UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/issue/view/725

DOI: http://dx.doi.org/10.14483/udistrital.jour.tecnura.2015.4.a01

INVESTIGACIÓN

Stress tests for videostreaming services based on RTSP protocol

Pruebas de estrés para servicios de videostreaming basados en el protocolo RTSP

Gabriel Elías Chanchí Golondrino[1], Franco Arturo Urbano Ordoñez[2], Wilmar Yesid Campo Muñoz[3]

Fecha de recepción: 29 de septiembre de 2014 **Fecha de aceptación:** 24 de agosto de 2015

Como citar: Chanchí Golondrino, G. E., Urbano Ordoñez, F. A., & Campos Muñoz, W. Y. (2015). Stress tests for videostreaming services based on RTSP protocol. Revista Tecnura, 19(46), 27-36. doi:10.14483/udistrital.jour.tecnura.2015.4.a02

Abstract

Video-streaming is a technology with major implications these days in such diverse contexts as education, health and the business sector; all of this regarding the ease it provides for remote access to live or recorded media content, allowing communication regardless of geographic location. One standard protocol that enables implementation of this technology is *real time streaming protocol*, or RTSP. However, since most application servers and Internet services are supported on HTTP requests, very little research has been done on generating tools for carrying out stress tests on streaming servers. This paper presents a stress measuring tool called Hermes, developed in Python, which allows calculation of response times for establishing RTSP connections to streaming servers, as well as obtaining RAM memory consumption and CPU usage rate data from these servers. Hermes was deployed in a video-streaming environment where stress testing was carried out on the LIVE555 server, using calls in the background to VLC and OpenRTSP open source clients.

Keywords: Hermes, RTSP, stress test, video streaming.

Resumen

El videostreaming es una de las tecnologías que actualmente tiene repercusiones importantes en diferentes contextos como la educación, la salud y el sector empresarial; todo lo anterior gracias a las facilidades que esta brinda para el acceso a contenidos multimedia de manera remota, ya sea en vivo o en diferido, permitiendo la comunicación independientemente de la ubicación geográfica. Uno de los protocolos estándar que permite la implementación de esta tecnología es RTSP, sin embargo dado que la mayoría de servidores de aplicaciones y servicios en internet están soportados en peticiones HTTP, es poco el trabajo que se ha realizado en cuanto a la generación de herramientas, para realizar pruebas de estrés sobre servidores de streaming. Este artículo presenta una herramienta de medición de estrés llamada Hermes, desarrollada en el lenguaje Python, la cual permite el cálculo de los tiempos de respuesta en el establecimiento de conexiones RTSP a servidores de streaming, así como la obtención de datos de consumo de memoria RAM y porcentaje de uso de CPU de estos servidores. Hermes fue desplegada dentro de en un entorno de videostreaming, sobre el

[1] Ingeniero En Electrónica y Telecomunicaciones, Magister en Ingeniería Telemática, Candidato a Doctor en Ingeniería Telemática. Docente Institución Universitaria Colegio Mayor del Cauca. Popayán, Colombia. Contacto: gchanchi@unimayor.edu.co

[2] Ingeniero En electrónica y Telecomunicaciones, Magister en Ingeniería, Área Telemática. Docente de la Fundación Universitaria de Popayán. Popayán, Colombia. Contacto: frurbano5@gmail.com

[3] Ingeniero en Electrónica y Telecomunicaciones, Magister en Ingeniería, Área Telemática, Doctor en Ingeniería Telemática. Docente de la Universidad del Quindío. Armenia, Colombia. Contacto: wycampo@uniquindio.edu.co

cual se realizó la evaluación de estrés para el servidor LIVE555, usando para ello invocaciones en segundo plano a los clientes libres VLC y OpenRTSP.

***Palabras clave*:** Hermes, pruebas de estrés, RTSP, videostreaming.

INTRODUCTION

The technology of *streaming* involves sending video or audio data requests to the server. In response, the server sends data flows, or streams. But it is not necessary for the entire stream to arrive in order to begin to view the images or hear sound on the client side. Rather, the video can be viewed or the audio listened to while the streams that form the whole file requested continue to arrive. This way of working on the net significantly improves waiting times and also allows the handling of both live and recorded media files (Biernacki, & Tutschku, 2013).

The video streaming systems operate on the client/server model, where the client requests data from a server via a network, and the network responds to the request by delivering media content to be interpreted by the client. The audio and video are encoded in special formats that compress the data to sizes that are easy to handle. The server then delivers them via the network, the client interprets them and deploys them for presentation to the end user.

It should be emphasized that video streaming technologies have advanced so dramatically that currently the experience of the end-user on the web, given a good bandwidth, is very close to that of watching TV. Modern servers and media clients can even work intelligently according to the bandwidth constraints of the network to try to reduce buffering times and interruptions. Full screen, high quality video with hi-fi sound can therefore be enjoyed on a computer connected to the Internet.

The benefits offered by streaming include companies able to communicate more effectively with their staff and clients, small businesses able to sell more products by improving and enriching the user experience, and thousands of people attentive and listening to radio stations from anywhere in the world. It also forms a major component of many emerging video and music services, as well as being one of the most important tools in areas such as distance learning, health, telemedicine, electronic banking and network infrastructure.

On the technical side, a widely used protocol for communication between client and streaming server is Real Time Streaming Protocol, or RTSP. It was developed by the Multiparty Multimedia Session Control Working Group (MMUSIC WG) of the IETF (Internet Engineering Task Force) and published as RFC 2326 in 1998 (Liu, Du, Wang, YANG, & Wang, 2010) and RTSP version 2.0 is currently in development, as a replacement for RTSP 1.0.

RTSP (Rao, Lanphier, Stiemerling, Schulzrinne, & Westerlund, 2011) is a connectionless data flow protocol in real time used to define how information is sent between client and server, enabling the delivery of previously stored or live media content to be initiated and controlled. The protocol works at application level and ensures that data delivery is carried out correctly. RTSP defines different types of connection and sets of requirements in its quest to get an uninterrupted shipment of efficient data flow via IP networks. RTSP is independent of transport protocol and may function either on UDP or TCP. However, in most cases TCP is used for control and UDP for data transmission with RTP (Markovic, Acanski, Bako, & Mudri, 2014). During a session, a client can open and close reliable transport connections to the server via RTSP requests. Among the RTSP protocol properties are:

- Multi-server capacity: each flow of audiovisual content belonging to the same presentation can reside in different servers.
- Similar to HTTP (hypertext transfer protocol): when possible, RTSP reuses HTTP concepts.

- Extendable: new methods and parameters can readily be added to RTSP.
- Secure: RTSP uses the security mechanisms of the network at the transport level.

In the world of Information and Communications Technologies (ICTs) meanwhile, the field related to software testing is very important for identifying and removing possible defects from products. For example, there are stress tests that allow testing of the robustness and reliability of a tool, subjecting it to extreme conditions in order to obtain data about its operating limits. To test HTTP web servers, several tools exist that simulate simultaneous and sequential HTTP requests, Apache Benchmark (Olson, Christensen, Lee, & Yun, 2011) being one of the most widely used options.

However, in the area of streaming based on the RTSP protocol, very few studies have explicitly focused on stress test for streaming media servers, taking account of the particular format of the video streaming protocol with respect to the format of the HTTP requests (Xue-sen, Jun, & Hong-sheng, 2009) (Lee, Min, & Kim, 2005). Also with the increase in networks bandwidth, the performance of streaming media server has become an important factor restricting streaming media application (Yan, Haocheng, & Jinyao, 2013) . This paper presents Hermes, a testing tool developed for measuring stress. The tool enables measurement of connection setup times obtained by submitting an RTSP server to multiple simultaneous connections. It was developed in Python programming language and internally uses calls in the background to VLC and OpenRTSP open source clients. Python is used because of the ease with which it allows calls to OS (operating system) commands (run in the background of VLC and OpenRTSP) as well as for its multi-threaded support characteristics and ability to perform complex mathematical calculations.

In order to assess the operation of the Hermes stress measuring tool, a set of open source tools and technologies were prepared in a video streaming environment allowing transmission and reception of media content to different types of clients and over a number of different OS, using RTSP protocol. In this way, the tool was used to simulate the establishment of multiple simultaneous connections to the RTSP server of the streaming environment (LIVE555 server). The paper is organized as follows: section 2 includes the methodology used for the development of this work; section 3 presents concepts and technologies considered for the development of the stress measurement tool; section 4 describes each of the functional modules of the stress measurement tool; section 5 shows the test scenario in which stress tests with the proposed tool were carried out; section 6 presents the results of test implementation, reflected in streaming client connection establishment times and memory tests from streaming server; and finally section 7 shows the conclusions drawn from this work, together with consequent future research intentions.

METHODOLOGY

In order to perform stress test for video streaming services based on RTSP, we develop this work in four phases, namely: analysis of technologies and tools, configuration of the video streaming environment, design and implementation of Hermes, tests (see figure 1).

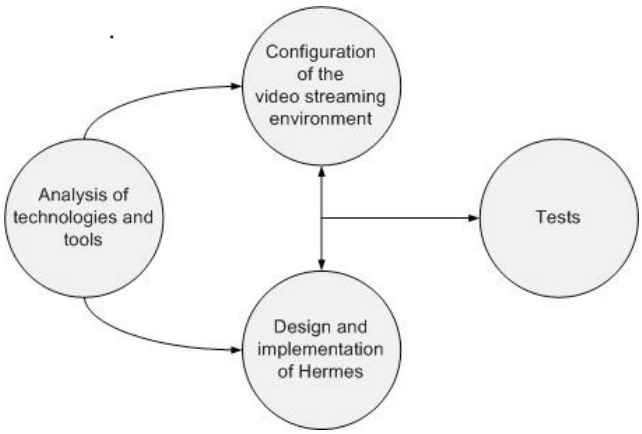


Figure 1. Phases of the methodology

Source: own elaboration.

In the first phase, we select a set of open source tools for video streaming transmission, taking into account the processes of encoding, diffusion and reception of multimedia content. In the second phase, we configure an end to end video streaming environment, considering the open source tools selected in the first phase. In the third phase, we develop a stress measurement tool called Hermes, taking into account the characteristics of the RTSP server configured in the second phase. It's important to highlight that the second and third phases are performed simultaneously, in order to allow the iterative development of Hermes. Finally, in the fourth phase we perform the connection establishment times and memory consumption tests from streaming server, using the stress measurement tool developed in the third phase. This paper addressed the phases of the methodology as follows: section 3 includes the phase of analysis of technologies and tools, section 4 considers the phase of design and implementation of Hermes, section 5 covers the phase of configuration of the video streaming environment, and section 6 describe the phase of tests.

CONCEPTUAL FRAMEWORK

Streaming technology enables content playback while downloading, rather than obliging the user to wait for the entire video to download before playing. These reproduction techniques, almost in real time, fit perfectly with the needs of users who have smart devices and access to all kinds of networks. Stream transmission can be point-to-point (Unicast) or multiple (Multicast). In addition, the material may be pre-encoded, live or real-time coded video. Communication channels may be static or dynamic, packet-switched or circuit-switched. They can support variable bitrate and some form of quality of service QoS or simply work in best effort mode (Barbero, & Gallardo, 2013).The specific properties of the video application are strongly determined by the codecs and the container format.

A codec is a compression and encoding algorithm used to reduce stream size. Video compression is achieved by exploiting the similarities or redundancies existing in a typical video signal. Examples of these are MPEG-1, MPEG-2, MPEG-4, Vorbis, and DivX. A container format meanwhile contains one or several already-encoded streams. AVI, MOV, ASF are container formats (Doumanoglou, Alexiadis, Zarpalas & Daras, 2014). In order to shape the videostreaming environment in which the Hermes tool operates, VLC and OpenRTSP were selected as clients and LIVE555 as server, taking into account that these tools have a degree of maturity and development in work with media content, specifically in regard to encoding, transmission and reception on multiple platforms, in addition to the features described below:

- VLC: an open source, cross-platform, multimedia player (Linux, Windows, Mac OS X, BeOS, Solaris, etc). It can read MPEG-1, MPEG-2 and MPEG-4/DivX files from a hard drive, CD-ROM, DVD, VCD and from a satellite receiver card (DVB-S). It supports unicast or multicast transmission and functions under IPV4 or IPV6 (Vun & Ansary, 2010) . VLC allows the reproduction and distribution of content on demand using the RTP/RTCP, RTSP protocols and 3GP files. To do this, an archive to be used as reference in the implementation of the program is defined. If the client makes a *rtsp://ip:port/resource* request, it will access the specified file. Although VLC is a great tool for transcoding or multicast transmission it has proved neither robust nor comfortable for streaming video on demand. Despite not being a greatly useful option as the main server, it can be used as a transcoder or even as an alternate server (Delgado, Quintana, Rufo, Rabadan, Quintana, & Perez-Jimenez, 2010). In the present work it is employed as a client.

  To decode a stream, VLC first unpacks it. This means that it reads the container format and separates audio, video, and subtitles. Then, each of these streams is sent to its decoder, which carries

out a mathematical processing to decompress the stream. VLC also has the ability to work in silent mode, i.e. it can be run from command line from any of the supported operating systems and can receive the streaming without displaying the graphical user interface (GUI), simultaneously managing to filter both audio and video if so desired.

- OpenRTSP: a command line program that can be used to open, transmit, receive and record streaming specified by a RTSP URL (rtsp://). OpenRTSP retrieves the session description (SDP), which allows control over the audio or video sessions of media content. The data received from these sessions are written in different output files extracted from the payload of the RTP protocol (Vun & Ansary, 2010) (Chu, Jiang, Hao & Wei-Jiang, 2013).
  Both VLC in its silent mode and OpenRTSP were used as videostreaming service clients. So that they can be launched, these tools are called in the background from the Python application proposed in this paper (Hermes).
- LIVE555: an open source RTSP server using RTSP, RTP and SDP protocols for media streaming. It is compatible with media players such as VLC and QuickTime. LIVE555 is an open source application. Its source code is available and it can be modified to meet specific requirements. LIVE555 can generate various types of media file streaming such as: TransportStream in MPEG (.ts), WebM or Matroska (.webm or .mkv), MPEG-1, MPEG-2 (.mpeg), MPEG-4 (.m4e), H.264 (.264), DV (.dv), MP3 (.mp3), WAV (.wav), AMR (.amr), and AAC (.aac). These streams can be received and/or reproduced by any RTSP/RTP media client that conforms to the standards, among these being: VLC Media Player, QuickTime Player, Amino Set-Top Boxes, and OpenRTSP (Vun & Ansary, 2010).

HERMES STRESS MEASURING TOOL

In this section the modular diagram of the Python application, Hermes is presented. It was developed to measure stress in establishing connection to the LIVE555 streaming server from the streaming environment (figure 2). In the diagram, the following main functional blocks can be seen: GUI Thread Launcher, Command Executor and Timer.

The *Graphic user interface* module uses *Tkinter graphics library*, in which a set of GUI components

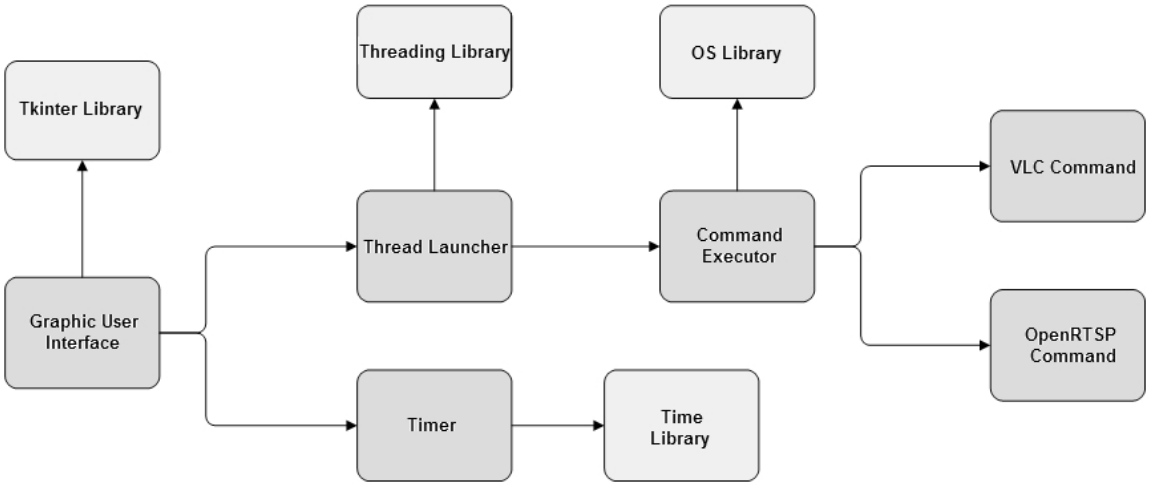


Figure 2. Hermes stress measurement tool modular diagram

Source: own elaboration.

(buttons and text fields) are created. These are used to request the RTSP streaming server address to be evaluated and the number of simultaneous connections to be established. This interface also has a pair of text fields in which the total time to establish the RTSP-type simultaneous connection with the streaming server and the average time of each connection are presented.

The *Thread launcher* module meanwhile is responsible for creating an instance of a thread for every simultaneous connection requested from the graphic interface and makes use of the Python *Threading Library*. Each thread launched is designed to establish an RTSP connection with the LIVE555 streaming server, so that for *n* threads launched, *n* parallel connections to the server are generated.

The Command Executer module is called whenever a thread is launched. Its function is to execute a command from the OS (Linux or Windows), to establish the RTSP connection with the streaming server. The command executed may be either of two types: *VLC command* or *OpenRTSP command*. VLC command launches a VLC client on the operating system console in silent mode (without opening the GUI) to establish a RTSP connection to the streaming server, while OpenRTSP command launches a OpenRTSP client on the operating system console, to establish the RTSP connection to the streaming server. For this module to execute OS commands, the *OS library* in Python is used, which allows direct interaction with the system console from the programming language.

A time count is launched simultaneously at the moment the threads are launched and counts in milliseconds until connection to the streaming server is established. This is performed by the *Timer* module with the help of the Python *Time Library*. When the connection establishment time for n simultaneous clients has been obtained in GUI, the connection processes can be stopped by calling, with the help of the OS library, the OS "killall" command.

TOOL USE ENVIRONMENT

This section presents a diagram showing the different components of the streaming environment in which Hermes was used for measuring stress (see figure 3a and figure 3b). The diagram highlights two important modules–the RTSP streaming server and the clients in which the Python tool is run. Communication between client modules and server is done through a wireless network, with the purpose of controlling the client access to streaming server through an independent network.

Server module

This module comprises the LIVE555 streaming server and media content packaged in the MPG container. The streaming server listens for RTSP requests through the 8554 port and supports the following media containers: .264, .aac, .ac3, .amr, .dv, .m4e, .mkv, .mp3, .mpg, .ts, vob, .wav, and .webm. The media content used was coded in standard definition (SD) using the MPEG-1 codec (mpga for audio and mpgv for video) using the ffmpeg open source encoding tool. This module was implemented on an AMD Quad Core computer with 4GB RAM, CPU clocked at 2.1 Ghz and Ubuntu 11.10 Linux OS.

Client module

The client module consists of two streaming clients deployed on two laptops with Ubuntu Linux 11.10 operating system. Each computer runs the Python application, Hermes, in establishing the connection, with the difference that in each case the Python application uses a different tool (VLC and OpenRTSP) in the background to connect to the LIVE555 server. For measurement purposes, the streaming clients are executed independently, i.e. measurements as a first step were taken with the VLC tool running in the background and then with the OpenRTSP tool running in the background (see figure 4).

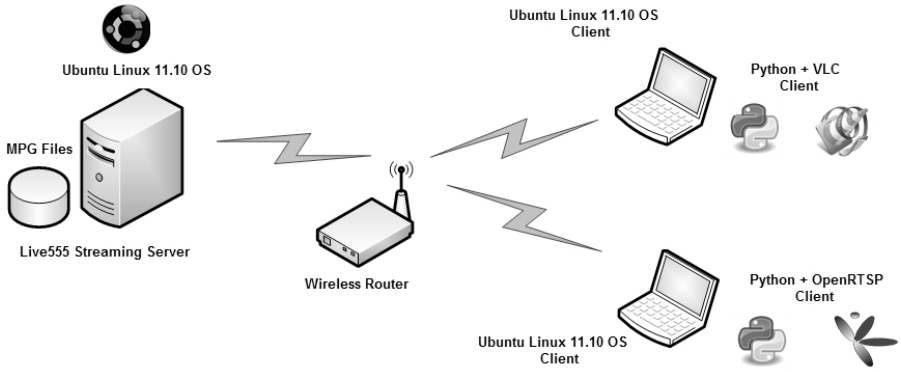


Figure 3a. Environment in which Hermes was used – detailed view

Source: own elaboration.

Figure 3b. Environment in which Hermes was used – view of actual scenario

Source: own elaboration.

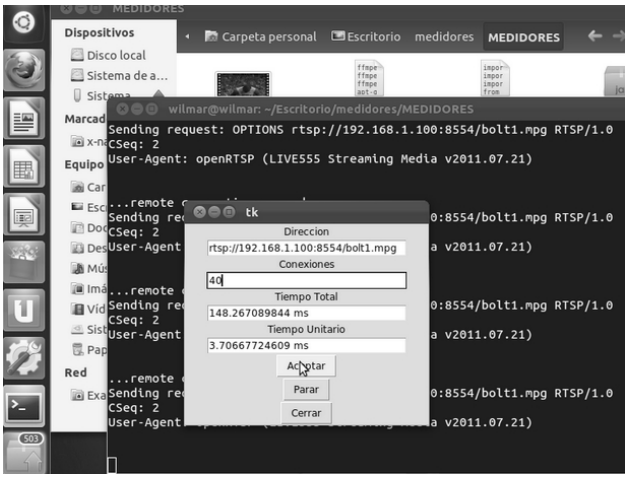


Figure 4. Client module

Source: own elaboration.

TESTS AND RESULTS

Figure 5 shows the results of testing for connection establishment time performed on the LIVE555 server and obtained by applying a number of concurrent connections using the Python stress measuring tool, Hermes. As mentioned in the previous section, this tool can establish two types of RTSP connections via the VLC or the OpenRTSP tool, in both cases running in silent mode.

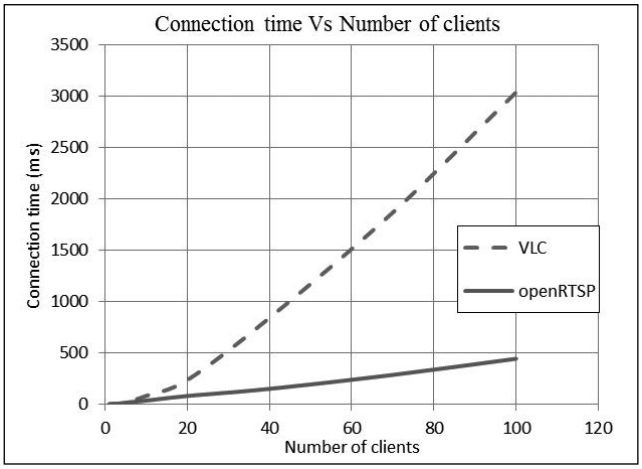


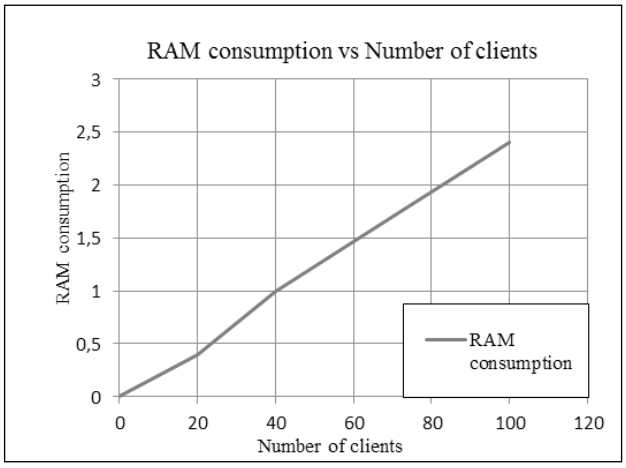
Figure 5. Connection time vs number of clients

Source: own elaboration.

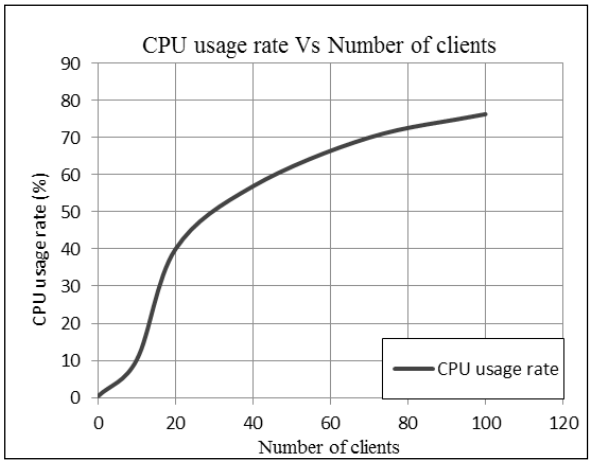
Meanwhile, in figure 6, the results of the RAM consumption testing and CPU usage rate, performed on the LIVE555 server, are shown. These tests were made by submitting the streaming server to multiple simultaneous connections using the Hermes tool that allows different number of instances of the OpenRTSP and VLC open source clients to run in the background. The memory consumption and CPU usage rate data were obtained using Linux "ps aux", which provides a report on the amount of RAM and CPU used by each of the OS active processes. This information was filtered by the awk programming language (included in the Linux OS), giving specific usage data for the LIVE555 process.

According to figure 5, when RTSP connections are established using Hermes with VLC running internally in the background, the relationship between the number of clients and the connection establishment time is directly proportional, reaching a value of 1,500 milliseconds from 0 to 60 clients, and doubling this time (3,000 milliseconds) from 60 to 100 clients.

When RTSP connections are established using Hermes with OpenRTSP running internally in the background, the relationship between the number of clients and the connection establishment time



a. RAM memory consumption (MB)



b. CPU usage rate

Figure 6. Memory usage tests

Source: own elaboration.

grows more slowly than it does with VLC, reaching a value of 250 milliseconds between 0 and 60 clients, and a value of 500 milliseconds between 60 and 100 clients.

Turning to figure 6.a, the relation between the number of simultaneous clients and the memory consumption is directly proportional, changing by approximately 0.5% each time the number of clients connected to the streaming server increases by 20. Figure 6.b indicates the relation between the number of simultaneous clients and percent CPU usage. This, as before, is directly proportional with greater variation in growth to 40 clients (a 60% increase) and a more stable variation (20% increase) between 40 and 100 clients.

CONCLUSIONS AND FUTURE WORK

From the tests on establishing connections it can be concluded that the times obtained for simultaneous requests below 100 clients are manageable and allow content to be received more than adequately. The difference in behavior between VLC and OpenRTSP lies in the fact that the latter is a tool run only from the console (silent mode without media playback) and as such responds much better to RTSP connections.

According to the tests of RAM memory consumption and CPU usage rate and according to the trend marked out by figures 5a and 5b, the CPU usage rate caused by requests containing over 100 clients on the LIVE555 server may lead to processing problems in broadcasting content, while causing difficulties in obtaining suitable reception of such by the clients.

The Hermes stress measuring tool facilitates obtaining connection establishment times for multiple simultaneous RTSP clients on a streaming server with support for this protocol. This tool as such allows secondary measurements of memory consumption to be made on the streaming server side, facilitating evaluation of the performance of the server given multiple simultaneous RTSP connections.

The use of Python language in constructing Hermes facilitates the calling (via OS commands) to RTSP clients such as VLC and OpenRTSP, through which it is possible to connect to the streaming server. Similarly the support of multi-thread features by the language allows the launching of multiple RTSP requests to evaluate the performance of the server.

The environment in which Hermes is used collects and integrates the most useful tools from the world of open source software, to implement services based on video streaming using the RTSP protocol.

In future work, it is the intention to extend the operation of Hermes so that graphics can be generated in real-time, with response times provided by the server.

ACKNOWLEDGEMENTS

This work has been carried out in the framework of the project *"Testbed for support of the service of video streaming of educational material in the FUP"* conducted in the Fundación Universitaria de Popayan (FUP). The work has benefited from the support of the *Colciencias National Doctorate Program*, Act no. 528 of 2011. We are especially grateful to Colin McLachlan for suggestions relating to the English text.

REFERENCES

Barbero, J.M., Gallardo, C. QoS for JPEG2000 Storage System: Based on Data Structure. Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on , vol., no., pp. 954 – 959, 25-28 March 2013.

Biernacki, A., Tutschku, K. (2013). Performance of HTTP video streaming under different network conditions. Multimedia Tools and Applications, V 72, N2, Springer US, pp. 1143 – 1166, 2013.

Chu, D., Jiang C., Hao, Z., Jiang, W. (2013) The Design and Implementation of Video Surveillance System Based on H.264, SIP, RTP/RTCP and RTSP.

Computational Intelligence and Design (ISCID), 2013 Sixth International Symposium on , vol.2, no., pp. 39 – 43, 28-29 Oct. 2013.

Delgado, F., Quintana, I., Rufo, J., Rabadan, J. A., Quintana, C., & Perez-Jimenez, R. Design and Implementation of an Ethernet-VLC Interface for Broadcast Transmissions. *IEEE Communications Letters, XIV*(12), pp. 1089 –1091, Dec 2010.

Begic, Z., Bajric, H., & Kos, M. (2010). Rapid synchronization of RTP multicast sessions using the retransmission server. *2010 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, (pp. 326-330).

Cherepanova, A., & Mukhina, I. (2010). Methods for quality estimation of video codecs and effective application of them in videoconferencing on outsourcing basis. *2010 International Conference and Seminar on Micro/Nanotechnologies and Electron Devices (EDM)*, (pp. 265-269).

Delgado, F., Quintana, I., Rufo, J., Rabadan, J. A., Quintana, C., & Perez-Jimenez, R. (2010, Diciembre). Design and Implementation of an Ethernet-VLC Interface for Broadcast Transmissions. *IEEE Communications Letters, XIV*(12), 1089-1091.

Lee, Y.-J., Min, O.-G., & Kim, H.-Y. (2005). Performance evaluation technique of the RTSP based streaming server. *Computer and Information Science*, (pp. 414-417).

Liu, Y., Du, B., Wang, S., YANG, H., & Wang, X. (2010). Design and Implementation of Performance Testing Utility for RTSP Streaming Media Server. *2010 First International Conference on Pervasive Computing Signal Processing and Applications (PCSPA)*, (pp. 193-196).

Olson, M., Christensen, K., Lee, S., & Yun, J. (2011). Hybrid web server: Traffic analysis and prototype. *2011 IEEE 36th Conference on Local Computer Networks (LCN)*, (pp. 131-134).

Rao, A., Lanphier, R., Stiemerling, M., Schulzrinne, H., & Westerlund, M. (2011, Septiembre 10). *Real Time Streaming Protocol 2.0 (RTSP)*. Retrieved from http://tools.ietf.org/html/draft-ietf-mmusic-rfc2326bis-27

Vun, N., & Ansary, M. (2010). Implementation of an embedded H.264 live video streaming system. *2010 IEEE 14th International Symposium on Consumer Electronics (ISCE)*, (pp. 1-4).

Wu, D., Hou, Y., Zhu, W., Zhang, Y.-Q., & Peha, J. (2001, Marzo). Streaming video over the Internet: approaches and directions. *IEEE Transactions on Circuits and Systems for Video Technology, XI*(3), 282-300.

Xue-sen, L., Jun, L., & Hong-sheng, X. (2009). Performance Evaluation Model of Streaming Media Server. *Computer Engineering*, 270-272.

Yan, H., Haocheng, H., & Jinyao, Y. (2013). Performance Measurement and Bottleneck Analysis for Streaming Media Servers . *3rd International Conference on Multimedia Technology (ICMT 2013)*, (pp. 1211–1219).

Yu, H., Chang, E.-C., Tang Ooi, W., Chan, M.-C., & Cheng, W. (2009). Integrated Optimization of Video Server Resource and Streaming Quality Over Best-Effort Network. *IEEE Transactions on Circuits and Systems for Video Technology. March 2009, XIX*(3), 374-385.

Zhang, H., Jiang, G., Yoshigira, K., Chen, H., & Saxena, A. (2009). Resilient Workload Manager: Taming Bursty Workload of Scaling Internet Applications. *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, (pp. 19-28). New York.