J Real-Time Image Proc (2011) 6:247–256
DOI 10.1007/s11554-010-0183-4

ORIGINAL RESEARCH PAPER

A real-time streaming server in the RTLinux environment using VideoLanClient

Alfredo Petrosino · Marco Miralto · Alessio Ferone

Received: 1 July 2008 / Accepted: 7 October 2010 / Published online: 27 October 2010

Abstract Accessing rich multimedia content through terminals and bandwidth-constrained networks is an important issue. Therefore, multimedia servers for delivering such data deserve much attention. A novel real-time streaming system for multimedia is presented, based on real-time operating systems. The VLC streaming server has been improved by employing a modified version of RTLinux that includes DMI and/or EDFI schedulers with the aim to build a complete streaming system for multimedia. Tests have been performed to measure one of the most important aspects of such systems, that is, jittering. The proposed system is able to achieve good performance both in simulated and real-world situations.

Keywords Real time · Audio/Video Streaming · Multimedia system · VLC · RTLinux · Real time scheduling

1 Introduction

Modern multimedia systems are getting increasingly popular. Indeed, concepts like video and audio streaming [1, 2], less popular years ago, are used daily by millions of people, thanks to the explosion of broadband internet connections.

Multimedia systems must comply with two requirements [3]:

A. Petrosino (✉) · M. Miralto · A. Ferone
Department of Applied Science, University of Naples "Parthenope", Centro Direzionale Isola C4, 80143 Naples, Italy
e-mail: alfredo.petrosino@uniparthenope.it

A. Ferone
e-mail: alessio.ferone@uniparthenope.it

- management of a high data rates
- need of real-time playback.

The first requirement should be in accordance with human sight capabilities, which can elaborate a great amount of data per second. The second point has been afforded during these past years by adding real-time capabilities to operating systems (OSs) [4]. This mainly involves that some parts of operating systems, such as the file system and the CPU scheduler, were modified so to lower the average response time of the OS to jobs submission [5]. For instance, we can imagine a simple scenario where we need to broadcast a movie in PAL format over a network requiring that the streaming server should be able to make at least 25 submissions each second.

One of the most important problems involved in the data streaming is known as jitter, which is formally defined by the CCITT as "short-term variations of the significant instants of a digital signal from their ideal positions in time" [6].

This definition implies that timing jitter possesses cumulative properties, so that without adequate control it can grow to an amplitude sufficient to merge adjacent submissions. For this reason, jitter has to be kept as low as possible. A solution usually employed to solve the jittering problem consists of using a cache on the client side to buffer a certain amount of incoming data before starting the playback. However, small systems like smart phones, PDAs and, in general, all portable devices have small amount of memory and hence the jitter problem can be addressed only partially by buffering data.

Many approaches have been proposed to address the problem of streaming multimedia contents in a real-time environment. They usually are hardwired based on specific compression algorithms [7] or specific hardware to exploit

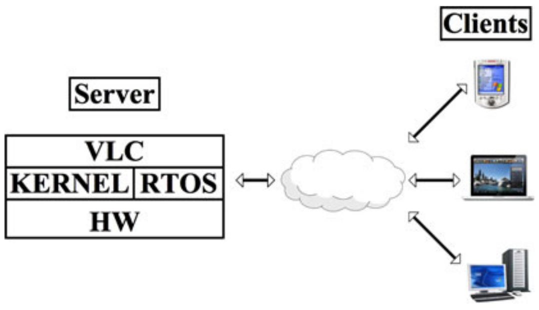

Fig. 1 The proposed system

real-time performance [8]. Both approaches are clearly not portable.

The aim of the present work is to build a portable streaming system (Fig. 1) which can exploit real-time capabilities of a real-time operating system, i.e. a system able to supply multimedia contents through a network with deterministic timing so to eliminate the source of jitter on the server side, thus minimizing jittering. The system should have the following features:

- Precision: the system has to send data using a socket within rigid time constraints.
- Scalability: the system has to adapt to many users' requests accepting new transmission streams.
- Reliability: when a new request is submitted, the system has to verify if it can be accepted; i.e. if the set of active tasks is not feasible adding the new one, the request has to be rejected.

There are many proprietary architecture solutions to achieve multimedia streaming, most of them using proprietary protocols. In this work we present a solution for an open source architecture, based on Video Lan Client (VLC) open source software [9]. Inspite of the name, VLC is not only a multiformat player that can manage playback for a variety of free and non-free multimedia formats, but it is also a multiprotocol streaming server that can handle RTP (unicast and multicast on the UDP layer) and HTTP-encapsulated streaming technology for Video-on-Demand. It can also reproduce streaming, supporting, once again, RTP, shoutcast and icecast protocols. The solution we propose is based on real-time capabilities of operating system. A variety of approaches to real-time have been recently proposed and many others have recently appeared too [10]. Montavista Linux, Windows CE and the recent RT patchset by Ingo Molnar to the vanilla Linux Kernel are some of recent approaches. All these operating systems share a basic feature: they are all soft real-time systems. What we need to gain our goal is a hard real-time environment like that one offered by QNX, RTLinux and RTAI. We need hard real-time to get both low latency over job submission and a deterministic job scheduler able to run a specific task at a given moment and, also, able to complete its operations within a given time (deadline).

We choose to lean on RTLinux [11] for developing our work, according to which every real-time task is realized as a kernel module, named by us *VLCKern module*. Tasks can be submitted by loading the modules into the kernel and run them in kernel space, thus sharing the kernel address space. There are no memory protection mechanisms implemented and tasks use non-swappable memory (swap in and out operations can be the cause of non-deterministic delays during the execution of a task).

Many applications can benefit of such system, for example, video surveillance, monitoring of protected areas, sensor networks and in general all those application fields where it is necessary to stream multimedia data through communication channels.

The work is organized as follows: Sect. 2 describes the proposed system and gives the details of the software architecture design and implementation, including the VLCKern module. Section 3 reports results in terms of time performance and jittering. Concluding remarks and further ongoing work are described in the last section.

2 Proposed system

To use a real-time operating system, a primary task is to analyse which operations should be performed in the real-time environment. The main idea is to keep out of the real-time core all those operations whose computations could be conditioned by non-deterministic events. In performing this division between real-time tasks (in kernel-space) and non real-time tasks (in user space) it is necessary to remember that floating-point operations should be avoided. Based on the VLC input/output chain depicted in Fig. 2, we can identify the following steps:

1. reading multimedia source
2. demultiplexing/decoding and packetization

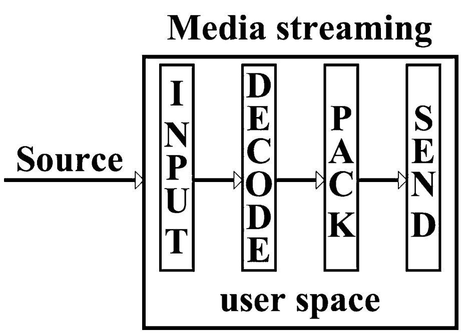

Fig. 2 The VLC input/output chain

3. coding into the final format
4. sending through a socket or writing in a file.

Reading the source media is not a big deal because of the device high transfer rate (such as ATA/SATA, USB2 or FireWire devices) and filesystem performance and hence real-time computation is not necessary. Steps 2 and 3 are the most CPU-demanding activities done in VLC but they depend on the file format to process and involve floating-point operations and hence they are not performed in real-time. Also, the decoding process relies on several audio/video codecs which should be ported in kernel space. This operation would involve intensive re-design of algorithms implemented into the codec libraries to be adapted to the real-time environment in kernel space. Last but not least, this approach would require a deep re-engineering of VLC architecture. What really needs to be handled in real-time is the buffering and sending of packets with hard real-time constraints.

We can summarize tasks in four points (Fig. 3):

1. VLC is merely a stream feed.
2. A real-time task gets data from VLC, saves them in a buffer and sends them using a socket.
3. VLC runs in user-space while the real-time task runs in kernel-space; hence, a communication method is necessary.
4. Packet submission performed by VLC has to be disabled so that the real-time task can handle it.

In the following sections, each part of the proposed system is explained, with respect to the scheme of Fig. 4, along with the interactions between them.

2.1 VLCKern module

VLCKern is a kernel module designed by us to guarantee real-time streaming. It involves the following tasks:

- Wait for a streaming request from a client in user space.
- Parse request parameters.
- Create a real-time thread to stream data according to the received parameters.

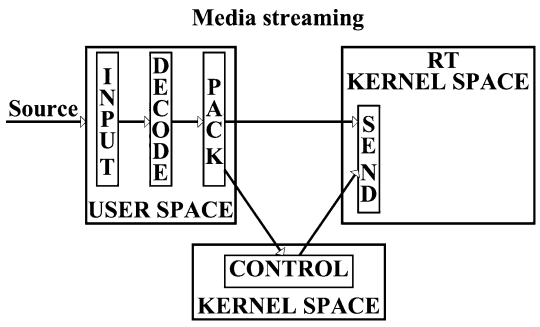

Fig. 3 The real-time VLC input/output chain

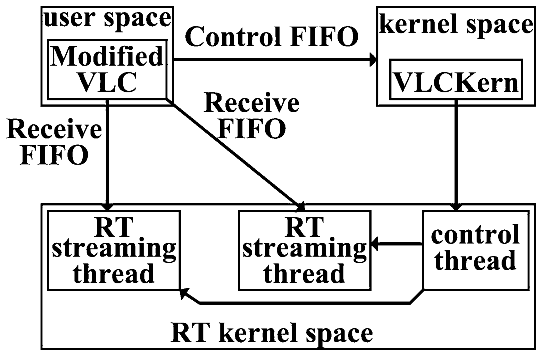

Fig. 4 Components of the proposed system

VLCKern module is developed following a hybrid multithread model where two kinds of thread are used: non real-time threads (kthread) scheduled by Linux kernel, and real-time threads whose creation and scheduling is handled by RTLinux kernel.

RTLinux works as a mini-kernel running side by side with a patched standard Linux kernel. It supports all hardware supported by the standard kernel. We used a free version of RTLinux, community supported, in combination with a 2.4.29 Linux kernel. RTLinux standard scheduler is priority driven and supports periodic tasks. It supports preemption, as higher priority tasks can preempt lower priority ones. Priorities are fixed and are declared when the job is submitted to the system. At time of task submission, we must specify the task period too. The scheduler keeps a queue of ready tasks, i.e. all the tasks whose period timer has been fired. Among them, the one featuring higher priority is selected for execution and can only be preempted by an higher priority task that eventually is moved to the ready queue. The scheduler is implemented in a kernel module and, most significantly, new schedulers can be built working on a simple source file, inserted at run-time at our convenience.

For our work we used the deadline monotonic with inheritance (DMI) [12] and earliest deadline first with inheritance (EDFI) [13] schedulers. As the name suggests, these schedulers are derived from the Deadline Monotonic and the Earliest Deadline First [14] schedulers, respectively. The difference between the two algorithms is that DMI sorts tasks by their period, while EDFI sorts tasks by their deadline. Both of them add deadline-inheritance mechanisms to the parent schedulers. They were realized by Marc Maurer [15] for RTLinux, based on [16]. The idea behind these schedulers is to provide the developer with tools to easily implement mutual exclusion of shared resource in a real-time environment. The choice of one of the above algorithms is strongly related to the application and the nature of task to be performed. In general, EDFI has proved to be more flexible though it needs more computational resources.

Every job submitted to these two schedulers must bring several information:

- period of the task
- deadline of the task
- expected CPU burst
- a string which specifies all used shared resource and the estimated time for which the task will hold each of them.

Being a kernel module, the commands *insmod* or *modprobe* can be used to initiate the execution of VLCKern as for any other kernel module. Execution of one of these commands cause a call to the module entry function (init_module()) which performs the following operations:

- Creation of both control and receiving FIFOs
- Creation of a (not real-time) kernel thread using the function kernel_thread()

Every FIFO has a preallocated memory made of 1,024 elements, each one of 1,328 bytes. The element dimension has been chosen based on the MPEG-TS format and is hardcoded because there are no methods to dynamically allocate it at run time. Each FIFO acts like a buffer for data produced by VLC in user-space and ready to be sent to clients.

2.1.1 Control thread

Control thread is a kernel thread, i.e. it is not a real-time thread because of the task it is supposed to perform. It periodically (every second) reads a specified FIFO (defined by the costant CONTROLFIFO) in a non-blocking way. This FIFO is used to submit requests, from user space, for allocating new streaming (real-time) threads.

Task's pseudo-code is reported below:

```
1: while true do
2:   if new_allocation_request then
3:     allocate_new_threadstructure()
4:     clean_new_thread_structure()
5:   end if
6:   n <- read_from_control_fifo()
7:   if n.magic > 0 then
8:     rtl_thread_struct <- n
9:     real_time_task_create(rtl_thread_struct)
10:    admission_control()
11:  end if
12:  sleep_one_second()
13: end while
```

The control thread pre-allocates a rtl_thread_struct, reads from control FIFO and saves data in a temporary structure (*n* into the code). If the structure member magic value is greater than 0, then the FIFO contains a new streaming request and the other structure values are the streaming parameters: period, deadline, estimated CPU time, FIFO id where data will be sent, multicast group IP address and port. These parameters are used to create a real-time thread to handle the streaming, which indeed will start its execution only if the EDFI or DMI admission control procedure is passed. This procedure admits a new task only if the scheduling remains feasible, i.e. only if the scheduling of the new task is compatible with the scheduling of the task already admitted. Hence, the admission control procedure is fundamental to guarantee that the system will never miss a deadline.

2.1.2 Real-time streaming thread

The first step consists in setting up all the parameters needed to perform data submission, like period, deadline and estimated CPU time (used for the timing constraints) and IP address, port and protocol (UDP) (used to effectively access the net). At this moment the thread is ready to begin the streaming task that can be summarized in six steps:

1. call to the function pthread_wait_np() (put the thread into the waiting queue)
2. allocate and clean the sending buffer
3. read multimedia data from a specified FIFO
4. create and initialize message to be sent
5. write data to the socket
6. deallocate send buffer

2.2 Real-time VLC

This section discusses the user space part of the proposed system, that is our changes to VLC. Even though other programs can be used to stream media, VLC has been chosen because of its modularity and the well-documented architecture. In particular, changes are made in the RTP/UDP plugin that is the one responsible of the network media access.

VLC standard interface for module developing requires that an initial description of the module is provided. This task is accomplished defining a certain amount of macros which describe the parameters configurable by user. In our implementation, a checkbox to activate real-time capabilities has been added, that, when checked, allows the user to choose other parameters such as the control FIFO number, the sending data FIFO number and the period in ms. These parameters are used for initiating the module, and in particular, if the previous checkbox is checked, for disabling standard sending procedure and enabling the real-time one. Specifically, using real-time extension in UDP module forces VLC to not create socket and timers, and to use instead a FIFO that will send packets to the real-time module.

3 Experimental results

In this section, we will show results obtained employing the proposed system. A first part of the test has been performed on VLCKern module in order to verify its performance separately from the rest of the system. The second part of the test has been carried out over the entire system to show its performance in a real-world situation. The hardware configuration adopted during development and testing phases is as follows:

- Desktop
  - CPU Intel Celeron D 2.5 Ghz
  - 512 MB RAM DDR
  - HDD ATA 7200 RPM
  - network interface Marvell/Yukon 10/100/1000
- Laptop
  - CPU Intel Mobile Celeron 1.8 Ghz
  - 256 MB RAM SDRAM
  - HDD ATA 5400 RPM
  - network interface Intel Pro VE 10/100

The software configuration adopted is as follows:

- Kernel Linux 2.4.29
- RTLinux version 3.2r1 patch for kernel 2.4.29
- DMI/EDFI patch for RTLinux (modified to work with RTLinux 3.2rc1)
- Debian GNU/Linux 3.1 (Sarge)

3.1 VLCKern: jittering

As stated before, using a real-time OS is meant to keep jittering as low as possible by adopting RT-FIFOs like send buffers.

For this reason, many tests have been performed to measure the jittering of the proposed architecture. A typical test would read audio/video data from a source and send it to clients using VLCKern. On the client side measures have been taken using tcpdump. Although a dedicated network has been created for testing, neither TCP and UDP are real-time protocols, hence nothing can be said about packet loss, receiving time or delay, due to many issues that can cause them in a network.

3.2 Test 1

The aim of this test is to verify the jittering values in a simple scenario. The parameters used for the testing scenario 1 are

- scheduler: EDFI
- 1 video streaming
- period: 15 ms

The first test has been performed using a single stream, so EDFI scheduler has to handle just a single task with a period of 15 ms. Figure 5 shows the mean values of the same test repeated 1,000 times. As can be seen in Fig. 5, except for some outliers, all packets arrive to clients with very low jitter values. On a total number of 1000 packet submissions, a 0.2% shows a low jitter, while a 0.1% shows a higher jitter.

3.3 Test 2

The parameters used for the testing scenario 2 are

- scheduler: EDFI
- 15 videos streaming
- period: 15 ms

The second test is performed by transmitting 15 video files at the same time, each one usable by a different IP address. Figure 6 shows the mean jitter measurement relatives to the 15 files. It can be observed that in this test, packet arrival is in the interval ± 2 ms, that leads to a higher jitter measure although very stable. Causes of this behaviour can be found by considering:

- the number of tasks that share the critical section used to obtain an exclusive access to the network media
- the network delays, i.e. both the clients employed in this test had requested all the 15 videos

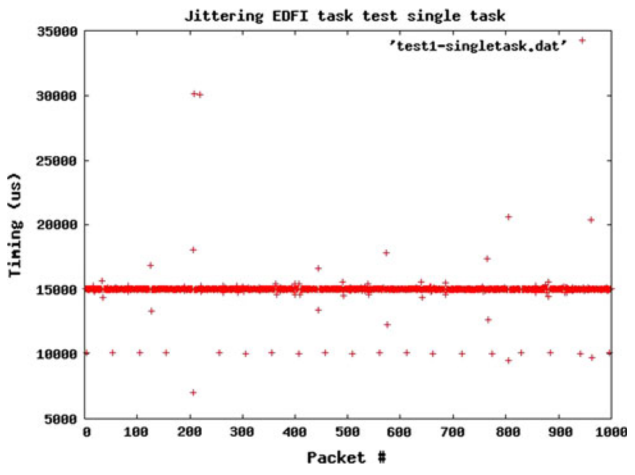

Fig. 5 The EDFI jittering, single task

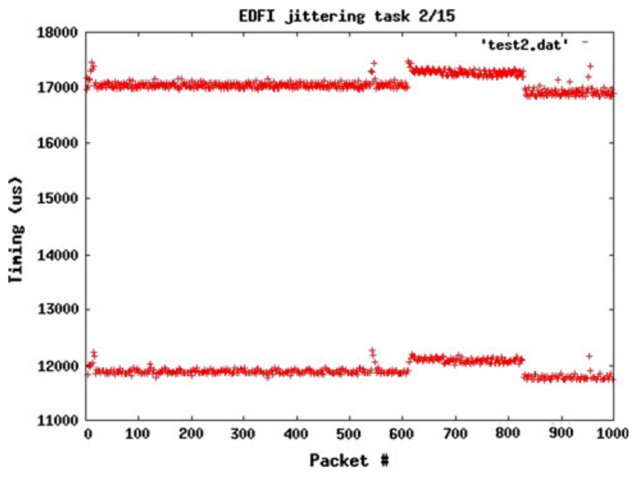

Fig. 6 EDFI jittering, 15 tasks 2/15

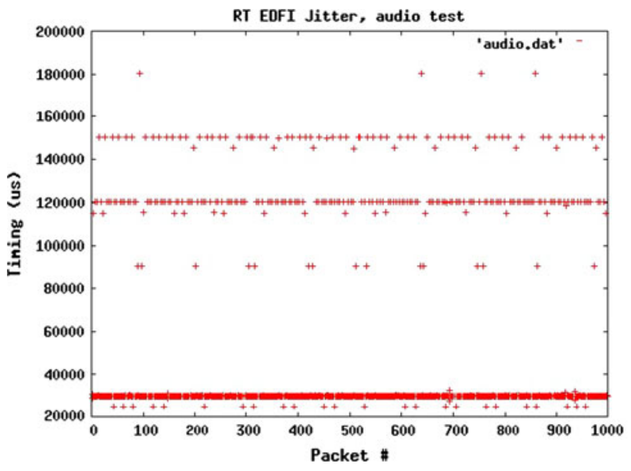

Fig. 7 Test jittering EDFI only audio (mp3 file)

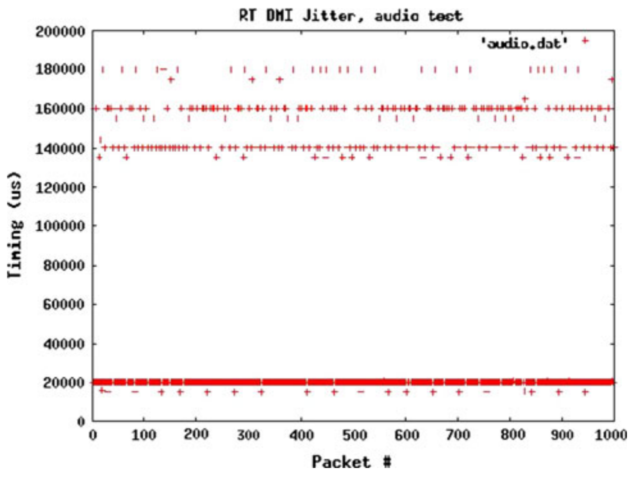

Fig. 8 Test jittering DMI only audio (mp3 file)

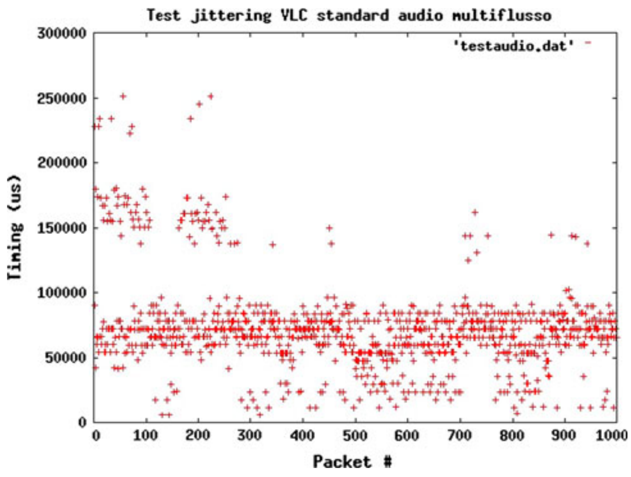

Fig. 9 VLC standard audio multiple streams (mp3 file)

3.4 Real-time VLC: jittering

Final tests have been performed on the entire designed software, i.e. the modified version of VLC and VLCKern, to measure jitter values with real data. Multimedia contents used are

- mp3 file
- DIVX video with mp3 audio
- DIVX video without audio
- XVID video with a52 audio

Both VLC with EDFI and DMI (i.e. with real-time capabilities) have been tested and compared with the unmodified version of VLC. The first test concerns the streaming of a mp3 file. Period is set to 30 ms after observing that the standard VLC performs a send every 55 ms. As shown in Figs. 7 and 8, both schedulers accomplish the task with very few non-deterministic packet delay. Standard VLC, on the contrary, shows a worst behaviour; it is possible to note (Fig. 9) how variable is the packets timing, which leads to higher jittering. Figure 10 shows performances of the unmodified VLC with a single audio stream active; the chart shows that, also for a single stream, packets' timings are much higher than timings obtained with the real-time capabilities enabled. This first set of tests allow us to assert a first result: VLC with hard real-time timings minimizes jittering, in particular when dealing with multiple streams.

Results are much different when dealing with a audio/video stream, like that one adopted in the second test. In such a situation, hard real-time system performs much better (Figs. 11, 12), either with a single stream and multiple streams, if compared with the standard version of VLC (Figs. 13, 14). As can be observed, very few packets arrive with an higher timing, so contributing to a very low jittering. Standard VLC instead presents a high variability of packets due to the lack of precise timing in submissions.

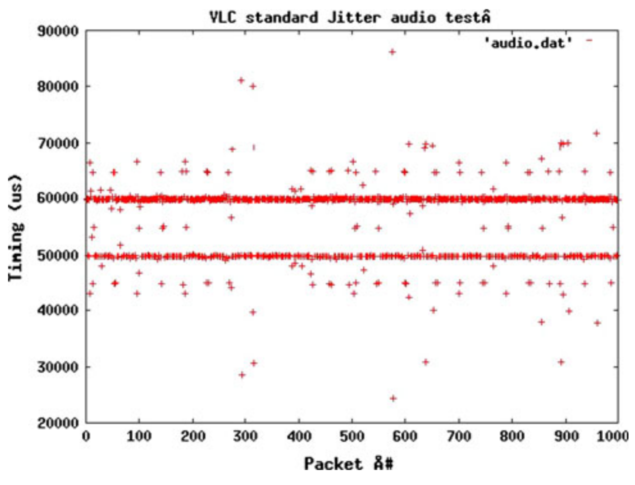

Fig. 10 VLC standard audio single stream (mp3 file)

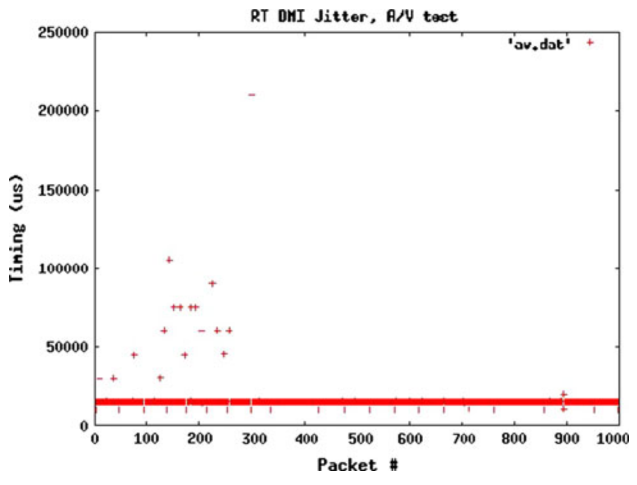

Fig. 12 Jittering audio/video with DMI

Similar results are obtained with the same video file without audio (Figs. 15, 16, 17, 18) although EDFI performs much better than DMI, showing a lower timing in packet arrival.

The aim of the last test, performed on transcoded audio/video file, is to measure how much the coding process would drop down overall performance. Also in this situation, VLC with real-time capabilities enabled works well, though some outliers can be observed in the chart (Figs. 19, 20). As expected, performance of the standard VLC gets worst as can be seen from the variability in packets timing (Figs. 21, 22). We remark that the coding process is not performed exploiting the real-time capabilities of the system, as stated above. It is clear that the lack of data to stream in the FIFOs is the main cause of the timing variability.

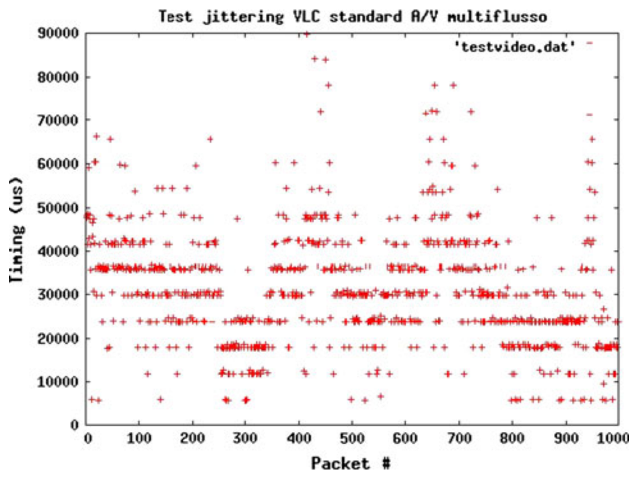

Fig. 13 Standard VLC audio/video multiple steams

4 Conclusions

The paper proposes a novel real-time streaming system for multimedia. When dealing with portable devices various

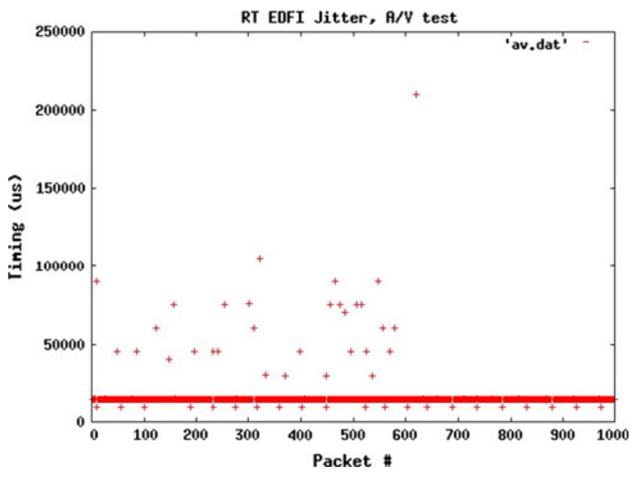

Fig. 11 Jittering audio/video with EDFI

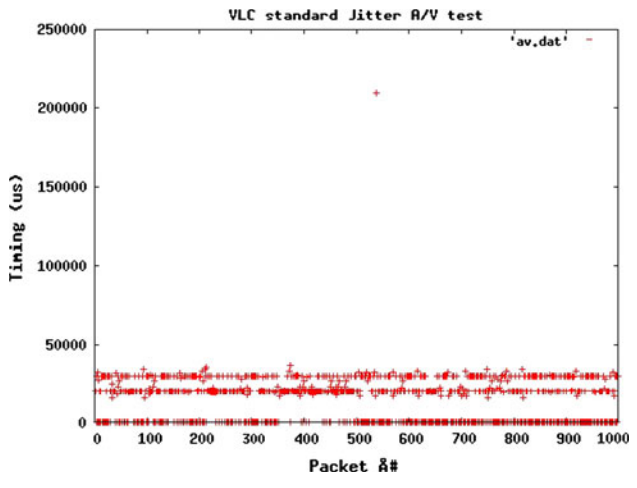

Fig. 14 Standard VLC audio/video single steam

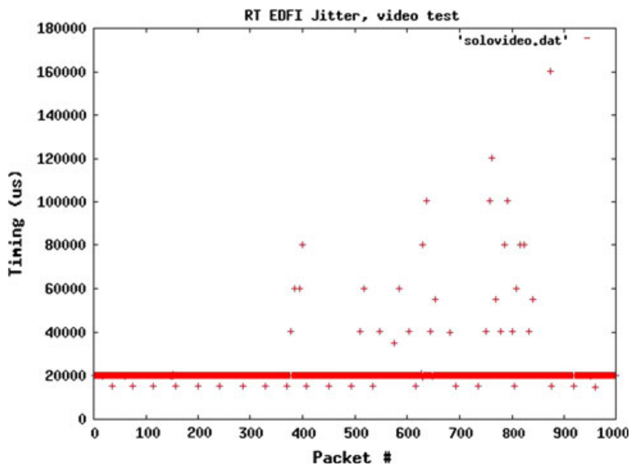

Fig. 15 Jittering video with EDFI

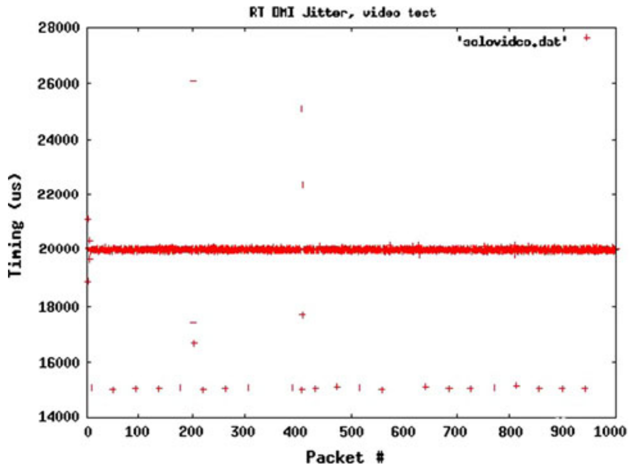

Fig. 16 Jittering video with DMI

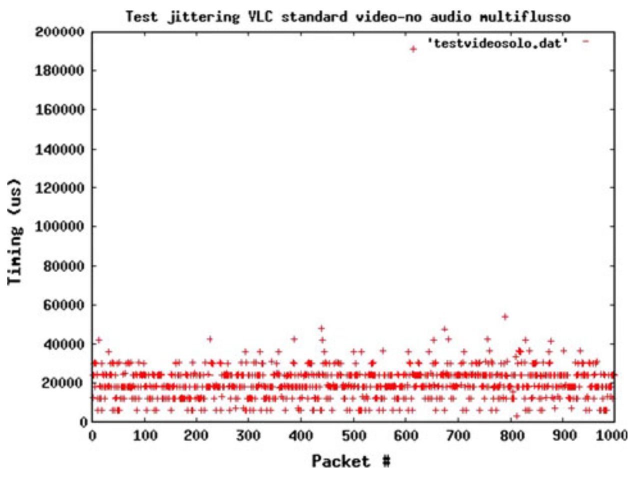

Fig. 17 Standard VLC video multiple steams

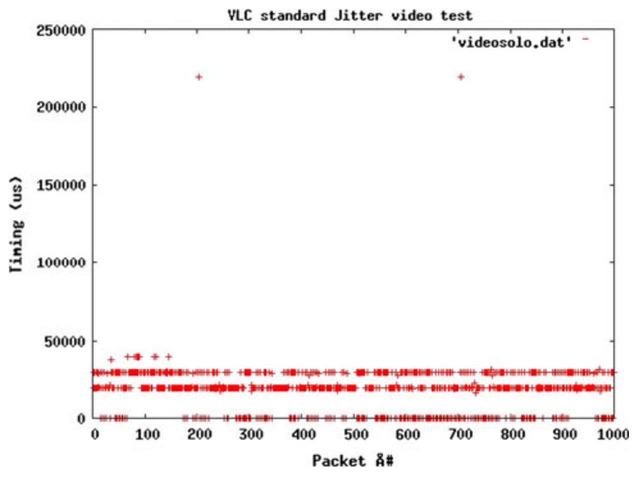

Fig. 18 Standard VLC video single steam

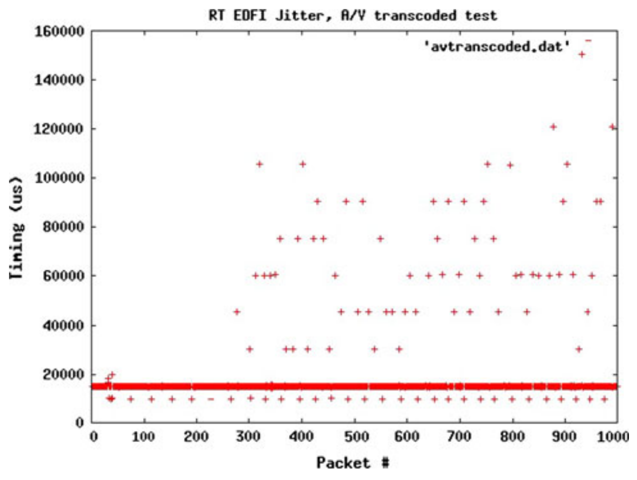

Fig. 19 Jittering transcoded audio/video with EDFI

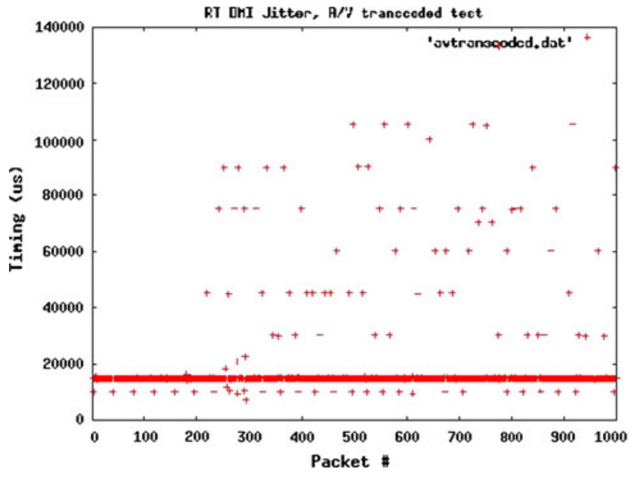

Fig. 20 Jittering transcoded audio/video with DMI

constrains have to be taken into account, like slow CPU and small amount of memory. These constraints motivated the employment of RTLinux, a hard real-time operating system, to ensure deterministic behaviour of performed tasks. Tests have demonstrated that the proposed architecture is able to reduce jittering in a network environment

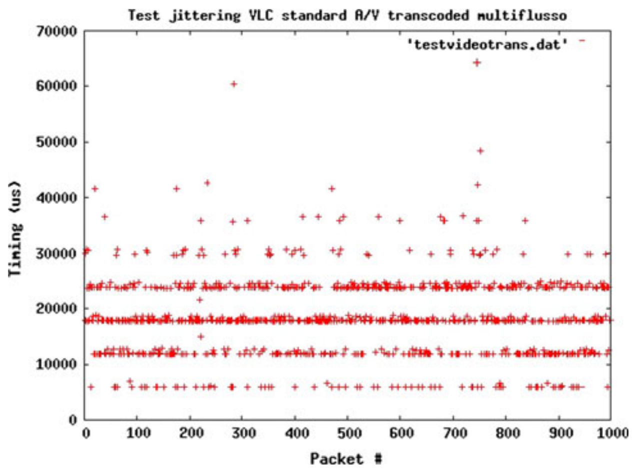

Fig. 21 Standard VLC transcoded audio/video multiple steams

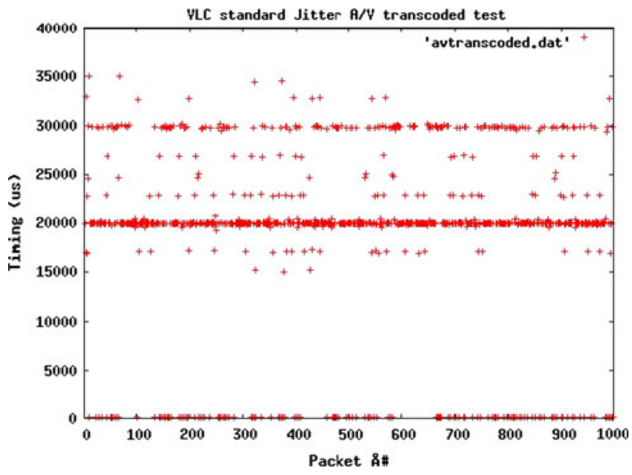

Fig. 22 Standard VLC transcoded audio/video single steam

dedicated to media streaming, due to the use of special real-time schedulers: EDFI and DMI. Both schedulers embed mutual exclusion policies, which is a very important feature when dealing with resource allocation in a real-time environment. Results obtained are so encouraging that future improvements could reside on the implementation of the proposed architecture on small systems like smart phones or PDAs. The implementation on such devices will be greatly useful in application fields where most of the computation has to be moved to mobile devices. Examples of such applications are peer-to-peer audio/video communications by means of mobile devices, video surveillance of protected areas where it is not possible to install cameras or other type of sensors, or sensor networks where, due to the reduced capabilities of each node, real-time capabilities have to be exploited to stream data over the network.

References

1. Zeng, W., Nahrstedt, K., Chou, P., Ortega, A., Frossard, P., Yu, H.: Special issue on streaming media. IEEE Trans. Multimedia **6**, 268–277 (2004)
2. Civanlar, M., Luthra, A., Wenger, S., Zhu, W.: Special issue on streaming video. IEEE Trans. Circuits Syst. Video Technol. **11**, 282–300 (2001)
3. Dan, A., Sitaram, D.: Multimedia Servers. Morgan Kaufaman (2000)
4. Mercer, C.W., Savage, S., Tokuda, H.: Processor capacity reserves: operating system support for multimedia applications. In: International Conference on Multimedia Computing and Systems, pp. 90–99 (1994)
5. Bai, L.S., Lekatsas, H., Dick, R.P.: Adaptive filesystem compression for embedded systems, design, automation and test in Europe, 2008. DATE '08, pp. 1374–1377 (2008)
6. CCITT: Recommendation G702, Red Book, 1984, Definition no.2013
7. Eichhorn, M., Schmid, M., Steinbach, E.: A realtime streaming architecture for in-car multimedia: design guidelines and prototypical implementation, ICVES 2008. pp. 157–162 (2008)
8. Mori T., Kaneko T., Moriya T., Ikeda K.: A real-time IMT-2000 audio transmission system. IEEE Trans. Consum. Electr. **47**(4), 860–866 (2001)
9. Ecole Centrale Paris: VLC developers documentation. Ecole Centrale Paris Ed. (2004)
10. Yang, D., Wang, H., Zhao, Y., Gao, Y.: A real-time streaming media file sharing mechanism based on P2P and SIP. In: 1st International Symposium on Pervasive Computing and Applications, pp. 731–736 (2006)
11. Yodaiken, V.: The RTLinux Manifesto. New Mexico Institute of Technology (1999)
12. Jansen, P.G.: Deadline Monotonic with Inheritance. University of Twente (2003)
13. Jansen, P.G., Mullander, S.J., Havinga, P.J.M., Scholten, H.: Light-weight EDF scheduling with deadline inheritance. University of Twente (2003)
14. Liu C.L., Layland J.W. Scheduling algorithms for multi-programming in a hard-real-time environment. J. ACM **20**(1), 46–61 (1973)
15. Maurer, J.M.: Building on the DMI and EDFI Foundations. Master Thesis, University of Twente (2005)
16. Jansen, P.G.: A Generalized Scheduling Theory Based on Real-Time Transaction. Master Thesis, University of Twente (2001)

Author Biographies

Alfredo Petrosino received the Laurea degree (cum laude) in Computer Science from the University of Salerno, in 1989, supervisor E. R. Caianiello. During 1989–1994 he was a fellow researcher of the Italian National Research Council (CNR). In 1995 he was a contract researcher at International Institute of Advanced Scientific Studies (IIASS). He held positions as Researcher of the National Institute for the Physics of Matter (INFM) (1996–2000), as Researcher at the National Research Council (CNR) (2000–2002) and as Senior Researcher at CNR from 2002. He is actually Associate Professor of Computer Science at the Univeristy of Naples "Parthenope". He tought at the Universities of Salerno (1991–2006), Siena (1997/1998), Naples "Federico II" (1999–2006), Naples "Parthenope" (2001–2010). He is author of more than 80 refereed papers and is associate editor of Pattern Recognition, editor of Image and Vision Computing, International Journal of Approximate Reasoning, Fuzzy

Sets and Systems. He is Senior member of the IEEE, IEEE Computational Intelligence Society, International Association for Pattern Recognition (IAPR) and Italian Neural Network Society. His research interests include image analysis and pattern recocognition with applications to target detection, remote sensing, biomedical imaging, digital movie restoration, uncertain information processing.

Marco Miralto received the Master degree cum laude in Computer Science from the University of Naples "Federico II". His research interests include operating systems, real-time processing and multimedia systems.

Alessio Ferone received the Master degree cum laude in Computer Science from the University of Naples "Parthenope" and is currently Assistant Professor in Computer Science. He is member of the IEEE and International Association for Pattern Recognition. His research interests include image processing, pattern recognition, real-time processing, and multimedia systems.