Again, the player used was the QuickTime, with the RTP protocol to access the server. QuickTime itself adds hint tracks to the MOV files as to work with the streaming server, but that operation is not automatic, being necessary to export the MOV file in the hinted track format.

Fig. 4 shows the monitoring graph of the Darwin server with MOV video files. Unlike the former step with MP4 files, the Darwin server uses this time 90% of the available bandwidth, attaining a total 90 MBps as data output rate, as can be seen in the graph. At some moments, some videos use almost the whole bandwidth available in the server. The image quality is equivalent to the one previously obtained using MP4 files.

The Darwin server has a management feature which provides a summary of the active sessions. Analyzing the active sessions we detected that the MOV files present an output rate five times higher than that of the MP4 video files.

Through the experiments, we can conclude that serving MP4 files via streaming mode takes more processing time of the Darwin server. For this reason, the server can not deliver files at the same output rate of the MOV files. We believe that this fact does not happens due to same ‘hidden’ fault in the Darwin server, but because of the characteristics inherent to each file format, being the MOV type files more naturally biased to be broadcasted via streaming due to its own nature and historical origins.

Hence, it is clearly noticed the importance of having a monitoring module such as the one developed here and implemented within the Darwin server architecture.

Fig. 4. Monitoring of MOV broadcasting

Without a proper user session following, such a broadcasting discrepancy between the two video file formats would be easily overlooked by the system administrator.

6 Conclusions

This paper describes a monitoring module implemented into a streaming video server, in this case the Darwin Streaming Server, from Apple, an open source server based upon the QuickTime Streaming Server code, of the same manufacturer.

For such a goal, the Internet multimedia broadcasting peculiarities were analyzed, as well as the RTP and RTCP protocols. Two popular video format files, MP4 and MOV, were selected for the experiments and also reported.

The Darwin server architecture was analyzed in detail as well as the architecture of our own monitor, which was implemented as a Darwin module.

The experimental results, properly obtained by means of our Monitor Module, show a significant difference in the delivery rate of the MP4 and MOV files, being the latter delivered at a rate five times higher. Therefore, the importance of having a monitoring module such as the one here presented, since it can become an important tool for the system's manager that needs to follow its performance.

The Monitor Module here presented is still an ongoing project. We intend in the near future to turn it able to capture information other than the used bandwidth and the packet loss rate of each user's session, thus enhancing its applicability. We will also analyze more carefully the different behavior found while serving MOV and MP4 media files.

References

1. Kurose, J., Ross, K.: *Redes de Computadores e a Internet*. Addison-Wesley, Reading (2003)
2. Tanenbaum, A.: *Sistemas Operacionais Modernos*, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
3. Schulzrinne, H.: *A Transport Protocol for Real-Time Applications (RTP)* (1996), <http://www.ietf.org/rfc/rfc1889.txt>
4. Schulzrinne, H., Rao, A., Lanphier, R.: *Real Time Streaming Protocol (RTSP)* (1998), <http://www.ietf.org/rfc/rfc2326.txt>
5. Mpeg4ip Commuty Site, <http://mpeg4ip.sourceforge.net>
6. Darwin Project Site, <http://developer.apple.com/darwin/projects/streaming>
7. QuickTime Apple Web Site, <http://www.apple.com/quicktime>
8. Ferguson, P., Huston, G.: *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. John Wiley, Chichester (1998)
9. Comer, D.E.: *Internetworking with TCP/IP: Principles, Protocols and Architecture*, 4th edn., vol. 1. Prentice-Hall, Englewood Cliffs (2000)
10. Busse, I., Deffner, B., Schulzrinne, H.: *Dynamic QoS Control of Multimedia Applications based on RTP*. *Computer Communications* 19, 49–58 (1996)
11. Stallings, W.: *High-Speed Networks and Internets: Performance and Quality of Service*, 2nd edn. Prentice-Hall, Englewood Cliffs (2002)
12. MPEG Site, Moving Picture Experts Group, <http://www.mpeg.org>

13. Gnustream: A P2P Media Streaming System Prototype (2003)
14. The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming over IP (2001)
15. Wakamiya, N., Miyabayashi, M., Murata, M., Miyahara, H.: MPEG-4 Video Transfer with TCP-Friendly Rate Control. In: Al-Shaer, E.S., Pacifici, G. (eds.) MMNS 2001. LNCS, vol. 2216, pp. 29–42. Springer, Heidelberg (2001)
16. Seungjoon, S.B.: Scalable Resilient Media Streaming (2003)
17. Zhao, W., Tripathi, S.K.: Bandwidth-Efficient Continuous Media Streaming Through Optimal Multiplexing (1999)
18. Sen, S., Rexford, J., Towsley, D.: Proxy Prefix Caching for Multimedia Streams (1999)
19. Chen, S., Shen, B., Yan, Y., Basu, S., Zhang, X.: Fast Proxy Delivery of Multiple Streaming Sessions in Shared Running Buffers. *IEEE Transactions on Multimedia* 6(7) (2005)
20. Anastasiadis, S.V., Sevcik, K.C., Stumm, M.: Server-Based Smoothing of Variable Bit-Rate Streams. In: *ACM Multimedia*, pp. 147–158 (2001)
21. Tripathi, Z.: Bandwidth-Efficient Continuous Media Streaming Through Optim (1999)
22. Schulzrinne, H., Gokus: RTP: A transport protocol for real-time applications (1996)
23. Schulzrinne, H., Rao, A., Lanphier, R.: Real time streaming protocol (RTSP), request for comments 2326 (April 1998)