A Multicast Transport Protocol Design Methodology: Analysis, Implementation and Performance Evaluation

Pilar Manzanares-Lopez, Juan Carlos Sanchez-Aarnoutse,
Josemaria Malgosa-Sanahuja, Joan Garcia-Haro
Department of Information Technologies and Communications,
Antiguo Cuartel de Antigones, E-30202, Cartagena, Spain
Email: {pilar.manzanares, juanc.sanchez, josem.malgosa, joang.haro}@upct.es

Abstract— **In this paper, we propose and analyze a multicast application called SOMA (SynchrOnous Multicast Application) which offers multicast file transfer service in an asymmetric intra-campus environment (several interconnected wired and wireless LAN networks interconnected through few routers).**

For efficient bandwidth utilization, SOMA transmits IP packets by using multicast addressing. We also propose a complete multicast transport protocol involving both, the flow and error correction algorithms. The protocol adapts the window size and the overall application transfer bitrate to the minimum network capacity, allowing synchronism and reacting quickly when congestion arises at any network router.

The application behavior has been tested by simulation in a mixture of wired and wireless intra-campus networks, and intensively studied by experimentation in an intra-campus environment composed of a wired Fast Ethernet and a wireless 802.11b Ethernet connected through an Access Point router. In addition, we develop a mathematical model to validate analytically the most important protocol parameters. The methodology employed to define, analyze and evaluate this multicast protocol is, indeed, another contribution of the work and can be easily extended to other multicast protocols.

Keywords: Multicast, flow and congestion control, transport protocol.

I. Introduction

The use of multicasting within a network has many strengths. Multicast minimizes the link bandwidth consumption because no multiple unicast connections (a connection for each receiver) are needed to send the information. In addition, the maintenance of only a multicast connection, instead of several unicast connections, also reduces the sender and router processing and the delivery delay. However, IP multicast technology adds additional tasks to the network devices. Network elements must be able to dynamically manage the multicast group composition and also they must implement adequate routing protocols to handle IP multicast traffic.

In spite of MBone creation (a virtual multicast network which allows multicast packets to travel through routers that only handle unicast traffic), IP multicast has not become an extended Internet service. A wide range of subjects have stalled the widespread use of IP multicast. However, the widespread use of Ethernet standard (inherently multicast compliant) as the technology to interconnect electronic devices, the proliferation of wireless networks based on IEEE 802.11 standard (also multicast compliant) and the use of Linux (an open source operating system which implements most of the network tasks) have re-opened the scenarios where IP multicast technology could be used. The work presented in this paper is focused on a new IP multicast scenario, that we have called intra-campus scenario. Concretely, we propose, analyze, implement and test a SynchrOnous Multicast Application called SOMA to synchronously transfer a large amount of data (files and hard disk partitions) from a server to a group of clients. It is specially featured to operate in an intra-campus environment, which is defined as several LANs (wires or wireless) interconnected through few routers.

To do it, the socket interface has been selected since it offers a relatively easy tool to define and to create a layer 3 multicast communication. However, multicast transport protocol requirements (flow and congestion control, error correction, etc.) are more complex than in a point-to-point one. Since TCP is a unicast oriented protocol, it cannot be directly used in a multicast environment. Therefore, the choice of an adequate transport protocol is the key issue in the multicast application development. As a consequence, the main contribution of the paper is the definition and test a synchronous multicast transport protocol to be used by our SOMA application in an asymmetric intra-campus environment.

Obviously, our solution requires multicast routing facilities, but this is not a problem since involved routers are located into our administrative domain. In spite of its simplicity, our proposed protocol provides the main tasks of a transport protocol: Efficient and simple flow control, congestion control and error correction algorithms.

SOMA protocol simplicity makes possible an easy codification and a feasible mathematical analysis of the main key features which enables the optimization of some parameter values. It has been written in C language using

This work has been supported by the Spanish Research Council under project ARPaq (TEC2004-05622-C04-02/TCM)

standard Linux kernel routines.

We also describe how to capture and post-process its generated network traffic. In particular, we have used a Linux kernel architecture to improve the packet capture process (LSD Linux Packet Filter) and we have modified two well-known open-source sniffers (tcpdump and ethereal) to interpret our protocol packets.

The paper is organized as follows. Section 2 presents a survey of multicast transport protocol evolution. Section 3 describes the prossed protocol. Section 4 analytically obtains the key protocol parameters. Section 5 presents our test results in a mixed wired and wireless LAN scenario. Finally, section 6 concludes the paper.

II. Related work

The extreme complexity associated to the definition of a unique and global multicast transport protocol that meets the requirements of all types of multicast applications leads the designers to several approaches for the transport protocol. The most widespread solution consists of the definition and codification of a specific multicast transport protocol which fits the requirements of a specific application.

Several multicast transport protocols were proposed to meet the requirements of delay-sensitive, real-time interactive applications, such as RTP/RTCP [1] to support multi-party multimedia conferencing tools, SRM [2] and TRM [3] to support distributed whiteboard tools, etc. These applications can tolerate a certain degree of data loss, but they are sensitive to packet delay variance.

On the other hand, other protocols were proposed to meet the requirements of reliable data distribution services, such as multipoint file transfer. These applications are not delay-sensitive, but require that the information is entirely received, or else the transfer fails. The Muse protocol [4] (which was developed to multicast news articles on the MBone), MDP [5] (the evolution of a protocol used in disseminating satellite images over MBone) and MFTP [6], RMTP [7] and TMTP [8] (other protocols for reliable one-to-many data transmission) are examples of this kind of protocols. Most of them are designed to work in the MBone when the number of receivers is too large (thousands of receivers). To reach scalability and, therefore, to solve the feedback implosion problem, some of them define complex hierarchical topologies and they even introduce some non-layer 3 functionality into the network devices.

In recent years, the IETF Reliable Multicast Transport (RMT) group [9] has taken a different approach to design a set of multicast protocols to suit the variety of applications and service requirements for one-to-many and many-to-many communications. Instead of defining and standardizing multiple protocols, they are defining "building blocks" and two "protocol instantations" [10]. Building blocks are modular components that solve a particular functionality common to multiple protocols. They include, among others, forward error correction schemes, two congestion control algorithms (PGMCC and TFMCC) and generic mechanisms for router assistance. Protocol instantations define how to combine one or more building blocks to create a working protocol. The first one is the Negative-Acknowledgment Oriented Reliable Multicast (NORM), which describes the framework and common components relevant to multicast protocols based primarily on NACK operation for reliable transport. The second one is the Asynchronous Layered Coding (ALC) protocol, which describes a massively scalable reliable content delivery protocol. ALC uses a multiple rate congestion control building block that is feedback free. A sender sends packets in the session to several channels at potentially different rates and receivers just adjust their reception rates individually by joining and leaving channels associated with the session. ALC uses the FEC building block to provide reliability.

As we have indicated in the Introduction section, our objective is to define a synchronous multicast transport protocol to be used by our SOMA application in an asymmetric intra-campus environment. Building blocks proposed by the RMT group are too complex since they cover a general multicast transport scenario. Therefore, we have recovered the first protocol design approach. We propose a complete, compact, and also simple SOMA transport protocol to be used by our SOMA application.

III. SOMA description

SOMA is a multicast application designed for transmitting synchronously large files and hard disk partitions to a set of clients. This protocol is an extension and enhancement of a previous work [11] to cover asymmetric intra-campus networks. SOMA introduces a transmission window to improve the obtained throughput. We also implement an improved flow control mechanism that allows SOMA to be used when unequal capacity networks are interconnected (asymmetric networks). This is a frequent situation when wireless and wired network coexist. Moreover, in wireless networks (whose proliferation has not doubt, nowadays), the available throughput does not only depend on the number of applications which share the network. In fact, it changes depending on the network capacity, which depends on the signal to noise ratio and other physical parameters. Therefore, it is important to design an adequate flow control mechanism that quickly reacts when congestion arrises.

The application employs IP multicast addressing and implements its own transport protocol over UDP. Thereby, port multiplexing and error checking facilities are automatically resolved by the kernel. However, due to the UDP simplicity, the flow control and error recovery mechanisms have to be implemented to fit the transport layer requirements of our application. For this reason, we alternatively refer to SOMA as an application or as a transport protocol.

A. Overall protocol description

SOMA splits the transmission process from the server to the multicast group of clients into two differentiated phases. In the **first one**, that it is also called multicast phase, the server multicasts a set of data packets (a transmission window) to all clients. The clients store the payload and, only when the last packet of the transmission window is detected, they contend to confirm the received set of data packets by sending an ACK packet. Although in this phase the server never retransmits any data packet, a client issues a NACK packet when packet losses are detected and it also saves an error mark instead of the packet payload. As it is described in section III-C, the feedback information (ACK and NACK packets) received at the server is used to resize the transmission window in a appropriately way. The above procedure is repeated until the file is completely transferred.

The **second phase** (also called unicast phase), which is focused on error correction, starts when the entire file has been transmitted. Each receiver re-scans its file looking for error marks. If one error mark is found, the client delivers a unicast REPAIR-REQUEST packet towards the server. The server answers the client sending a unicast REPAIR-RESPONSE packet.

Error correction tasks are relegated to a final phase since current network technologies offer low error rates. This assumption avoids a complex protocol design, solving infrequent packet losses during the transmission.

One of the main SOMA protocol features is synchronicity. The proposed flow control algorithm, which is explained and tested below, adapts the server transmission rate to the slowest bitrate of a participant network. Therefore, all the clients receive the information at the same time.

SOMA is mainly used to replicate a large amount of information. In this scenario, the reduction of packet flows to only one multicast data flow is the objective, and synchronicity is thus, a consequence but not the main concern. However, disabling the error correction phase, the synchronicity feature converts SOMA into a useful and simple multicast transport protocol also for on-line applications.

B. Proposed header

The SOMA packet header consists of 4 fields:

- The Sequence Number (SN, 4 bytes long) used mainly for packet loss detection.
- The Type Of Packet (TOP, 1 byte), which distinguishes a DATA, a ACK, a NACK, a REPAIR REQUEST or a REPAIR-RESPONSE packet.
- The Payload Length (PL, 2 bytes) indicates the total packet length in bytes.
- The Last Window Sequence Number (LWSN, 4 bytes) is used to indicate the last packet of a given window and then to implement a effective feedback reduction scheme.

The header is followed by the data payload. Although theoretically the payload could encapsulate to 2^{16} bytes of data, we use a payload size of 512 bytes in our implementation because UDP packet usually does not exceed 512 bytes and also because 512 is the usual disk sector size.

C. Flow control algorithm

After a data packet is sent by the server, it starts a timer called timeout and immediately it waits until an ACK packet for each participating LAN (not for each client) acknowledges the window or until the timer expires. If the timer expires before the ACKs are received, its value is increased multiplying it by a factor of α ($\alpha > 1$). But if the window is confirmed in time, the timer value is decreased as denoted by expression (1)

$$T_{out} = max\{\frac{Tout}{\beta}, dflt\_Tout\} \tag{1}$$

Where $\beta > \alpha > 1$ and *dflt_Tout* is the bottom threshold value. The server repeats this above operation until the file is completely transferred.

A window is only confirmed when the server receives one ACK for each participating LAN, ensuring synchronism among all multicast clients. Therefore, if one of the networks suffers congestion, the timeout value is increased and therefore, the data transmission rate decreases. When congestion disappears, the timer redefinition allows to increase the transfer rate again.

To improve the flow control reaction, it is convenient that not only the timer but also the window size changes appropriately. To accomplish this, just before sending the next data window, the server modifies the window size as follows:

- If the expected ACKs associated to this window have been received before the timer expires, the server increases the window size in one unit.
- If the timeout expires, the server decreases the window size in one unit.
- For each NACK that indicates a different packet loss (only the first NACK indicating a particular packet loss is considered), the server decrements the window size in one unit.

On the other hand, the clients are waiting for data packets. When a packet arrives, each client extracts the sequence number and compares it with the expected value:

- If the sequence number is the expected one, the client stores the payload and updates the sequence number.
- If the sequence number is greater, the client detects packet losses and sends a NACK with the sequence number of the received data packet. Simultaneously, it finds out the number of lost packets and it stores an error mark for each one. Finally, it also stores the data contained in the received packet.
- If the sequence number is smaller, the data packet is discarded.

In addition, if the SN matches with the LWSN value, the client competes for sending an ACK to confirm the entire window issued by the server (see the feedback implosion reduction below).

D. Feedback implosion reduction

To reduce the amount of ACK feedback packets in the network, a client must wait a random period called ARTP (ACK Random Time Period) before sending an ACK and simultaneously, it listens if another client belonging to its LAN is transmitting the same ACK. If the ARTP expires and the ACK has not been received, the client generates and multicasts its own ACK. The rest of clients will receive the ACK but only the clients at the ACK sender side (belonging to the same subnetwork) will disable its own ACK transmission. The ARTP value is obtained from a uniform probability distribution function ranging between zero and $ARTP_{max}$. Thereby, only one ACK for each participant LAN is sent to the server, independently of the number of clients.

The effective ACK generation time is a random variable defined as: $ARTP = min(ARTP_1, \cdots, ARTP_n)$, where $ARTP_i$ is the random time value obtained by each client and n is the number of clients attached to the same LAN.

As it has been described above, each random value $ARTP_i$ is uniformly distributed. Therefore, the associated probability density function (*pdf*) and cumulative distribution function (*cdf*) are given by

$$f(artp_i) = \frac{1}{k};\ 0 \leq artp_i \leq ARTP_{max} \tag{2}$$

$$F(artp_i) = P(artp_i \leq ARTP_i) = \frac{artp_i}{k}; \tag{3}$$

For sake of simplicity, k is the maximum ARTP value.

The cumulative distribution function of the $ARTP$ variable can be obtained as follows:

$$F_n(artp) = P(ARTP \leq artp) =$$
$$P(ARTP \leq artp|ARTP_1 \leq artp_1)P(ARTP_1 \leq artp_1)+$$
$$P(ARTP \leq artp|ARTP_1 \geq artp_1)P(ARTP_1 \geq artp_1)$$

but

$$P(ARTP \leq artp|ARTP_1 \leq artp) = 1$$
$$P(ARTP \leq artp|ARTP_1 \geq artp) = F_{n-1}(ARTP)$$

Therefore,

$$F_n(artp) = \frac{artp}{k} + F_{n-1}(artp)\left(1 - \frac{artp}{k}\right) \tag{4}$$

due to the recursive nature of equation (4) and the polynomial form of equation (3), it can be stated that

$$F_{n-1}(artp) = c_1\frac{artp}{k} + c_2\left(\frac{artp}{k}\right)^2 + \ldots + c_{n-1}\left(\frac{artp}{k}\right)^{n-1}$$

Substituting the above equation into equation (4), the obtained expression is also polynomical,

$$F_n = (c_1+1)\left(\frac{artp}{k}\right) + (c_2-c_1)\left(\frac{artp}{k}\right)^2 + \ldots$$
$$+(c_{n-1}-c_{n-2})\left(\frac{artp}{k}\right)^{n-1} - c_{n-1}\left(\frac{artp}{k}\right)^n \tag{5}$$

An exhaustive observation of the coefficients leads us to the Tartaglia Triangle. Then, the *cdf* can be expressed in a more compact way,

$$F_n(artp) = \sum_{i=1}^{n}(-1)^{i+1}\binom{n}{i}\left(\frac{artp}{k}\right)^i \tag{6}$$

and the probability density function is

$$f_n(artp) = \frac{d}{dartp}F_n(artp) =$$
$$= \sum_{i=1}^{n}\frac{i}{k^i}(-1)^{i+1}\binom{n}{i}artp^{i-1} \tag{7}$$

Now, the mean value of ARTP is given by:

$$E[ARTP] = \int_0^k artp \cdot f(artp)dartp =$$
$$= \int_0^k \sum_{i=1}^{n} i(-1)^{i+1}\binom{n}{i}\left(\frac{artp}{k}\right)^i =$$
$$= k\sum_{i=1}^{n}\frac{i}{i+1}\binom{n}{i}(-1)^{i+1} \tag{8}$$

due to the following properties

$$F(k) = \sum_{i=1}^{n}\binom{n}{i}(-1)^{i+1} = 1$$
$$\frac{1}{i+1}\binom{n}{i} = \frac{1}{n+1}\binom{n+1}{i+1}$$

equation (8) can be expresed as

$$E[ARTP] = k\left[1 - \sum_{i=1}^{n}\frac{1}{(i+1)}\binom{n}{i}(-1)^{i+1}\right] =$$
$$= k\left[1 - \sum_{i=1}^{n}\frac{1}{n+1}\binom{n+1}{i+1}(-1)^{i+1}\right] =$$
$$= k\left[1 - \frac{1}{n+1}\sum_{i=1}^{n}\left[\binom{n}{i} + \binom{n}{i+1}\right](-1)^{i+1}\right]$$

However, due to the signal alternance of the sumatory terms, all of them are cancelled except the first one. Therefore,

$$E[ARTP] = k\left[1 - \frac{1}{n+1}\binom{n}{1}\right] =$$
$$= \left(1 - \frac{n}{n+1}\right) \cdot ARTP_{max} \tag{9}$$

Which is clearly decreasing with the number of clients.

Figure 1 briefly summarizes the usual protocol operation. The server sends a set of data packets, each time increasing the window size until W size is reached. At this point, the timer expires just before all ACKs are received, probably because at some network point

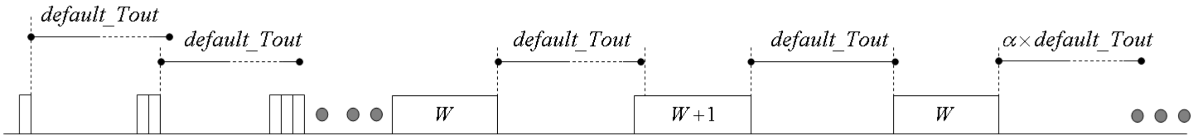

Figure 1. Window size evolution in an asymmetric network environment

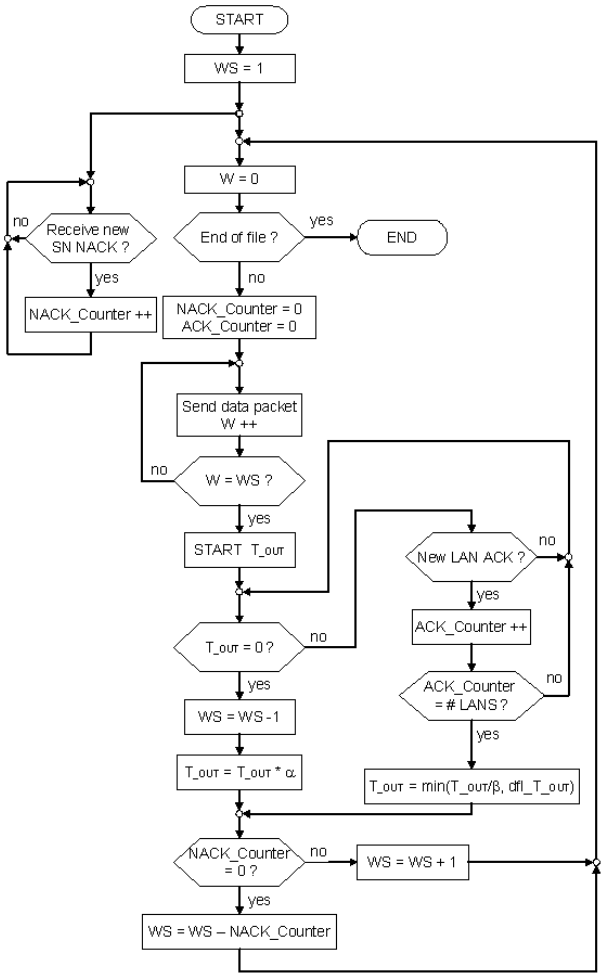

Figure 2. Flowchart of the multicast stage of the server.

congestion arises. The server reacts quickly increasing the timer value and decreasing the window size. It is clear that for protocol consistency, the timeout must be greater than the mean ARTP value ($\overline{ARTP}$).

E. Formal protocol description

In above subsections, the proposed protocol has been described in detail using natural language and narrative description. However, in order to facility its understanding and implementation, it is important to expose the protocol in a formal way. Figures 2 and 3 summarize the server and the client behavior using flowchart diagrams. It can be seen that basic set instructions of any programming language could be used to implement both the server and the client.

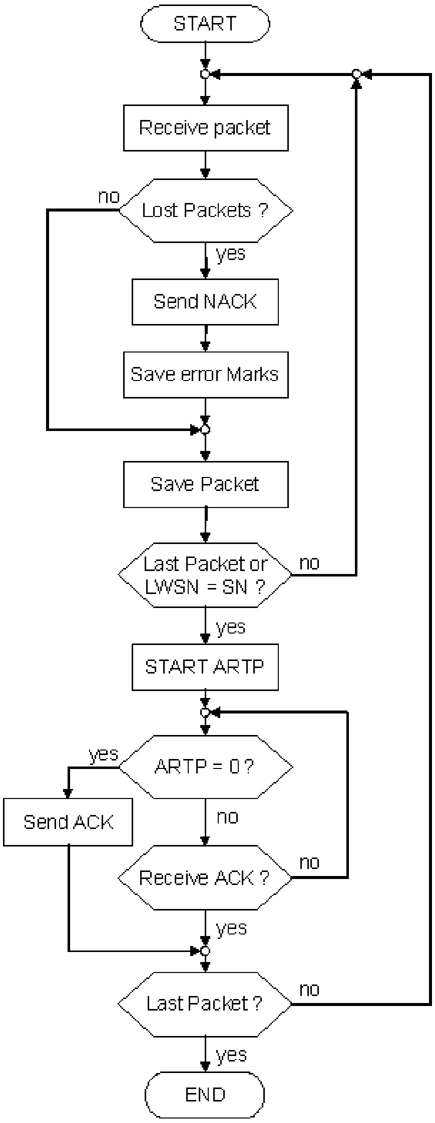

Figure 3. Flowchart of the multicast stage of the client.

The timeout timer management requires an special attention. Most existing network protocols are designed to send some information from a source to a receipt and then, wait for an answer during a defined lapse of time. Therefore, the protocol must set a timer and wait for any one of multiple events to occur: the timer has passed or the answer is received. To approach those timers, most developers use the well-known OS *select* function [16]. Although that function lets us specify a resolution in microseconds, it is recommended to use it to manage timer values of the order of hundredth of seconds. Therefore, this function is good enough to code protocols which timers do not require very small time values (for instance, TCP ACK waiting time).

Fortunately, there are other solutions to code protocols which require smaller timers. In [17], V. Oberle has developed a module which provides precise timers for the Linux 2.4 kernels and further. The module use the timer of the local APIC (Advanced Programmable Interrupt Controller) available on Pentium processors family since P6 chips. This module provides microsecond precision timers that are programmed with a TSC (Time Stamp Couter) value. In order to use the interrupts generated by the APIC, like the timer interrupt in the module, it is needed to patch the kernel to redirect the corresponding Interrupt Service Routine (ISR).

IV. Protocol characterization

The protocol behavior is strongly correlated with the flow control performance. In particular, the maximum window size, the steady state window size and the maximum throughput values are the three most important protocol parameters.

A. Maximum window size

The transmission rate is determined by the network capacity, the timeout timer and the window size. The proposed flow control algorithm modifies the last two parameters to reach an optimum transfer rate.

If there is no congestion, the server increases the window size up to its maximum value (supposing also an error-free transmission channel). To simplify, but without loss of generality, it is supposed that there is only one LAN with capacity C bps. Let us also suppose that the file size is large enough to assume that the transmission is performed by the maximum window size. Under these conditions, the total transfer time can be calculated as

$$T = \frac{FileS}{PayloadS} \cdot \frac{DataPS}{C} + \\ + \frac{FileS}{PayloadS \cdot W} \left(\overline{ARTP} + \frac{AckPS}{C} \right) \quad (10)$$

Where *FileS* is the file size, *PayloadS* is the data packet payload size, *DataPS* and *AckPS* are the data and ACK packet sizes respectively, $\overline{ARTP}$ is the ARTP mean value (that is, the time interval between the reception of a data window and the generation of the corresponding ACK packet), and *W* is the maximum window size.

The first addend is the time needed for the server to transfer the file and the second one is the average time required by the clients to issue the ACK packets. It is obvious that a high maximum window value enables a faster transmission rate, but at the same time the protocol has fewer opportunities to react to network congestion.

By simply operating in (10), the transfer time reduction due to the use of a window size W_2 instead of W_1 ($W_2 > W_1$) is equal to

$$\frac{FileS}{PayloadS} \left(\overline{ARTP} + \frac{AckPS}{C} \right) \cdot \frac{W_2 - W_1}{W_1 W_2} \quad (11)$$

If an appropriate window size W_1 is selected, an alternative window size W_2 (where $W_2 >> W_1$) does not provide a remarkable transfer time reduction since

$$\lim_{W2 \to \infty} \frac{W_2 - W_1}{W_2 W_1} = \frac{1}{W_1} \quad (12)$$

For example, the transfer time reduction when a maximum window size of 100 is used instead of a maximum window size of 1 is

$$\frac{FileS}{PayloadS} \left(\overline{ARTP} + \frac{AckPS}{C} \right) \cdot 0.99 \quad (13)$$

That is a notable transfer time reduction. However, the transfer time reduction when a maximum window size of 1000 is used instead of a maximum window size of 100 is only

$$\frac{FileS}{PayloadS} \left(\overline{ARTP} + \frac{AckPS}{C} \right) \cdot 0.099 \quad (14)$$

Therefore, there is an optimum maximum window size value from which the transfer time reduction is not fundamental, but it reduces the server opportunities for detecting and reacting to network congestion.

According to (11) and (12) we choose a maximum window size of 100 data packets (rule of thumb) since it achieves a fast data transmission rate, a quick response when congestion arises, and it avoids protocol starvation (that is, it enables to fairly share the network capacity with other flows).

B. Window Size Convergence

In a general scenario, the proposed flow control mechanism increments the transmission window size and modifies the timeout timer value during the multicast transmission until a steady state situation is reached. This steady state window size value, that is obtained according to certain network conditions, is strongly correlated with the throughput. In this section we derive a mathematical expression to this parameter.

In our analytical model, we must assume some simplifications to reduce the extremely complex general situation, which, however, does not invalidate the generality of our analysis. We assume that the intra-campus network consists of unequal capacity LAN networks (some of them working at C_1 and the others at C_2, where $C_1 >> C_2$) connected through multicast routers. We also assume that there are no other applications using the network and that the server is reasonably situated at one of the fastest LANs.

Congestion may arise in routers interconnecting LANs with different capacities. Those routers can be modeled as a pair of buffers serving packets at C_1 and C_2 Mbps respectively.

Supposing an initial window size of one (see figure 1), the server sends only one data packet to the network and it waits for ACK packets (one ACK packet from each interconnected LAN with clients). The last ACK packet received at the server is the ACK going through the path formed by the highest number of C_2 networks (it is composed of N_{C1} LANs at C_1 Mbps networks and by N_{C2} LANs at C_2 Mbps). When all ACK packets have arrived, the window size is increased by one unit and the next data window is issued. For a W window size, the server will receive the last ACK packet approximately at

$$\frac{W \cdot DataPS}{C_2} + (N_{C2} - 1)\frac{DataPS}{C_2} + N_{C1}\frac{DataPS}{C_1} +$$
$$\overline{ARTP} + N_{C1}\frac{AckPS}{C_1} + +N_{C2}\frac{AckPS}{C_2} \approx$$
$$\approx \frac{W \cdot DataPS}{C_2} + (N_{C2} - 1)\frac{DataPS}{C_2} + \overline{ARTP} \quad (15)$$

where LAN_2 to LAN_1 buffer delay can be neglected because the service rate at the other side is very high (C_1 Mbps).

The server will detect congestion when all the ACK packets do not arrive in time, that is, when the timeout timer expires before all the ACK packets arrives to it. Therefore, the window size just before congestion is detected (W_T) can be obtained when equation (15) slightly matches with *dflt_Tout*:

$$W_T = \left\lfloor \frac{\left(dflt\_T_{out} - \overline{ARTP}\right)}{DataPS} \cdot C_2 - (N_{C2} - 1) \right\rfloor \quad (16)$$

It can be noticed that for each C_2 network added to the critical path, the W_T value is decremented in one unit.

At this time, the server increases the window size again and it sends the next data block. Now, congestion is declared since the timer expires before the last ACK packet arrives. Therefore, the flow control multiplies the timer by α and decreases the window in one unit. In this new situation, it can be guaranteed that the server assumes the congestion has disappeared, since $\alpha > 1$. Once again, the window is increased and the timer is divided by β. But since $\beta > \alpha > 1$, the timer value reaches its default value again and then congestion comes back. This behavior is continuously repeated. Therefore, the window size reaches a steady-state value slightly oscillating around W_T.

C. Maximum throughput

SOMA transport protocol obtains the maximum throughput and the maximum window size (W_{max}) when it is the only running application using network resources and there is no congestion at any router. In that situation, the time interval between two consecutive data windows is restricted by the ARTP mean value (9) and not by the timer ($dflt\_Tout >> ARTP_{max}$). Therefore, in this case the maximum throughput is bounded by

$$\frac{W_{max} \cdot DataPS}{\frac{W_{max} \cdot DataPS}{C} + \overline{ARTP}} \quad (17)$$

Where C is the network capacity in bps at the server side.

However, if congestion arises at some network point, the timeout timer restricts the time between data blocks and the window size reaches its steady-state value. Therefore, the maximum throughput is bounded by

$$\frac{(W_T + 1) \cdot DataPS}{\frac{(W_T + 1) \cdot DataPS}{C} + dflt\_Tout} \quad (18)$$

V. Capturing and Processing SOMA Traffic

There are several solutions to capture real traffic in a network. The first one is to use a hardware protocol analyzer. They are able to capture frames and to get some traffic figures in real time. The capture can also be stored in a file to obtain more statistical parameters by subsequent processing. The equipment price and a very limited adaptation to new emerging network technologies are the main drawbacks of this solution.

Other solution is to use a software network sniffer running in a computer. Many companies (Cisco, HP, Nortel Networks, etc.) add this feature to their network management tools. However, the software cost and the limited traffic analysis characteristics are the main drawbacks in this case.

However, there is another solution: The freeware sniffers like *ethereal* [13] and *tcpdump* [14] (the first one is a GUI network sniffer and the second one is a line-oriented sniffer). This is the best solution in most of the situations. Their features are comparable to hardware solutions and they require a low investment. Furthermore, this option can adapt to any network technology since, nowadays, there are many network interfaces (Frame Relay, ISDN, ATM) available to PC at competitive prices. We have chosen *tcpdump* and *ethereal* sniffers to evaluate our proposed protocol.

Working with open-source sniffers allows us to easily improve the protocol analysis capabilities. The software can be modified to be able to identify a SOMA packet and to show the packet fields in an adequate format. *Ethereal* is a multiplatform software written in C language. Amongst other libraries, *ethereal* uses the packet capture and filtering library *libpcap* (Packet CAPture LIBrary), the graphical user interface *gtk+* and the *glib* library which allows the sniffer to generate and manage a protocol stack similar to the recommended OSI model. Adding a new protocol to the protocol supported set requires the codification of a new dissector (a C language file which name must be *packet_soma.c*; that is, the reserved word *packet_* followed by the new protocol acronym). The dissector encodes the new protocol definition, its header fields, its names and the format they must be shown in the result window. Moreover, the new dissector must be registered to the lower level dissector, and then *glib* library is able to insert SOMA protocol into the global protocol stack. In our case, SOMA packets are encapsulated into UDP packets, and therefore, the lower dissector is the specific UDP dissector.

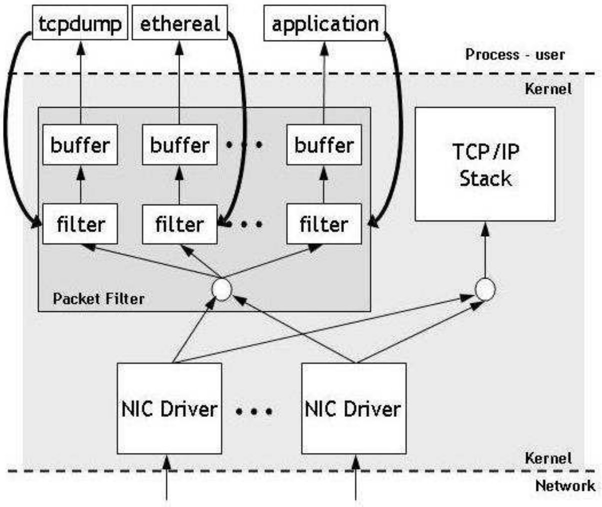

Figure 4. Operating System with and without Packet Filter. With a Packet Filter, each application establishes a kernel level filter. Each packet filter decides whether a packet is to be accepted and how many bytes of each packet should be saved. Without Packet Filter, all packets must go through the protocol stack.

Like *ethereal*, *tcpdump* is an open-source sniffer written in C that uses the *libpcap* library. However, the capture information is shown in console mode. This feature allows to capture traffic using computers without a graphical interface (i.e. X Server). This fact allows to save system resources and therefore to improve the traffic capture process.

Tcpdump adaptation implied different modifications. First, the predefined display format has been completely redefined. Original *tcpdump* displays the packet information in a tree structured format, where each protocol information is printed in a different line. Our customized *tcpdump* displays all the packet information in only one line (for post-processing purposes, as it is explained later). Second, as *ethereal*, *tcpdump* must be able to detect and interpret a SOMA packet. For that, the UDP protocol module (*print_udp.c*) has been modified to determine the port number associated to a SOMA packet. Then, if a UDP port number of a received packet matches the SOMA port number, *print_udp.c* calls the *print_soma()* function, coded in *print_soma.c* which analyzes and prints the SOMA header.

As well as adapting open-source sniffers to interpret our protocol, we use the Linux Packet Filter (LPF) inspired by the BSD Packet Filter (BPF) [12], a kernel architecture for packet capture. The BPF has two main components: The network tap and the packet filter. The network tap collects copies of packets from the network device drivers and delivers them to listening applications. The filter decides if a packet should be accepted and, if so, how much of it to copy to the listening application.

Figure 4 illustrates how the operating system manages the incoming packets in both cases: with and without a Packet Filter. When a packet arrives at a network interface, the link level device driver normally sends it up to the system protocol stack. But when BPF is listening on this interface, the driver first calls BPF. BPF feeds the packet to each participating process filter. For each filter that accepts the packet, BPF copies the requested amount of data to the buffer associated with that filter. The device driver then regains control.

The main advantage of the BSD Packet Filter is that it discards unwanted packets as early as possible and therefore it minimizes the packet copies across the kernel buffers.

Once traffic is captured and stored, information from packet header must be processed to extract the desired statistical figures. For that, we have use *awk*, a powerful pattern scanning and processing language. *Awk* scans input lines, line by line, to see if a line matches a set of patterns or conditions specified in a program. If a line matches a certain pattern, a specified action is carried out.

The high processing rate offered by the *awk* language has determined this election. *Awk* functions and programming philosophy are very similar to C language. Multiple arithmetic calculations can be programmed in an extremely easy way, and therefore many protocol figures and parameters can be obtained efficiently and quickly.

VI. TEST RESULTS DISCUSSION

In this section, we evaluate SOMA in a real situation. It should be noticed that our analytical study is focused on a transport layer but test experiments are obviously the result of all OSI layers integration, from the physical layer up to the transport one. Particularly, in section IV we have not taken into consideration the MAC, LLC, IP and UDP protocols and sub-layers. Moreover, SOMA runs over a multi-task OS, which has non real-time facilities (Linux kernel 2.4). Therefore, although we try to minimize the computational load in each computer (unnecessary processes, like *cron*, are killed), sometimes the kernel may give priority to other processes instead of SOMA. Both effects, the OSI layers integration and the multi-task OS may cause that the test results reveal some smaller differences with the analytical ones.

The intra-campus environment is formed by two LANs of extremely unequal capacities, a wired Ethernet LAN at 100 Mbps and a wireless LAN 802.11b at 2 Mbps, both connected through a wireless access-point router (see figure 5). The access-point router is a Linksys WRT54G, co-sponsored by Cisco Systems. We changed its firmware by a stable and configurable Linux OS called OpenWrt [15].

To verify that the analytical results obtained in section IV fit well enough with the test results, the same intra-

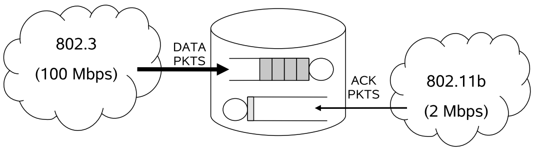

Figure 5. Router model. Delay from LAN_2 (802.3) to LAN_1 (802.11b) network is negligible

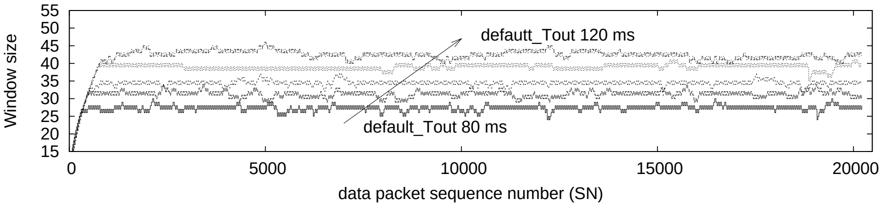

Figure 6. Window size evolution for different *dflt_Tout* values: 80, 90, 100, 110 and 120 ms

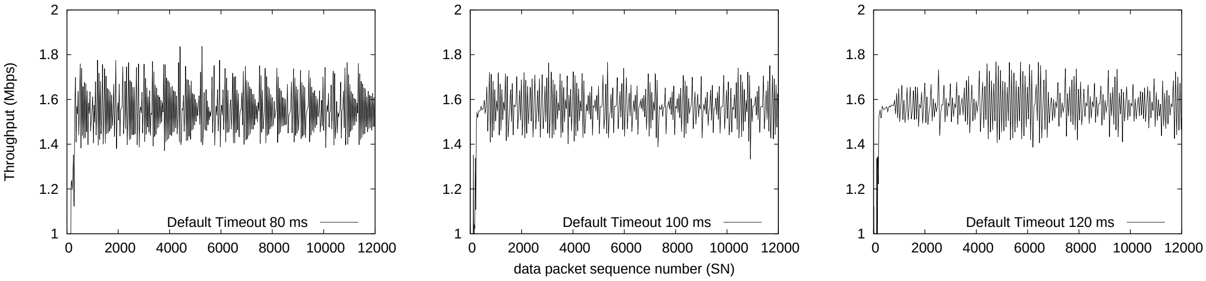

Figure 7. Instantaneous throughput evolution for different *dflt_Tout* values: 80, 100 and 120 ms

campus environment is used: the clients are situated in both LANs and the server is situated in the wired network.

Our test intra-campus network forces congestion since the wireless LAN capacity (2 Mbps) is fifty times lower than the wired network capacity (100 Mbps).

Figure 6 shows the evolution of the transmission window size when a 10 MBytes file is transfered to a group of clients (4 clients are attached to the wired network and 4 clients are attached to the wireless one) for different *dflt_Tout* values: 80, 90, 100, 110 and 120 ms. According to expression (16), the window size should oscillate around 29, 32, 36, 39 and 43 packets respectively. To obtain these values it is assumed that: (a) The $\overline{ARTP}$ is 120 μs, which is calculated using (9) when n=4 and the $ARTP_{max}$ is 600 μs. (b) The effective wireless LAN capacity at the transport layer is around 1.55 Mbps instead of the theoretical 2 Mbps due to the OSI layers integration.

As it can be observed, the analytical values fit well enough with the experimental ones and the window size always remains around its steady state value (W_T). Sometimes the window size slightly decreases due to sporadic packet losses at the wireless LAN side and also because of background control applications packets, such as BPDU spanning-tree, which overload the access point buffer capacity.

Additionally to the model validation using experimental results, we have validated our window size convergence study in more complex scenarios using the Opnet simulator. Each possible scenario is formed by several C_1 and C_2 networks so that the last ACK packet received at the server goes through a path formed by N_{C1} and N_{C2} networks. Table I presents the W_T value obtained by simulation and theoretically (16) when the value N_{C2} varies among 1 and 4.

It can be observed that simulated results validate the analytical study. In addition, the case $N_{C2} = 1$ (the scenario studied experimentally) fits good enough with the experimental results showed in figure 6.

TABLE I.
W_T VALUES OBTAINED THEORETICALLY AND BY SIMULATION (IN PARENTHESIS), SUPPOSING A WIRELESS LAN CAPACITY C_2 OF 1.55 MBPS AND $\overline{ARTP}$=120 μS

	dflt_Tout				
N_{C_2}	80 ms	90 ms	100 ms	110 ms	120 ms
1	29 (29)	33 (33)	37 (37)	40 (40)	44 (44)
2	28 (28)	32 (32)	36 (36)	39 (39)	43 (43)
3	27 (27)	31 (31)	35 (35)	38 (38)	42 (42)
4	26 (26)	30 (30)	34 (33)	37 (37)	41 (41)

Returning to test experiments, figure 7 represents the instantaneous throughput. Irrespective of the *dflt_Tout* value, the server throughput slightly oscillates around 1.55 Mbps. Therefore, the proposed flow control algorithm is able to adapt the server transmission rate to the slowest network capacity using a unique flow, maintaining synchronism among all clients and avoiding congestion.

This test result can be corroborated analytically by introducing the value of W_T (16) in (18) when $N_{C2} = 1$. Always assuming that mean ARTP value is negligible, the throughput can be approximated by

$$\frac{dflt\_Tout \cdot C_2 + DataPS}{\dfrac{dflt\_Tout \cdot C_2 + DataPS}{C_1} + dflt\_Tout} \approx C_2. \quad (19)$$

Where $C_2 << C_1$ and $DataPS << dflt\_Tout \cdot C_2$

In the next experiment, our protocol is evaluated in a single congestionless wired LAN. Figure 8 depicts the window size evolution and the instantaneous throughput. In this scenario the window size reaches its maximum value limited by the protocol (W=100) and the maximum experimental throughput is around 97 Mbps, which approximately matches the theoretical result (97.4 Mbps,

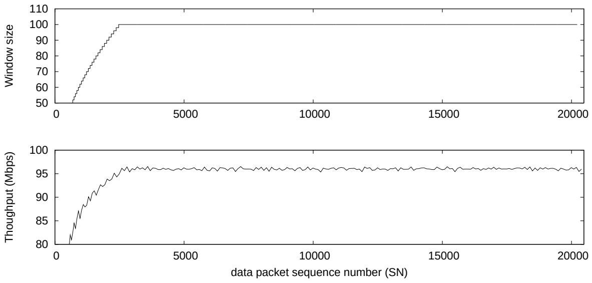

Figure 8. Window size and instantaneous throughput evolution in a congentionless 100 Mbps wired LAN

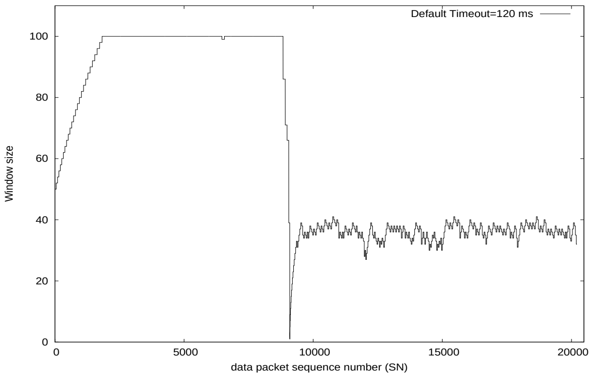

Figure 9. Window size evolution in a mixed wired and wireless intra-campus. The wireless LAN terminals join the file transfer approximately in the middle of the transfer

from equation 17). Again, the flow control is able to adapt the transmission to the maximum network capacity.

Finally, figure 9 illustrates the window size evolution in the same scenario but in a different experiment. At the beginning only wired clients participate in the file replication process. As it can be seen, the window size reaches its maximum value (W=100). But approximately in the middle of the transfer, the wireless terminals join the file transfer. As it can be appreciated, the SOMA flow control is able to quickly adapt to the new situation by resizing the window (and also the timer, although it is not shown) synchronizing both networks and avoiding congestion. If the router buffer is not high enough, some data packets could be lost during the transition period, which will be recovered in the error correction phase. To minimize this effect, the response time of our proposed protocol is an important factor since the wireless channel capacity is strongly dependent on physical parameters.

VII. Conclusions

SOMA is a multicast application for fast file replication in an intra-campus environment (several LANs interconnected through few routers). One of its most remarkable aspects is its own transport protocol definition focused mainly on flow control which is designed to work fine in an asymmetric intra-campus scenario, formed by LAN networks with different capacities (probably a mixture of wired and wireless LANs).

The proposed flow control algorithm is able to quickly react under congestion, resizing adequately the window size and the time between data blocks to maximize the throughput and to minimize the lost packet probability. Additionaly, the protocol also implements a mechanism to restore the lost data packets. The proposed error correction algorithm allows the free-error networks (networks whose clients have completely received the transfered file during the multicast phase) to finish the communication.

The main protocol parameters have also been characterized analytically. Some of them are the mean ARTP value, the maximum window size, the steady state window size and the maximum throughput. In addition, the mathematical study has been validated with real traces in a intra-campus environment composed of a wired Fast Ethernet network and a wireless 802.11b Ethernet connected through an access point router. It has been neccesary to adapt open-source sniffer tools (tcpdump and ethereal) to capture the real SOMA traffic traces. To be able to extract the protocol performances from the traces, we have developed some scripting tools as well.

Although the proposed transport protocol is used in SOMA for file transfer, its synchronicity and simplicity makes it interesting for other type of applications, like on-line applications. The metodology employed to define, analyze, and evaluate this multicast protocol is, indeed, another contribution of this work and it can be easily extended to other multicast protocols.

Acknowledgments

This work has been supported by the Spanish Research Council under project ARPaq (TEC2004-05622-C04-02/TCM).

References

[1] Schulzrinne, H. et al.: RTP. A Transport Protocol for Real-Time Applications. RFC 3550, Internet Engineering Task Force, July 2003.

[2] Floyd, S., Jacobson, V., Liu, C., McCanne, S., Zhang, L.: A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. IEEE/ACM Transactions on Networking, Vol. 5, No. 6, pp. 784-803, December 1997.

[3] Sabata, B., Brown, M. J., Denny, B. A., Heo, C.: Transport protocol for reliable multicast: TRM. In Proc. of IASTED International Conference on Networks, pp. 143-145, January 1996, Orlando, Florida.

[4] Lind, K. et al.: Drinking from the Firehose: Multicast USENET News. In Proc. of the Winter 1994 USENIX Conference, pp. 33-45, 1994, San Francisco, CA.

[5] Macker, J.: The Multicast Dissemination Protocol (MDP) Toolkit. In Proc. of IEEE MILCOM, Vol. 1, pp. 626-630, 1999.

[6] Miller, K. et al.: StarBurst Multicast File Transfer Protocol (MFTP) Specification. IETF Internet Draft, draft-miller-mftp-spec-03.txt, April 1998.

[7] Lin, J.C., Paul, S.: RMTP. A Reliable Multicast Transport Protocol, In Proc. of Infocom96, pp. 1414-1424, March 1996, San Francisco, CA.

[8] Yavatkar, R. et al.: A reliable dissemination protocol for interactive collaborative applications. In Proc. of the ACM Multimedia'95, pp. 333-344, 1995.

[9] http://www.ietf.org/html.charters/rmt-charter.html

[10] Kermode, R., Vicisano, L.: Author Guidelines for RMT Building Blocks and Protocol Instantiation documents. IETF Internet Draft, draft-ietf-rmt-author-guidelines-03.txt, January 2002.

[11] Manzanares-Lopez P., Sanchez-Aarnoutse J.C., Malgosa-Sanahuja J., Garcia-Haro J.: Empirical and Analytical Study of a Multicast Synchronous Transport Protocol for Intra-Campus Replications Services. In Proc. of the International Conference on Communications (ICC'04), June 2004, Paris, France.

[12] McCanne S., Jacobson V.: The BSD Packet Filter: A New Architecture for User-level Packet Capture. In the 1993 Winter USENIX conference, San Diego, CA, (January 1993), pp. 259-269.

[13] http://www.ethereal.com

[14] http://www.tcpdump.org

[15] http://openwrt.org

[16] Stevens,W. Richard: UNIX Networking Programming. Networking APIs: Socket and XTI, Vol. 1, Second Edition, pp. 151, ed. Prentice Hall PRT, 1997, ISBN 0-13490012-X.

[17] APIC timer Module for Linux. http://www.oberle.org

Pilar Manzanares-Lopez received the Engineering degree in Telecommunications in 2001 from the Technical University of Valencia (UPV), Spain. In April 2006, she received the Ph.D. degree in Telecommunication from the Polytechnic University of Cartagena (UPCT), Spain. She is an Assistant Professor at the Department of Information Technologies and Communications (Polytechnic University of Cartagena) since 2001. She has been involved in several National research projects related to multicast technology and multimedia facilities. She is the co-author of several papers in the fields of transport protocols and distributed systems.

Juan Carlos Sanchez-Aarnoutse received the Automation and Electronic Engineering degree in 2001 from the Polytechnic University of Cartagena (UPCT), Spain. In May 2006, he received the Ph.D. degree in Telecommunication from the UPCT, Spain. He has been an Assistant Professor at the Department of Information Technologies and Communications (Polytechnic University of Cartagena) since 2002. He has been involved in several National research projects related to multicast technologies and communication protocols. He is author of several papers in the fields of multicast and broadcasting protocols, as well as p2p traffic measurement and control.

Josemara Malgosa-Sanahuja received the Telecommunication Engineering degree in Telecommunications in 1994 from the Polytechnic University of Catalonia (UPC), Spain. In November 2000, he received the Ph.D. degree in Telecommunication from the University of Zaragoza (UZ), Spain. He has been an Assistant Professor at the Department of Electronic and Communications Engineering (University of Zaragoza) since 1995. In September 1999, he joined the Polytechnic University of Cartagena (UPCT), Spain, as Associated Professor. He has been involved in several National and International research projects related to switching, multicast switching technologies, traffic engineering and Multimedia value-added services design. He is author of several papers in the fields of switching, multicast technologies and distributed systems. Since 2000 he is in charge of the development of the new Information and Communications Technologies for the Polytechnic University of Cartagena. He is regional correspondent of the Global Communications included in the IEEE Communications Magazine since 2002.

Joan Garcia-Haro received the Telecommunication Engineering degree and the Ph.D. in Telecommunications in 1989 and 1995 respectively, both from the Polytechnic University of Catalonia (UPC), Spain. He has been an Assistant Professor at the Department of Applied Mathematics and Telematics (DMAT-UPC) since 1992, and Associate Professor since 1997. In September, 1999 he joined the Polytechnic University of Cartagena (UPCT), Spain, where he is Professor of the Department of Information Technologies and Communications. He is author or co-author of more than 50 papers mainly in the fields of switching and performance evaluation. From April 2002 to December 2004 he served as EIC of the IEEE Global Communications Newsletter, included in the IEEE Communications Magazine. He is Technical Editor of the same magazine from March 2001. He also holds an Honorable Mention for the IEEE Communications Society Best Tutorial paper Award (1995).