ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
CONSIGLIO NAZIONALE DELLE RICERCHE

C. Gaibisso, G. Gambosi, M. Lancia, G. Martufi, E.A. Mastromartino

MOBILE CODE IMPLEMENTATION OF THE RTP PROTOCOL IN JAVA: DESIGN CHOICES AND EVALUATION

R. 484 Novembre 1998

Carlo Gaibisso – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30 - 00185 Roma, Italy. Email: gaibisso@iasi.rm.cnr.it.

Giorgio Gambosi – Dipartimento di Matematica, Università di Roma "Tor Vergata", Via della Ricerca Scientifica - 00185 Roma, Italy. Email: gambosi@mat.uniroma2.it.

Maurizio Lancia – Centro Elaborazione Dati del CNR, P.le Aldo Moro 7 - 00185 Roma, Italy. Email: lancia@iasi.rm.cnr.it.

Giuseppe Martufi – Consultancy and Projects Group, Via della Camilluccia 693, Roma, Italy. Email: g.martufi@cpg.it.

Emiliano Antonio Mastromartino – Lucent Technologies N.S. UK, Windmill Hill Business Park, Swindon Wiltshire SN5 6PP, United Kingdom.
Email : emastromartino@lucent.com.

This paper has been presented at the "SBT/IEEE International Communications Symposium". August 9-13, 1998. Sao Paulo, Brazil.

Istituto di Analisi dei Sistemi ed Informatica, CNR
viale Manzoni 30
00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.rm.cnr.it
URL: http://www.iasi.rm.cnr.it

Abstract

In this paper, the use of a mobile code, partly interpreted language such as Java is investigated as a tool to build highly portable implementations of multimedia communication protocols. In particular, we concentrate on the implementation of the Real Time Protocol and focus our attention on three main issues: first, we evaluate the quality and accuracy of the services offered by some relevant primitive operations as they are provided on different Java platforms. Second, we concentrate on a suitable object oriented design which makes RTP functionalities both well integrated and easily accessible. Finally, we perform an overall evaluation of the efficiency of the proposed implementation both in terms of resulting packet loss and delay jitter.

1. Introduction

This paper investigates the potentiality of the JAVA platform ([2]) in the design and implementation of network protocols supporting the real time transmission of multimedia information.

The main network protocol requirements for real time multimedia applications are low transmission delays and delay jitters. The loss of information is a less crucial issue in this context.

Unfortunately, due to packets retransmission, TCP ([1]) does not effectively support the real time transmission of multimedia data. UDP ([6]) does not retransmit packets, but once again the delay jitter cannot be kept under control. Consequently, the adoption of a transport protocol, like the Real Time Transport Protocol ([3], [5], [4]), RTP in what follows, especially conceived to improve the steadfastness of the delivered packets inter-arrival times, seems to be mandatory in order to improve the quality of the user perception.

Beyond these technical aspects, there is another major obstacle to the global diffusion of multimedia technologies: software must be compiled separately to run on different platforms. Java seems to be an answer to this problem. Programs written in Java can be run whenever the Java platform is present.

In this paper we deal with the realization and the performance evaluation of a Java version of RTP.

The protocol has been first of all modeled by an object oriented approach ([7], [8]). This activity resulted in a complex object oriented architecture, which is general enough to model the characteristics and the functionality of a generic network protocol. The object oriented model has been successively "translated" in a library of Java functions whose effectiveness in smoothing the delay jitter has been tested by transmitting several audio streams through the network.

The obtained results have been very encouraging, affirming the potentiality of Java in the design and the implementation of network protocols supporting the real time transmission of multimedia information.

2. The Real Time Transport Protocol

Before entering into the the description of RTP, let us introduce some notations. As shown in figure 1, we will denote by:

- s_i the instant in which the RTP source entity starts to process the i^{th} packet;
- t_i the instant in which the RTP source entity makes the i^{th} packet available to the UDP source entity;
- a_i the instant in which the UDP destination entity makes the i^{th} packet available to the RTP destination entity;
- p_i the instant in which the RTP destination entity makes the content of the i^{th} packet available to the application.

RTP makes it possible to keep the end-to-end packet delay $p_i - t_i$ constant. Such temporal consistence is obtained at the price of an additional buffering delay $b_i = p_i - a_i$ and of the loss of information. The lower is the accepted delay, the could be the amount of lost information.

More in details, $p_1 = a_1 + b_1$, while $p_i = p_1 + T * (i-1) + f(a_1, \ldots, a_{i-1}) * T$, where T is the emission period and $f(a_1, \ldots, a_{i-1})$ is an integer valued function adapting $p_i - t_i$ to the particular load of the network ([4]). Figure 2 illustrates the main idea underlying RTP. Packets discarding is substantially due to the unpredictable variations of $f()$.

4.

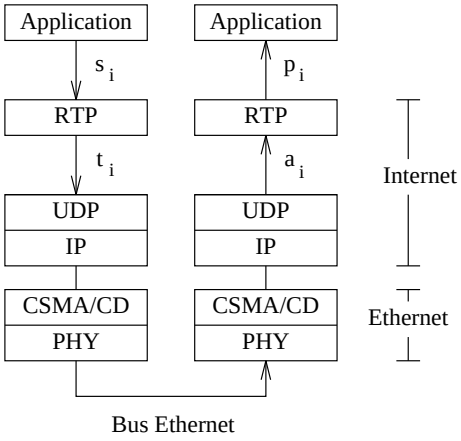

Figure 1.

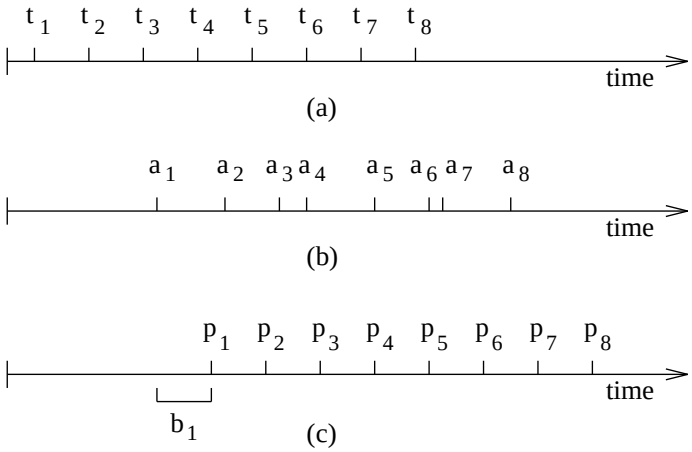

Figure 2: Diagrams (a), (b) and (c) respectively describes the temporal structure of the transmitted data, the delay introduced by the network and the way in which the RTP destination entity reconstructs the original data temporal structure.

3. The Design Activity

Several basic approaches to the implementation of RTP have been taken into considerations. Implementing RTP as an integrant part of some multimedia application have the major advantage of better fitting the particular applicative needs, but the implementation will not be, with a high probability, easily adaptable in different contexts.

Implementing RTP as a separated daemon process, presents obvious advantages from the reusability point of view, but unfortunately the implementation could to be extremely inefficient. In fact, in order to keep the implementation independent from the particular hardware/software platform, also processes running on the same host must communicate, in Java, through the network.

Thus we decided to implement RTP as a library of Java functions.

The design activity has been mainly finalized to define an object oriented architecture that is general enough to model the main functionality of a generic network protocol, thus representing a reference point in the object oriented design and implementation of protocols different from

RTP.

The defined software architecture is shown in figure 3: the Network Adapter component mainly encapsulates RTP packets in UDP packets and extracts RTP packets from UDP packets. The component is implemented as an instance of the *RTP Socket* class, whose *Send()* and *Receive()* methods make it possible to transmit and to receive RTP packets.

The RTP Packet Processing is the core component of the system and is further subdivided into the RTP Packet Storing and the RTP Packet Scheduling components.

As shown in figure 4, the component mainly consists in a daemon, which, by repeatedly activating the *Receive()* method of the *RTP Socket* class, accesses to the RTP packets received by the Network Adapter component, and in a queue of RTP packets, represented by an instance of the *Storage* class, whose *Send()* method, when invoked by the daemon, assigns to the particular RTP packet the current value of the system clock and inserts the packet into the queue.

The RTP Packet Scheduling component, as shown in figure 5, is once again mainly implemented as a daemon, which, by repeatedly activating the *Receive()* method of the *Storage* class, accesses the queue maintained by the RTP Packet Storing component and as an instance of the *Scheduler* class, which represents a priority queue where the packets are organized according to their associated presentation time. When invoked by the daemon, the *Send()* method of the *Scheduler* class, validates the packet, assigns to the packet its presentation time and inserts the packet into the priority queue.

The presentation time for each received packet is computed by an algorithm represented as an instance of the *Delay Algorithm* class. It is worth noting that, since the characteristics of such an algorithm are strictly dependent from the particular application, it is not considered as a component of the protocol.

Finally, let us consider the RTP API component, which implements the functionality required:

- to establish an RTP session;
- to package together the payload and the header of each RTP packet while sending packets;
- for each received RTP packet, to extract the payload from the packet and to make such payload available to the application according to its presentation time.

The component is represented as an instance of the *RTP Service Access Point* class, *RTP SAP* in what follows (see figure 6). In order to open a new RTP session the application has to create a new instance of the *RTP SAP* class, by specifying the UDP socket by which data will be delivered to the network.

To transmit data, the application invokes the *Send()* method of the corresponding *RTP SAP* instance, specifying the transport address, the relative position of the RTP packet inside the transmitted flow of packets (time stamp) and the relative position of the first sample of the packet inside the flow of transmitted samples (sequence number).

4. Test Activity

A preliminary testing activity has been finalized to investigate the effectiveness of the implementation of some functionalities, provided by the Java Virtual Machine and of relevance in this framework, on different platforms (Alpha/OSF, Sun Ultra/Solaris, Pentium/Windows 95).

First of all we evaluated the temporal accuracy in suspending the execution of threads, i.e. the precision of the *Sleep()* method, since it has great great influence the precision of the additional buffering delay and of the emission rate. Unfortunately, the Pentium/Windows 95 platform is

6.

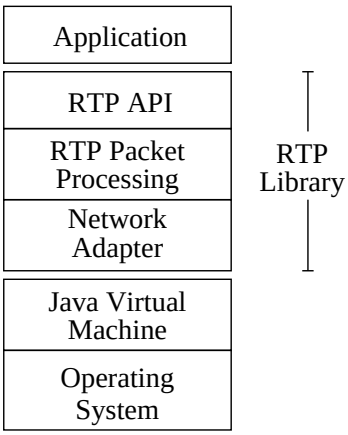

Figure 3.

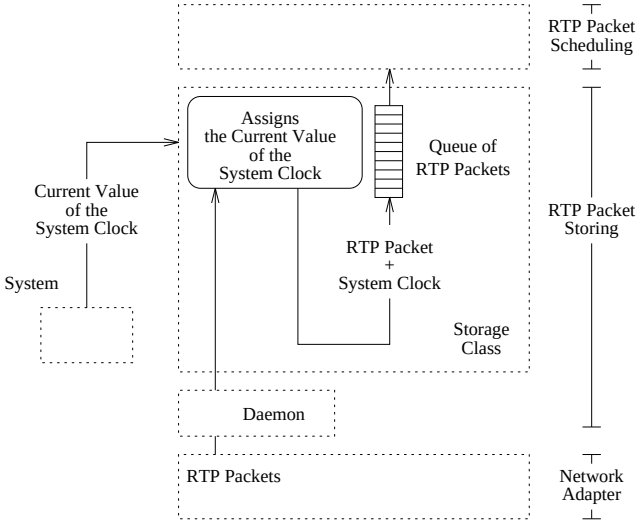

Figure 4: the RTP Packet Storing component.

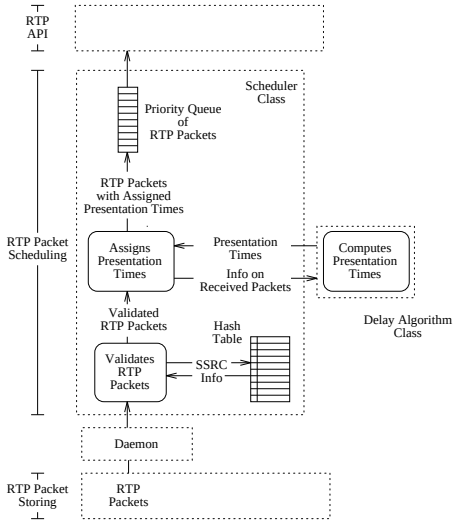

Figure 5: the Packet Scheduling component.

so imprecise from this point of view that we decided to exclude it from further investigations. The Sun-Ultra/OSF platform performs very well for temporal intervals which are longer than 20 msec and integer multiple of 10 msec. Finally, the Alpha/OSF platform has an excellent behaviour for any temporal interval longer than 8 msec.

After that we investigated, by several tests, the Java reliability in transmitting packets through the network. Each test consists in the transmission of 1,000 RTP packets, while the size and the rate of emission of packets change from test to test. Both the considered platforms very well performs from this point of view, since for any rate of emission lower than 200 packets per second, less than the 3% of packets is lost.

Finally we investigate the policies of threads scheduling implemented by the different platforms considered. Tests have been conducted by a very simple program, activating two different threads. Each thread consists in an infinite loop adding 1 to the value of a local variable at each cycle. After 10 seconds the program stops and returns the value of the two variables.

Unfortunately both the considered platforms do not correctly implement a time sharing policy. In fact, just one thread is executed, the one with higher priority, if different priorities have been assigned to the threads. Since, the Java implementation of RTP is based on the concurrent execution of threads, in order to avoid unpredictable behaviors of the protocol on different platforms, we never assign threads the same priority, and CPU bounded threads have been assigned the lowest priorities.

Finally we tested the effectiveness of the RTP library in smoothing the delay jitter introduced by the network. The system used for our tests is composed by two workstations connected by an Ethernet LAN (10 Mbps nominal rate). In order to evaluate the effectiveness of the RTP Java implementation, we consider the unidirectional transmission of an audio signal through the network. A third workstation is connected to the LAN, running a Gateway Process, which simulates different traffic conditions by capturing each RTP packet and introducing an additional delay, according to some distribution function, to that required for its transmission.

For each test we evaluated the number of packets lost by the destination RTP entity, and for each transmitted packet, we measured:

- $t_i - s_i$, i.e. the time required by the RTP source entity to process the packet;
- $a_i - a_{i-1}$, which is indicative of the delay jitter introduced by the network;
- $p_i - p_{i-1}$, which is indicative of the accuracy by which the RTP destination entity maintains the temporal structure of transmitted data.

We do not take into account $t_i - t_{i-1}$, since this value is greatly influenced by the precision of the *Sleep()* method and we already investigated such aspect.

During all the implemented tests, the number of lost packets and, for each transmitted packet, $t_i - s_i$ resulted to be negligible.

A first series of tests have been conducted where the RTP source entity was the only source of traffic on the network. These tests have been mainly finalized to investigate the maximum emission rate tolerable by our Java implementation of RTP. We considered different emission rates starting from 100 packets per second. For each test 1,000 packets have been transmitted whose dimension has been selected, for each considered emission rate, to determine a bandwidth requirement of 64 Kbps, which is a typical requirement for the transmission of an uncompressed audio flow.

The Alpha/OSF platform has a quite satisfactory behavior if the emission rate if fixed to 10 packets per second, as shown in figure 7 where the density functions of the inter-arrival ($a_i - a_{i-1}$) and inter-presentation times ($p_i - p_{i-1}$) are plotted.

The performances of the platform improve as long as the emission rate decreases and are absolutely excellent if 1 RTP packet is transmitted each 40 msec, which is a quite usual emission rate for the transmission of both audio and video streams (see figure 8).

Unfortunately, the Sun-Ultra/Solaris platform has a very bad behavior if the emission rate is fixed to 1 packet each 10 msec, as shown in figure 9. This is substantially due to a not precise implementation of the *Sleep()* method. In fact, the performances of the platform are very interesting if the emission rate is fixed to 50 packets per second, as shown in figures 10, and comparable to those of Alpha/OSF if such emission rate is halved.

Once verified the efficiency of the RTP library in processing typical audio and video data flows, we investigate its effectiveness in smoothing the delay jitter introduced by the network. The presence of additional traffic onto the network has been simulated by the Gateway Process. A Gaussian distribution, with mean 400 and standard deviation 16, and a uniform distribution, into the interval of integers $[385, 415]$, are the distributions according to which the additional delays have been introduced. This choice was done to limit the number of packets which are delivered in a sequence different from that of emission, number which is in practice very low. In any case the delay jitter introduced in this way is critical enough to guarantee significant experimental results. In all the tests the rate of emission has been fixed to 25 packets per second. On both the Sun-Ultra/Solaris (see figures 11 and 12 respectively for the uniform and the Gaussian distributions) and the Alpha/OSF (see figures 13 and 14 respectively for the uniform and the Gaussian distributions) platforms RTP proved its effectiveness in smoothing the delay jitter introduced by the network.

5. Conclusions

In this paper we focused on the realization of a Java version of RTP: first, we evaluate the quality and accuracy of the services offered by some relevant primitive operations as they are provided on different Java platforms. Then, we concentrate on a suitable object oriented design which makes the RTP functionality both well integrated and easily accessible. Finally, we perform an overall evaluation of the efficiency of the proposed implementation both in terms of resulting packet loss and delay jitter.

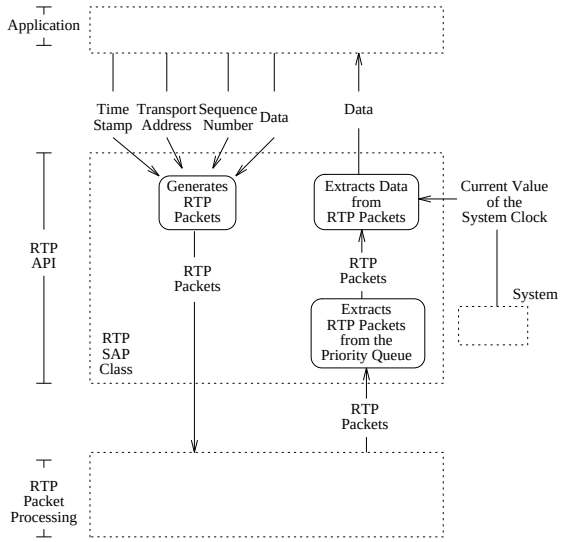

Figure 6: the RTP API component.

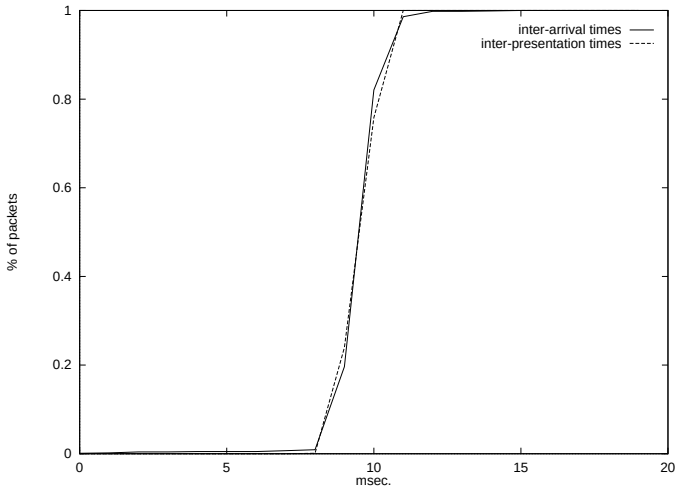

Figure 7.

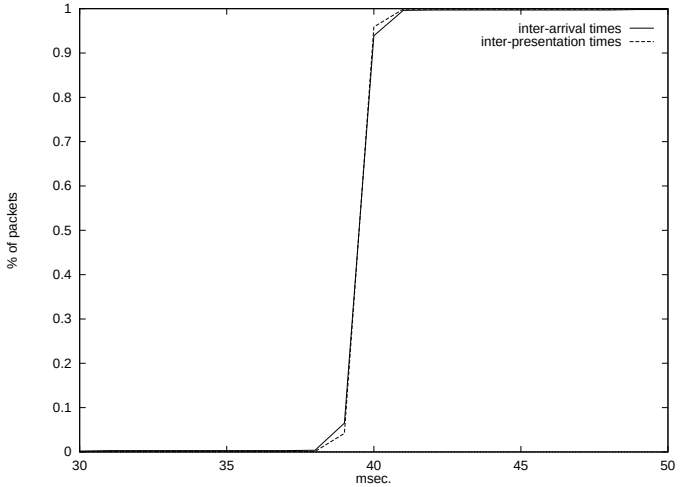

Figure 8.

10.

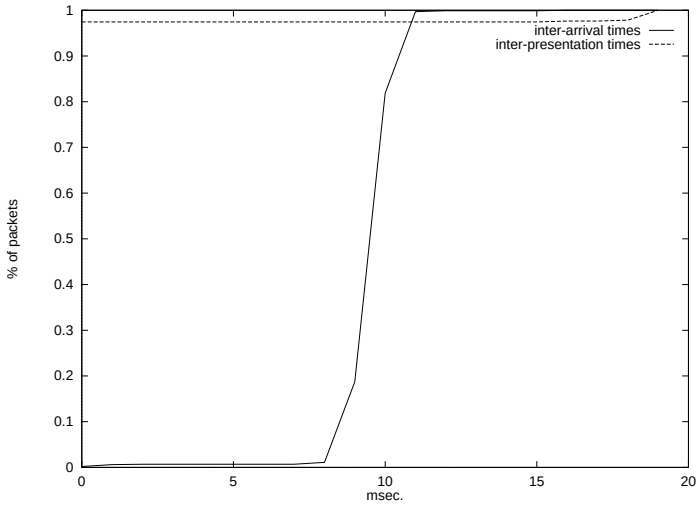

Figure 9.

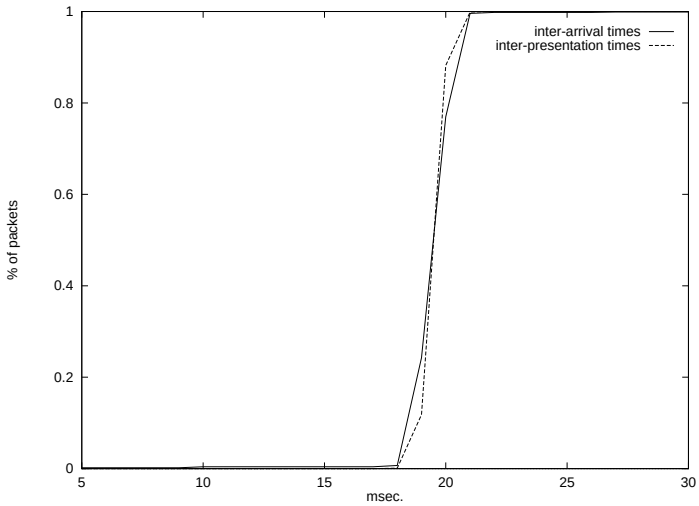

Figure 10.

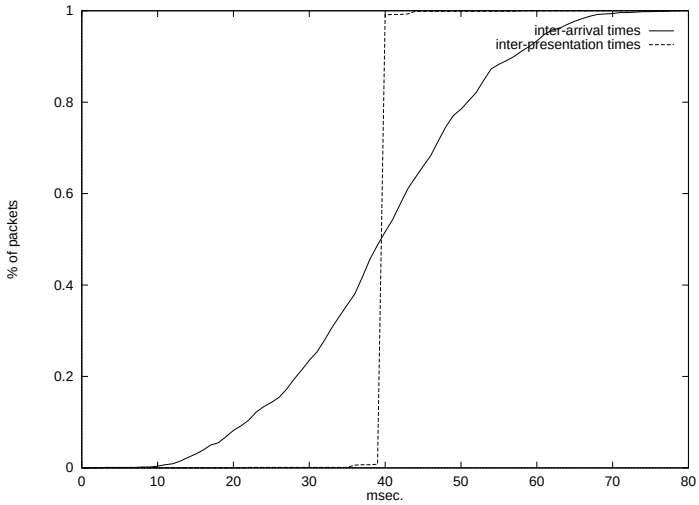

Figure 11.

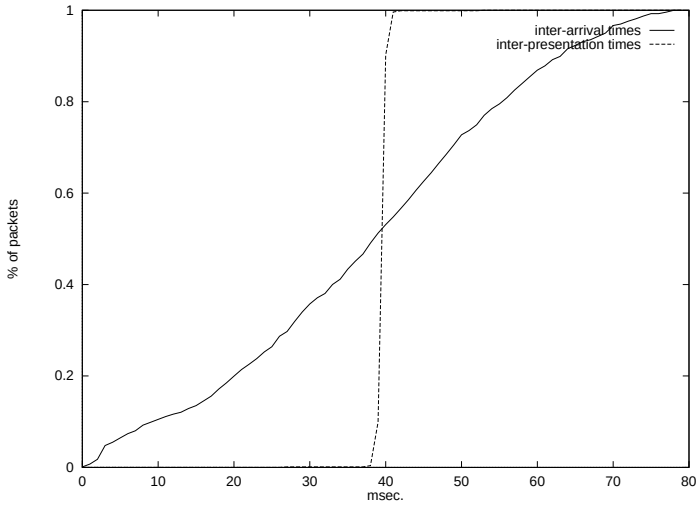

Figure 12.

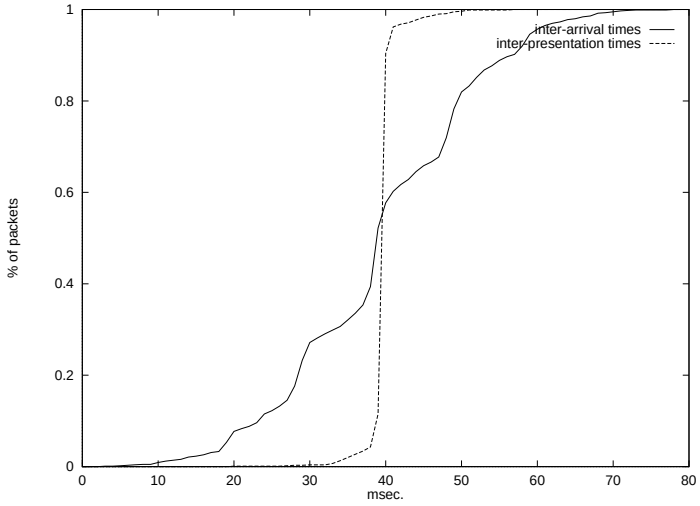

Figure 13.

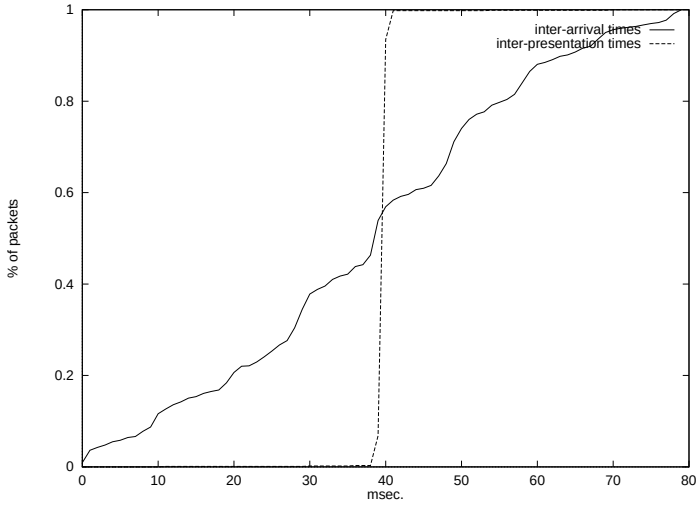

Figure 14.

12.

References

[1] D.E. Comer: Internetworking with TCP/IP (2-nd ed.)". Prentice-Hall (1991).

[2] D. Kramer: The Java Platform, A White Paper. JavaSoft, May (1996).
URL:http ://www.javasoft.com.

[3] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson: RTP: A Transport Protocol for Real-Time Application. RFC 1889, IETF, January (1996).

[4] H. Schulzrinne: Issues in Designing a Transport Protocol for Audio and Video Conference and Other Multipartecipant Real-Time Applications. Audio Video Transport Working Group, May (1994).

[5] H. Schulzrinne: RTP Profile for Audio and Video Conferences with Minimal Control. RFC 1890, IETF, January (1996).

[6] J. Postel: UDP: User Datagram Protocol. RFC 766.

[7] G. Bakker: OMT Object Model. February (1996).
URL:http//wwwedu.cs.utwente.nl/ẽnting/miso/.

[8] G. Booch: Object-Oriented Analysis and Design (2-nd ed.). Addison-Wesley 1994.