# Analysis and Performance Evaluation of a Multicast File Transfer Solution for Congested Asymmetric Networks

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
{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. For efficient bandwidth utilization, SOMA uses IP multicasting. 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 intensively tested by simulation and experimentally in a lab, using a mixture of wired and wireless intra-campus networks. In addition, we develop a mathematical model to validate analytically some of 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.

## 1 Introduction and related work

The use of multicasting within a network has many strengths. Multicast minimizes the link bandwidth consumption because no multiple unicast connections are needed to send the information. In addition, it also reduces the sender and router processing and the delivery delay. On the other hand, IP multicasting may be used to add anonymity to a communication, because there is not a univocal relationship between an IP multicast address and a host.

In this paper we propose, analyze, implement and test a SynchrOnous Multicast Application called SOMA to synchronously transfer a large amount of data from a server to a group of clients. It is specially featured to operate in an intra-campus environment (several interconnected LANs through few routers).

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.

The extreme complexity associated to the definition of a 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 an 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.

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.

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 standard Linux kernel routines.

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

## 2 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.

### 2.1 Overall protocol description

SOMA splits the transmission process into **two phases**. In the **first one**, the server multicasts a set of data packets (a transmission window) to all clients. The clients store the payload and contend to confirm the received packets by an ACK. 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. The feedback information (ACK and NACK packets) received at the server are used to resize the transmission window. The above procedure is repeated until the file is completely transferred.

The **second 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 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 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.

### 2.2 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, an 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 payload, which transports 512 information bytes.

### 2.3 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 $\alpha$ ($\alpha > 1$). But if the window is confirmed in time, the timer value is decreased as denoted by expression (1)

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

Where $\beta > \alpha > 1$ and *default_Tout* is the bottom threshold value. The server repeats this 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 SN and compares it with the expected value:

– If SN is the expected one, the client stores the payload and updates the sequence number.
– If SN 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 SN 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).

### 2.4 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 $n$ is the number of clients. Therefore, the mean ARTP value is [11]

$$\overline{ARTP} = \left(1 - \frac{n}{n+1}\right) \cdot ARTP_{max} \qquad (2)$$

It 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 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 ($\overline{ARTP}$).

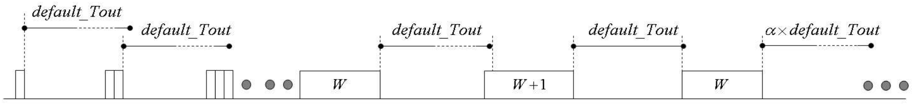

**Fig. 1.** Window size evolution in an asymmetric network environment

## 3 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.

### 3.1 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

$$T = \frac{FileS}{PayloadS} \cdot \frac{DataPS}{C} + \frac{FileS}{PayloadS \cdot W} \left( \overline{ARTP} + \frac{AckPS}{C} \right) \tag{3}$$

Where $FileS$ is the file size, $PayloadS$ is the data packet payload size, $DataPS$ and $AckPS$ are the data and ACK packet sizes respectively, 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 ACKs. 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 (3), 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

$$\frac{FileS}{PayloadS}\left(\overline{ARTP} + \frac{AckPS}{C}\right) \cdot \frac{W_2 - W_1}{W_1 W_2} \tag{4}$$

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

$$\lim_{W2 \to \infty} \frac{W_2 - W_1}{W_2 W_1} = \frac{1}{W_1} \tag{5}$$

According to (4) and (5) 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).

### 3.2 Window Size Convergence

The window size during the transmission reaches a steady state value, which 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 ACKs (one from each LAN). The last ACK received at the server is the ACK going through the path formed by the highest number of $C_2$ networks (it is composed by $N_{C1}$ LANs at $C_1$ Mbps networks and by $N_{C2}$ LANs at $C_2$ Mbps). When all ACKs 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 approximately at

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

$$+ N_{C2}\frac{AckPS}{C_2} \approx \frac{W \cdot DataPS}{C_2} + (N_{C2} - 1)\frac{DataPS}{C_2} + \overline{ARTP} \tag{6}$$

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 window size just before congestion is detected ($W_T$) can be obtained when (6) slightly matches with *default_Tout*:

$$W_T = \left\lfloor \frac{\left(default\_Tout - \overline{ARTP}\right)}{DataPS} \cdot C_2 - (N_{C2} - 1) \right\rfloor \tag{7}$$

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 $\alpha$ 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 $\beta$. 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 $W_T$.

### 3.3 Maximum throughput

SOMA obtains the maximum throughput and the maximum window size ($W_{max}$) when it is the only running application and there is no congestion. In that situation, the time interval between two consecutive data windows is restricted by the ARTP mean value (2) and not by the timer ($default\_Tout >> ARTP_{max}$). Therefore, in this case the maximum throughput is bounded by

$$\frac{W_{max} \cdot DataPS}{\frac{W_{max} \cdot DataPS}{C} + \overline{ARTP}} \tag{8}$$

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

$$\frac{(W_T + 1) \cdot DataPS}{\frac{(W_T + 1) \cdot DataPS}{C} + default\_Tout} \tag{9}$$

## 4 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 3 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. 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 [12].

To verify that the analytical results obtained in section 3 fit well enough with the test results, the same intra-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).

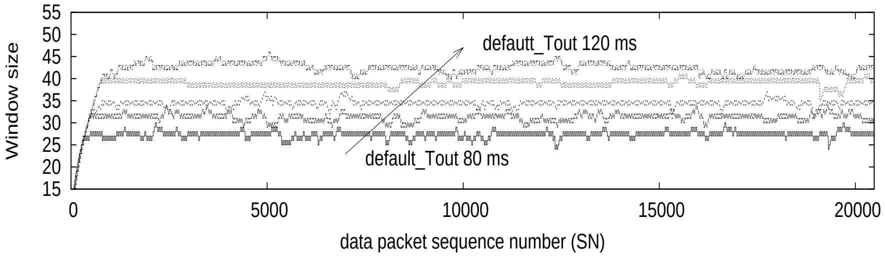

**Fig. 2.** Window size evolution for different *default_Tout* values: 80, 90, 100, 110 and 120 ms

Figure 2 shows the evolution of the window size for different *default_Tout* values: 80, 90, 100, 110 and 120 ms. According to expression (7), 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 $\mu$s, which is calculated using (2) when n=4 and the $ARTP_{max}$ is 600 $\mu$s. (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.

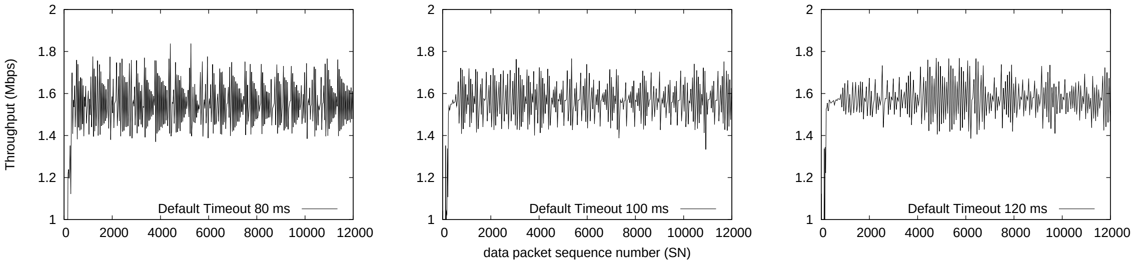

**Fig. 3.** Instantaneous throughput evolution for different *default_Tout* values: 80, 100 and 120 ms

Additionally, 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 1 presents the $W_T$ value obtained by simulation and theoretically (7) when the value $N_{C2}$ varies among 1 and 4.

**Table 1.** $W_T$ values obtained theoretically and by simulation (in parenthesis), supposing a wireless LAN capacity $C_2$ of 1.55 Mbps and $\overline{ARTP}$=120 $\mu$s

| | *default_Tout* | | | | |
|---|---|---|---|---|---|
| $N_{C2}$ | 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) |

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 2.

Returning to test experiments, figure 3 represents the instantaneous throughput. Irrespective of the *default_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 $W_T$ (7) in (9) when $N_{C2} = 1$. Always assuming that mean ARTP value is negligible, the throughput can be approximated by

$$\frac{default\_Tout \cdot C_2 + DataPS}{\dfrac{default\_Tout \cdot C_2 + DataPS}{C_1} + default\_Tout} \approx C_2 . \qquad (10)$$

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

In the next experiment, our protocol is evaluated in a single congestionless wired LAN. 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, from equation 8). Again, the flow control is able to adapt the transmission to the maximum network capacity.

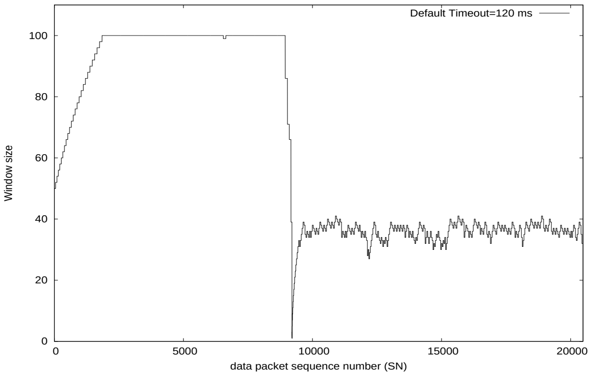

**Fig. 4.** Window size evolution in a mixed wired and wireless intra-campus. The wireless LAN terminals join the file transfer approximately in the middle of the transfer

Finally, figure 4 illustrates the window size evolution 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.

## 5 Conclusions

SOMA is a multicast application for fast file replication. 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. 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.

Some of the main protocol parameters have also been characterized analytically under certain constrains. In addition, the mathematical study has been validated with real traces in a test lab network.

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.

## 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. http://openwrt.org