PERFORMANCE EVALUATION OF MULTIMEDIA SERVICES OVER IP NETWORKS

Odd Inge Hillestad *Bjornar Libak* *Andrew Perkis*

Centre for Quantifiable Quality of Service in Communication Systems
Norwegian University of Science and Technology
Trondheim, Norway
e-mail: {hillesta, libak, andrew}@Q2S.ntnu.no

ABSTRACT

We present a streaming media test bed for IP networks. Besides a streaming server and a streaming media client, it consists of an IP network emulator, a high-performance packet capture device and a packet flow regenerator enabling repeatable performance measurements of streaming media applications and network QoS mechanisms in a controlled environment. Some initial results from the work involved in verifying the performance of the packet regenerator are also presented.

1. INTRODUCTION

Streaming services over IP networks are gaining momentum, and consumers show an increased interest in being able to play and enjoy their media wherever they are. Deployment of high speed Internet access networks and continuous development of more efficient compression schemes for audio and video are two of many important factors enabling higher quality IP-based multimedia services to end-users. The recent success and large-scale deployment of portable media players indicate a consumer demand for portability requiring transparent delivery of media resources to end-users irrespective of network access type, current network conditions or limited capabilities of the end-user's communication device. Research in these topics form the basis for the concept of UMA (Universal Multimedia Access) [1]. Ideally, all processing in both end-systems and network nodes for a given multimedia application should strive to maximize the end-user's perceived quality of this application. Measuring end-users perception of audio-visual quality or to which extent they react objectionable to distortions introduced by compression and packet-switched transmission is extremely difficult and depends on factors that are not easily modeled (e.g. human diversity, preferences and application knowledge). VQEG (Video Quality Experts Group) [2] is currently working towards a standardization of quality metrics that could be employed by multimedia streaming applications.

Figure 1 depicts a typical streaming session in which some pre-encoded media content is being delivered from a streaming server. Compressed media data is wrapped in RTP, UDP an IP headers, and transmitted over an access network. On the network layer, the media data is only recognized as a packet flow throughout the session. Because IP networks introduce packet loss, delay and delay jitter, a playout buffer is needed both to absorb the variation in delay and allow for retransmission of dropped packets. To alleviate the effect of missing media data, the decoder should be error resilient and be able to perform error concealment [3][4].

In this paper, we present a IP-based streaming media test bed for evaluating the performance of streaming media applications and different strategies that enable network Quality of Service (QoS) for these applications. The test bed is flexible in that it consists of a set of off-the-shelf and/or open-source software components that can be put together to simulate a specific streaming scenario. Besides streaming servers and streaming media client software, components include a controlled test network, a hardware network emulator, a high-precision packet capture device and a packet stream regenerator to enable accurate recreation of specific network conditions.

To be able to run repeatable tests and measurements in a controlled environment using real streaming media systems and real network devices, we need to verify the performance and precision of the testbed components. For instance, to be able to recreate specific network scenarios for interactive applications that require low bounds on delay (and buffer sizes), we must verify that the variable delay introduced by network emulation and packet stream regeneration will not considerably affect our results. The paper is organized as follows; Section 2 describes the different network and end-system components of the testbed, section 3 gives some preliminary results of the performance of our packet regenerator, while section 4 gives a short summary and discussions.

2. STREAMING MEDIA TEST BED

Conceptually, the test bed consists of an *application part* and a *network part*, where the former is media aware, and the latter can be either media aware or unaware. Before describing each component, an overview of the network part is presented.

In general, the network part of the testbed setup can consist of several *manipulation elements* $\{m_i\}_{i=0}^{n}$ which distort the original flow by introducing packet loss and delay. Examples are real routers and switches fed with competing traffic, software routers, network simulators (e.g. ns-2[5]), or network emulators. The elements will be chained together like shown in figure 2.

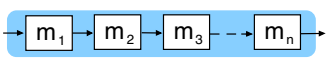


Fig. 2. Chain of n flow manipulation elements

The transfer of flows between elements can either be done online, by having a network link between the two elements, or offline, by capturing the flow and writing it to a *trace file* together

0-7803-9332-5/05/$20.00 ©2005 IEEE

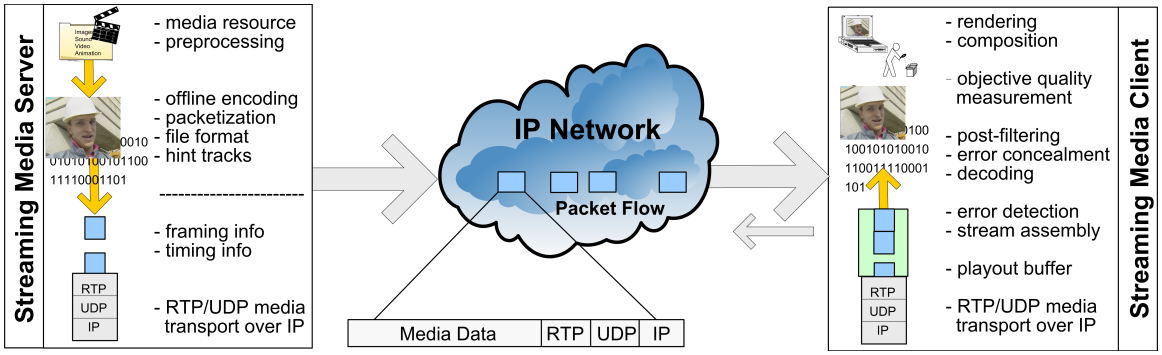


Fig. 1. Streaming system

with timing information for later manipulation. Further, we can say that each offline transfer divides the chain into two *phases*, separate in time. Typically, an offline transfer is necessary when the element on the right hand side is a Discrete Event Simulator, which because of its own time-scale needs to reside in a separate phase. After offline manipulation, it may be necessary to regenerate the packet flow using a *flow regenerator*. Regeneration is another phase separation point. It may be used to replay a captured flow into a router, or to replay an already manipulated flow for reception at the media decoding host.

Figure 3 gives an overview of the specific test bed setup referred to throughout this paper. The packet flow originates at the

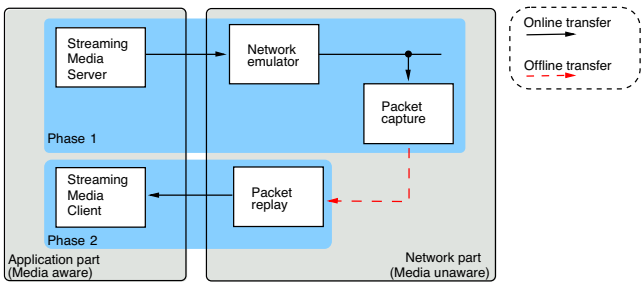


Fig. 3. Test bed overview

streaming server, and ends up at the decoding host. In the middle, there is a simple network part only consisting of a single manipulation element, namely a network emulator. After passing through the emulator (phase 1), the flow is captured by a capture device, and later replayed by a flow regenerator (phase 2)[1]. None of the components in the network part are media-aware, i.e. all operations are performed on IP level or below.

Our aim is to study the effect of packet loss and delay on packet flows containing media data in a controlled environment. It is therefore crucial that no unexpected non-measurable delay is introduced by any of the components. In section 3, we present results indicating the delay introduced by the packet regenerator.

[1] The experiments in this paper could have been conducted using a one-phase setup. However, for convenience, and also to test and verify the flow capture and replay components, two phases were used.

2.1. Streaming media server

In our testbed setup, the Envivio 4Sight MPEG-4 Streaming Server (4-Sight) [6] is used. The server transmits pre-encoded MPEG-4 video over RTP/UDP [7] by parsing the hint track available in an MP4 file [8]. The framing and timing information available in this hint track decides how video data is mapped into RTP packets and sent across the test network. As we are using the playlist broadcaster functionality available in both 4-Sight and open-source alternative Darwin Streaming Server [9], no RTSP negotiation is performed prior to transmission.

2.2. IP Network Emulator

Network emulation is a way of synthetically subjecting applications and hosts to real-world network impairments. While network emulators can not match the controllable and repeatable characteristics of discrete event simulators, they offer real-time operation, are easily modifiable and offer better repeatability than measurements in live networks [10]. Network impairment patterns such as packet loss, delay, jitter and bandwidth constraints are just some of the parameters configurable in a network emulator. In our test bed, we use the PacketSphere Network Emulator from Empirix [11], which is a commercial solution, while Nist Net [12] is an open-source network emulator for Linux.

2.3. Capturing packet flows

To capture packet flows, an Endace DAG3.5E card [13] was used. This is a *network monitoring interface card* capable of capturing incoming packets at high rates without loosing packets. The packets are written to a trace file which format includes the packet headers (link layer and IP), IP payload and a microsecond precision timestamp for each packet, reflecting the arrival time relative to the first packet in the flow.

The freely available open source software package *dagtools* [13] provides conversion tools from the trace format to the well known *Pcap* (tcpdump) format. Neither data nor time resolution are lost during conversion.

2.4. Packet flow regeneration

A flow regenerator consecutively reads packets and their corresponding timestamps from a trace file and sends them out on a network interface. In this process, there are several possible sources

of unwanted delay, depending on the operating system, hardware resources, system load and the implementation of the regenerator. Examples are disc access, timer resolution and the process scheduler in the operating system. To minimize these effects, one could use a real-time operating system with higher timer resolution and the ability to prioritize real-time processes. On the other hand, the delays introduced may not be significant for the test results. For instance, if the delays are much smaller than the decoder's playout buffer, they can most likely be ignored.

With this in mind, one of the purposes of this work was to study the performance of a flow regeneration tool called *tcpreplay* [14] running on a regular Linux operating system. Tcpreplay takes a pcap file as input. Because of the real-time extensions of the 2.6 kernel, both 2.4 and 2.6 kernels were evaluated at different loads. The 2.4 kernel is a pre-built debian sarge kernel image of version 2.4.26. The 2.6 kernel used was version 2.6.8 with the "Preemptible Kernel" option (CONFIG_PREEMPT=y) set. This option allows low-priority processes running in kernel mode to be interrupted by time critical events [15]. The following hardware configuration was used: 1 GB RAM, 3.0 GHz Pentium 4 CPU and SATA 7200rpm 8MB cache disks.

2.5. Streaming media client

As shown in figure 1, the streaming media client receives, reorganizes and buffer media data in a playout buffer. The decoder fetches media data from the playout buffer, performs error detection, decoding and error concealment, so that the video frames or audios sample are available for rendering by a display/sound device at presentation time. In our work we use the VLC player from Videolan [16]. It uses the 3rd. party libraries *livedotcom* [17] and *libavcodec* [18] for RTP streaming and MPEG-4 decoding, respectively.

3. PRELIMINARY RESULTS

This section will present preliminary results from the process of verifying the performance of the individual testbed components.

3.1. Test sequences

For our measurements we used the "Standardized Evaluation Material" (StEM) under license from Digital Cinema Initiatives (DCI) [19]. StEM is a mini-movie available in 4K resolution (4096 by 1714 pixels), 24 frames per second and 16605 frames in total. Only considering the video component of StEM, a lower resolution version was made by cropping the 25 upper and 25 lower lines of the original frames, followed by downscaling using bilinear downsampling to obtain a version with resolution 1024x416 pixels. This clip was encoded using Envivio 4Coder 3.0 [6] to MPEG-4 Advanced Simple Profile at various constant bit rates (CBR), with a intra period of 1 second (GOP size = 24) and encoder video buffer size of 1 second.

3.2. Performance of the packet regenerator

To measure how accurately tcpreplay is able to regenerate packet flows, we compared a 5 Mbps MPEG-4 network trace from 4-Sight with several network traces from tcpreplay regenerating the 4-Sight trace. Similar measurements where performed with tcpreplay running on both the Linux 2.4 and 2.6 kernel. Figure 4 (a) shows the distribution of differences in packet inter-arrival times between the traces from 4-Sight and tcpreplay running on the 2.4 kernel. Figure 4 (b) shows the corresponding distribution with tcpreplay running on the 2.6 kernel. As these plots clearly indicate, there is a considerable performance gain when using the Linux 2.6 kernel. In fact, considering the extreme values obtained from our measurements, tcpreplay running on a Linux 2.6 kernel is able to recreate the packet flow ten times more accurately than tcpreplay running on the 2.4 kernel. While tcpreplay seems unable to recreate the trace with a higher precision than ± 20 ms on the 2.4 kernel, these bounds are around ± 2 ms on the 2.6 kernel.

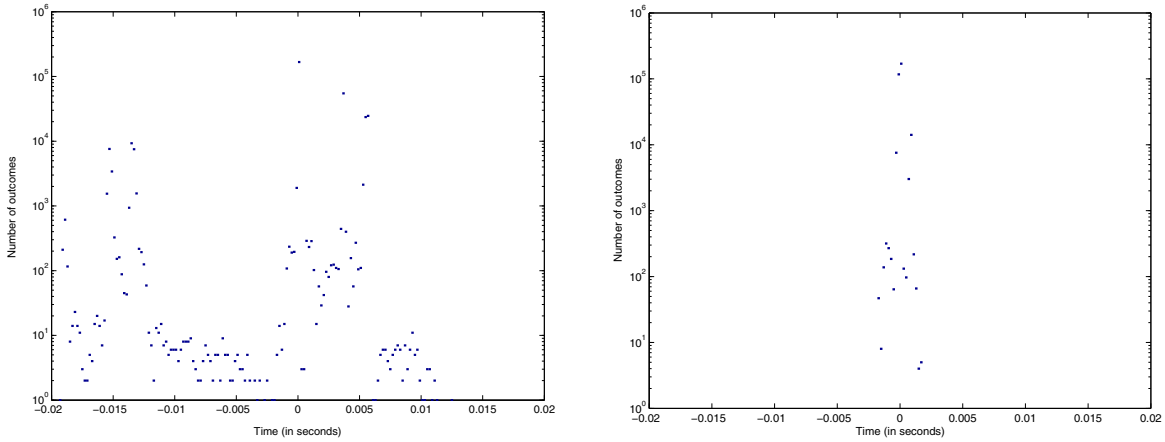
	Absolute difference in inter-arrival times (ms)		
Bit rate	Mean	99th percentile	Max value
1 Mbps	0.261	0.953	2.10
5 Mbps	0.139	1.100	2.70
10 Mbps	0.077	0.914	1.80
15 Mbps	0.078	1.100	5.90
20 Mbps	0.073	1.200	24.5

Table 1. Absolute difference in packet inter-arrival times between traces from 4-Sight and tcpreplay running on Linux 2.6 kernel, for different bit rates.

Table 1 shows statistical characteristics of the absolute differences in packet inter-arrival times between traces from 4-Sight and tcpreplay running on Linux 2.6 kernel, for five different MPEG-4 flows encoded at 1, 5, 10, 15 and 20 Mbps. We see that, e.g. for the 15 Mbps flow, 99 % of packets in the regenerated packet flow are sent within ± 1.1 ms of the intended sending time. The maximum error increases for higher sending rates, e.g. the worst-case difference between actual and intended sending time for the 20 Mbps packet flow is 24.5 ms. It should be noted however, that only 3 out of 1244686 packets were sent more than 3.3 ms later than intended during this session. This phenomenon could be related to higher priority system processes forcing tcpreplay to sleep longer than intended [14]. Also, we see that the mean error in sending time decreases as the bit rate increases. For higher bit rates and an increasing number of packets per video frame, more and more packets are sent immediately succeeding each other, without tcpreplay having to sleep until the next calculated sending time. This way, the relative frequency of idle periods is reduced, and the rate of times at which tcpreplay has to wake up at the exact right moment is reduced.

4. SUMMARY AND DISCUSSIONS

We have presented a streaming media test bed for IP networks. Besides a streaming server and a streaming media client, it consists of an IP network emulator, a high-performance packet capture device and a packet flow regenerator enabling repeatable performance measurements of streaming media applications. Analyzing the delay introduced by the packet regenerator in our measurements, we observe that this may be limited to ± 2.0 ms when tcpreplay is running on a Linux 2.6 kernel. Depending on the application at hand, this uncertainty may or may not be significant. For instance, interactive conversational applications require small playout buffers (in the millisecond region), and thus this uncertainty introduced by the packet regenerator is more significant than if the playout buffer



(c) (d)

Fig. 4. Distribution of difference in packet inter-arrival times comparing traces from 4-Sight and TCPReplay: (a) 5.0 Mbps StEM trace, Linux 2.4 (b) 5.0 Mbps StEM trace, Linux 2.6 (Bin width equal to 0.5 ms for both traces.)

is large (in the region of several seconds), like in video-on-demand or multicast streaming scenarios.

5. REFERENCES

[1] A. Perkis, Y. Abdejaoued, C. Christopoulos, T. Ebrahimi, and J. F. Chicharo, "Universal multimedia access from wired and wireless systems," *Circuits, Systems and Signal Processing; Special issue on Multimedia Communications*, vol. 20, no. 3, pp. 387–402, 2001.

[2] "Video quality experts group (vqeg)," http://www.vqeg.org.

[3] D. Wu, Y.T Hou, W. Zhu, Y-Q Zhang, and J.M Peha, "Streaming video over the internet: approaches and directions," *CSVT*, vol. 11, no. 3, pp. 282–300, March 2001.

[4] Y. Wang and Q. Zhu, "Error control and concealment for video communication: A review," *Proceedings of the IEEE*, vol. 86, no. 5, May 1998.

[5] "The Network Simulator ns-2," http://www.isi.edu/nsnam/ns/.

[6] "Envivio Inc.," http://www.envivio.com/.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," *IETF, Request for Comments: 1889*, 1996.

[8] J. Chakareski, J. Apostolopoulos, S. Wee, W t. Tan, and B. Girod, "R-d hint tracks for low-complexity r-d optimized video streaming," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, Taipei, TW, June 22–25, 2004.

[9] "Darwin," http://apple.com/darwin.

[10] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002, USENIX Association, pp. 255–270.

[11] "Empirix Packetsphere Network Emulator," http://www.empirix.com/.

[12] Mark Carson and Darrin Santay, "Nist net: a linux-based network emulation tool," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 111–126, 2003.

[13] "Dagtools," http://dag.cs.waikato.ac.nz/.

[14] "Tcpreplay," http://tcpreplay.sourceforge.net.

[15] Robert Love, "Introducing the 2.6 kernel," *Linux J.*, vol. 2003, no. 109, pp. 2, 2003.

[16] "VLC media player," http://www.videolan.org.

[17] "Live.com," http://www.live.com/liveMedia/.

[18] "FFMPEG," http://ffmpeg.sourceforge.net/.

[19] Digital Cinema Initiatives (DCI) and The American Society of Cinematographers (ASC), "StEM mini-movie access procedyre, available at http://www.dcimovies.com/," November 2004.