A Reliable multicast protocol for a white board application

Walid Dabbous, Blaise Kiss

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

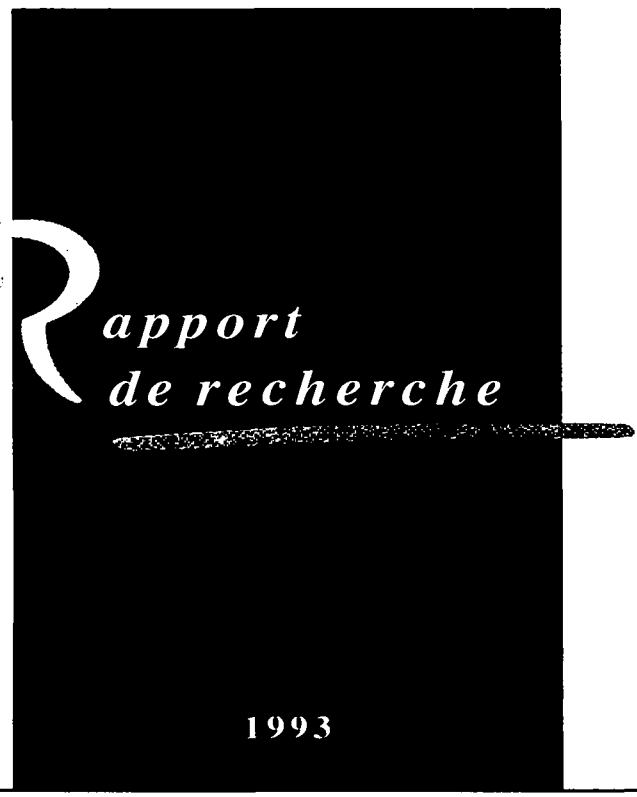
A Reliable Multicast Protocol for a White Board Application

Walid DABBOUS
Blaise KISS

N° 2100
Novembre 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués



A reliable multicast protocol for a white board application

Walid Dabbous
Blaise Kiss

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet RODEO

Rapport de recherche n°2100 — Novembre 1993 — 14 pages

Abstract: This paper describes a proposed mechanism for a reliable multicast protocol working on top of unreliable network service and making efficient use of the underlying multicast facilities available on the Internet. The primary goal is to provide a reliable multicast transport for a multi-user shared workspace application which is commonly used with other interactive video or audio conferencing tools.

Key-words: Reliable multicast, Shared white board.

(Résumé : tsvp)

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Un protocole de diffusion fiable pour un tableau blanc partagé

Résumé : Dans cet article nous proposons un protocole de transmission multipoint fiable. Ce protocole tourne au dessus d'un service réseau sans connexion et tire parti du support de la transmission multipoint disponible actuellement sur le réseau Internet. Ce protocole fournit un support de transmision fiable pour une application de tableau blanc partagé utilisée en général en parallèle avec d'autres outils de vidéo ou d'audio conférence.

Mots-clé : Diffusion fiable, Tableau blanc partagé.

1 Introduction

A reliable multicast protocol maintains uniform semantics for multicast message delivery even in the presence of failures. Usually, such a protocol ensures the delivery of a multicast message to either all or none of the destinations. In addition, many protocols guarantee that multicast messages adressed to a group are received in the same order by all the members of the group.

A number of reliable multicast protocols have been proposed [1], [2], [3], [4], [5], [6]. Each of these protocols ensures reliable transfer of messages with different types of message ordering. On the other hand, an unreliable IP multicast service [7] is now supported on a wide part of the internet - the MBONE or Multicast backBONE. However, most the reliable multicast protocols cited above have been developed without assuming the availability of such network level multicast. The ISIS protocol ([3],[8]) performs multicasts using s transport protocol, such as TCP, to establish multiple reliable point to point connections. This protocol cannot utilize an unreliable internet facility. Other protocols like the Central Receiver Protocol (CRP) [1] and the Propagation Scheme (PS) [2] use negative acknowledgment schemes and could perhaps utilize an unreliable multicast facility more efficiently than ISIS. These protocols require, however, point-to-point communication when network faults occur [9]. Other protocols like [6] use a token scheme to provide reliability.

In this paper, we propose a reliable multicast protocol called RMP which makes efficient use of the underlying multicast facilities. This protocol doesn't use a token scheme but is based on a more robust virtual distributed queue mechansim. The primary goal is to provide a reliable transport service to be used by multi-participant applications. The protocol is designed to run on top of UDP transport protocol. Although the lower UDP layer is unreliable, the protocol guarantees a reliable delivery of all messages. In general, data is delivered to the application in sequence. However, if a packet has not been received before a given wait interval, or if a packet is delayed in the network, out of order packets may be delivered to the application first and the delayed packet will be delivered on arrival. The protocol will be able to notify the upper layer on a late arrival to be inserted in the existing list of messages.

Even if the main purpose of this protocol is to use the UDP multicast facilities for multi-participant multi-media conferences, it can be used for a point to point communication as well. A special interest, however, is given to the use of this protocol by a shared workspace drawing application. This protocol was implemented for a white board application and tested in a local area environment.

The packet formats are compatible with the real-time transport protocol (RTP) [10]. The use of a single RTP header for both real-time and reliable data transfers facilitates the synchronization of the different data flows.

In the next section, we will describe the part dealing with the reliable transfer. Section 3 will detail the ordering mechanisms. Section 4 presents the packet formats, section 5 presents future work and section 6 concludes the paper.

2 Reliability

The reliabilility is ensured by using a negative acknowledgments (Nack) scheme. Messages can either be multicasted to the IP multicast group or sent as a point to point datagram. In both cases, the network service is the UDP unreliable service.

2.1 Local sequence numbers

Each station uses a local sequence number (S_n) which is incremented only if the station emits new data (e.g. a drawing data or a refresh data in the case of a drawing application). The station can emit other packet types as well. Each station will keep a list in memory of all other active stations (i.e. those from which the station received at least one data message) and of their last known sequence numbers. Each element of the active stations list will contain the following fields:

- a *station identifier* provided by the lower UDP layer at the reception of a message (e.g. IP address).
- a *sequence number* equal to the last valid sequence number received from that source. A sequence number is considered valid if its value is equal to the previous valid sequence number incremented by one.
- a flag indicating whether there are more elements in the list.
- a flag indicating whether the receiving station has confirmed the message with the current *sequence number* or not.
- a *depth* parameter indicating the estimated position of the current station in the queue of active stations (see below for further description of this variable).

2.2 Sending Messages

Data and control messages are encapsulated with RTP data units (RPDU). The value of the "content" field should indicate "Reliable Multicast". The fixed RTP header is followed by a number of options corresponding to the reliable multicast protocol (RMP) and by the application data (application specific options).

The first byte of the option indicates, among others, the option type, and therefore the option size for fixed length options. Otherwise, for variable length

options, the size is included in the option header as described in 4.2.5 and 4.2.6. The first bit of each option header, i.e. the 'final' flag (F), is set to zero if there are more options following, 1 otherwise. There is a bit in the RTP header indicating the presence of options.

The header formats are described below, and each emitter station is to fill out the proper fields correctly before emitting a message.

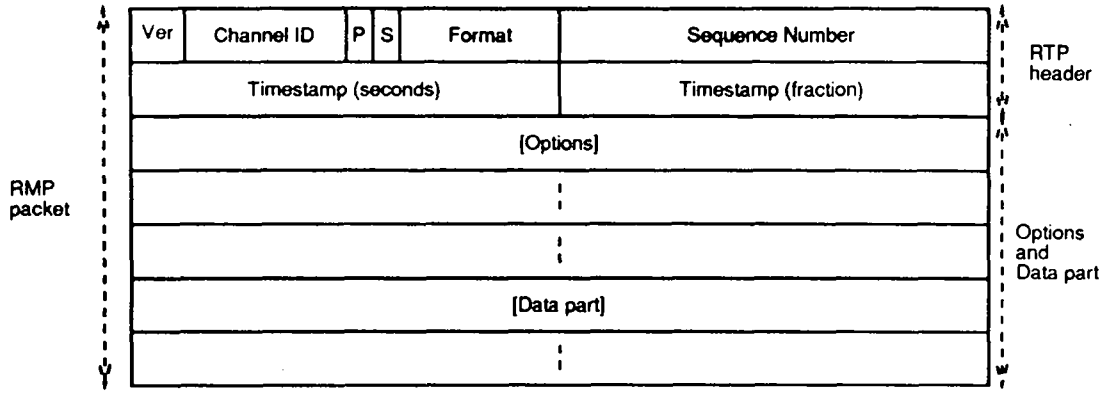


Figure 1: RMP Packet format

Basically, there are two kinds of options:

- Control options transmitted by the protocol layer for different purposes. The packets containing only control messages are not stored in memory for later retransmissions. Unless otherwise specified, a control option may be present several times in the same packet with different parameters. RTP packets containing only RMP control options are called control packets.

- A data part, transmitted by the upper layer. This could be a new data, a command to be executed on the existing data (e.g. modify the parameters of an existing object) or a refresh of the existing data. If the RTP packet contains a data part, the sequence number present in the RTP header has to be incremented for reliability purposes. These packets are "stored" in memory at reception time, for possible retransmission. RTP packets containing application data are called data packets. Note that control options can also be present in such packets. In a data packet, there can be only one data part. The data is placed after the last "option" indicating the presence of data. The length of the data part is calculated from the current position and the length of the RMP packet (given by the lower layer).

2.3 Confirmations

When a station receives a packet, it checks whether the emitter station is in its list of active stations or not. If it is not, the receiver will add the station in its list of active stations (with the starting sequence number of 0).

Then, the receiver will check the sequence number and the content of the message. The sequence number has to be equal or one more than the last sequence number received from that same source. (This test is not performed for retransmitted packets.)

If the received packet contains a data part and the sequence number is valid, then this sequence number is updated in the list of active stations, the A flag is set to zero, the 'depth' of the corresponding station is set to zero, and the data message is delivered to the application. If the sequence number is not valid then the packet is put in a waiting list, and a negative acknowledgement (Nack) is sent to the message originator. The receiver station will check every second if this Nack has been answered, and if not, it will re-send the Nack, but not more often. If a time period of WI seconds (Wait Interval, say 10 seconds) elapsed before the reception of the answer to the Nack, the corresponding packet is delivered to the application. If the arriving message filled the gap between the last valid sequence number and the waiting message, all in sequence packets are processed and the variables (A, depth, S_n) of the station in the memory list are updated.

Every time a station sends out a packet, it includes all the last received sequence numbers that it did not acknowledge yet, confirming this way the reception of all the messages up to a certain sequence number. Confirmations of this type constitute the piggy-backing of the acknowledgments. When a data packet containing a confirmation option is multicasted, the A flags of the confirmed sequence numbers is set to 1. From that time, this sequence number will not have to be confirmed because such a packet must be received once by every other station (it will be stored and retransmitted if necessary).

Confirmations may be included in all packets, but they should be in a data packet at least. Including confirmations in control packets may lead to faster detection of errors, but it should not influence the corresponding A flags on the emitter. This one will still have to include the confirmations until it sends a data packet.

2.4 Last packet problem

Data packets have to be confirmed. If they are not confirmed, this means that either no other station received them or that all other stations are quiet i.e. have nothing to transmit (confirmations are piggy-backed in data packets). In both cases the emitter must retransmit its last message periodically, to remind the others his last sequence number that needs to be confirmed. If any station did not get the last message, it will see the "reminder", and finally confirm it.

The "reminder" packet contains a remind number which is the number of times this message has been re-sent.

The necessity to include the remind number is explained in the following section (2.5).

The period of the reminder packets should be a little more than the round trip time in bad conditions on the network, typically 0.5 to one second. Since reminder packets are transmitted only once a second when there are no other packets, they will not overload the network significantly.

The remind option must be present before all other options in the packet because the following parts are no treated the same if the remind number is different from the depth of the receiver:

- Nacks are not treated
- automatic retransmissions are not done based on faulty confirmations

2.5 Depth value

Each station has its own depth value and can be estimated by all other stations.

Whenever a station joins the group, the value of its depth is set to the number of active stations incremented by one. When this station multicasts a data message, its depth value is set to zero. The depth value is incremented by one every time the station receives a multicasted message from an other station which had a higher depth value than the receivers'.

This method helps to put all active stations in a virtual queue where whoever emits goes to the front with a depth of zero.

When a "reminder" packet is sent, it will contain all necessary confirmations, and the remind number. Any other station can detect whether an emitting station missed a packet by two means:

- If the emitter confirmed an old valid S_n,
- If the emitter keeps sending its reminder packets periodically whereas a confirmation of this packet has been received. If the emitter did not get a packet, and the receiver detects this, it will retransmit the missing message to the emitter if its own depth value is less or equal to the remind number.

2.6 Negative acknowledgements

Each station can detect that it did not receive a message in two ways:

- First, the same emitter can send a second message, and the receiver will see a gap in the sequence numbers.
- Second, a third station can emit a packet containing a confirmation for a packet that the receiver did not get.

In both cases, the receiver station can extract the address of the original sender from the received packet, and multicast a Nack asking for the message with the missing S_n from that a source. The Nack, if not answered, will be retransmitted after a short period of time (say one second, same time delay as for the reminder, but not more often, to let enough time to the retransmission to arrive). Nacks contain the number of times they have been multicasted (Nack number). The station whose depth value is equal or less than this number must answer the request and retransmit the packet which was asked for. This is a practical way of making sure that not all stations answer a multicasted Nack, and also that some station will finally answer, if the first station on the virtual queue left the group.

2.7 Retransmissions

Retransmissions are generally done point to point. They can also be multicasted if there are many stations requesting the same message. Note that the way retransmissions are done has no effect on the reliability, it's only a question of compromise between redundancy and the network load.

Retransmissions do not need to be confirmed, because if they are not received, they will be requested again.

The retransmitted packets contain one single option from which the receiver station can extract the original data packet. Once this extraction is done, the receiver simulates a late reception, checking all the contained confirmations and other options. There is only one difference with a normal reception: the receiver is not allowed in this case to do automatic retransmissions, because the confirmations contained in the retransmitted packet (i.e. the state of the original senders) may not be the same any more. However if there are any messages confirmed that the receiver does not have, it can ask for them.

The amount of packets stored by the protocol layer for later retransmissions can be based upon a time limit (e.g. packets older than 5 minutes are thrown out) or upon a memory limitation (e.g. if there are more than 200 packets the oldest ones are removed). The data for the refresh has to be stored by the application. If the "refresh" feature is implemented in the application, the data is always stored.

2.8 Joining a group

When joining a group, a station has to ask for the current data state (for example a screen refresh) and also for the current sequence numbers of all active stations. The joining station is not allowed to undertake any other actions until it completed properly the join procedures. To get all this information, the joining station sends a join message to the group. This message can be sent only a finite number of times (typically 5), after which, if there is no answer, the joining station can consider itself to be alone in the group.

The join message contains the number of times it has been sent (the join number), and the station with the depth value equal or less than this number must answer by sending the last sequence numbers of all stations it has in its list, including his own. The joining station may see his own address in the list of the station, meaning that it had a previous session. In this case the joining station must update his own sequence number, as shown in the list.

The answering station has to ask the upper layer to send the current data state. The messages containing this data refresh must contain sequence numbers so that if they are lost, they can be asked for. The data is then multicasted on the network. The join procedure is considered successfully terminated if the whole image is received i.e. with the reception of a packet with the S bit of the RTP header set to one. (see 4.2.5) Note that it could also be sent point to point, but in that case the protocol would have to update the (hidden) sequence numbers at every other station in order to avoid false gap detection by a third party station for subsequent data packet exchange. (As other station will not receive the refresh packet with the incremented sequence number if the retransmission was done point to point.)

It is assumed that the local clocks at all stations are running close (e.g. by the use of a synchronization method such as NTP [11], [12]). However, the answers to the join message will contain a timestamp which may help to reduce the clock offset between the joining station and the rest of the group. The joining station can compute the difference between its own clock and the timestamps in the packet headers. The difference with the average group time will be at most the transmission time, which is generally under a second. So the joining station can approximate the general time being the minimum of all timestamps received from the answering stations. If this difference is lower than 0.5 seconds, the time offset can be considered as null.

It is the responsibility of the answering station to tell the joining station when the whole image has been transferred by setting the S bit to one in the last image message.

3 Ordering

The ordering of data messages is done according to the timestamps contained in the RTP header. If these are equal, it's done according to the address of the originator. For this reason, in retransmitted messages, the address of the original sender has to be included.

It is quite possible that a message with a earlier timestamp arrives later than an other message with a later timestamp. In this case the new message will have to be inserted in the existing ordered list, and the upper layer will have to take the appropriate actions to do the same. (For example, redraw all the data with a later timestamp than the last one to arrive.)

When a gap is detected in the received sequence numbers, the arriving packet is put on hold in a waiting list, and a Nack is sent to get the missing packet. However, if the request has not been answered before a given wait interval (e.g. 10 seconds), the first packet may be delivered to the application out of sequence. The delayed packet will be delivered on arrival, and the protocol will notify the upper layer on a late arrival to be inserted in the existing list of messages.

4 Packet formats

The packet formats are described below. They all have the same fixed size RTP header, followed by a certain number of options.

4.1 RTP header format

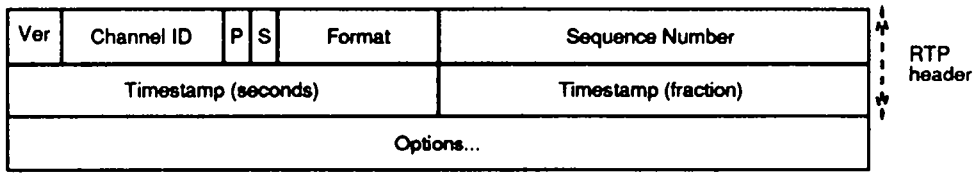


Figure 2: RTP header

The *ChannelID* field serves for multiplexing/de-multiplexing at the RTP level (several audio or video channels within the same conference). The *P* bit indicates whether or not the RTP fixed header is followed by on or more options. The *S* bit can be used as an indicator for the end or the start of a synchronization unit. The *format* field indicates the format of the RTP payload (e.g. RMP packets). The *sequence number* field is used to restore packet sequence and identify packets to the application. reflects the wall clock time when the RTP packet was generated. It wraps around approximately every 18 hours.

4.2 Options

The packet header is followed by options, if any, and the media data. Optional fields are summarized below. Unless otherwise noted, each option may appear several times per packet. Each packet may contain any number of options.

The option header always starts with the (*F*) flag which is set to zero if this is the last option in the message. The following 7 bits indicate the type of the option. The RMP protocol will use the following options types: *Join*, *Nack*, *Confirm*, *Reminder*, *Image*, *Data* and *Retransmit*.

Figure 3: The Join option

4.2.1 Join

Final-flag (F=0): 1 bit
This value is set to one if this is the last option in the message, zero otherwise. This must be a unique option, so the value of F must be zero. A station can not emit any kind of data, if it did not complete properly the join procedure.

Join: 7 bits
The option type is join (66).

Join number: 8 bits
The number of times this join message has been sent.

4.2.2 Nack

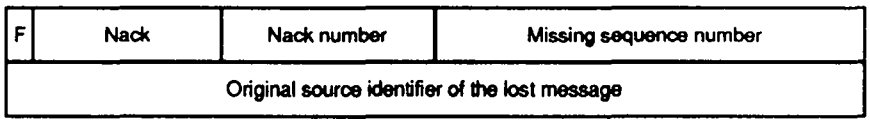


Figure 4: The Nack option

Nack: 7 bits
The option type is a Nack (67).

Nack number: 8 bits
The number of times this Nack has been sent.

Missing sequence number: 16 bits
The sequence number of the message that has not been received.

Source identifier: 32 bits
The identifier of the original source of the message.

4.2.3 Confirm

Confirm: 7 bits
The message type is confirm (68)

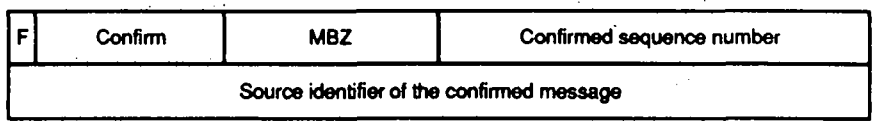


Figure 5: The Confirm option

Confirmed sequence number: 16 bits
This field contains the sequence number that is confirmed.

Source identifier: 32 bits
The identifier of the original source of the message.

4.2.4 Remind

Figure 6: The Remind option

This option is inserted in front of all other options if the message is the last message of a station and is periodically repeated.

Remind: 7 bits
The message type is remind (69)

Remind number: 8 bits
This field contains the number of times this message has been repeated (modulo 256).

4.2.5 Image

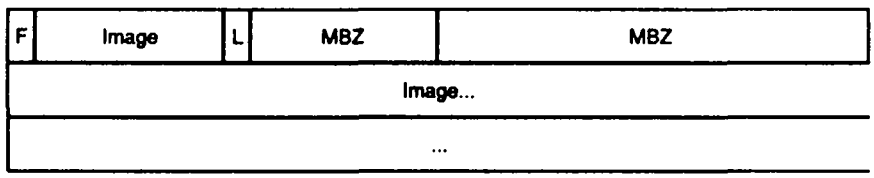


Figure 7: The Image option

Image: 7 bits
The message type is image(70)

Last: 1 bit
This bit is set to one if this image part is the last one to complete the image transfer.

An image message is the refresh of the current data state. Messages of this type are requested when joining the group, or at any other time for a refresh.

4.2.6 Data

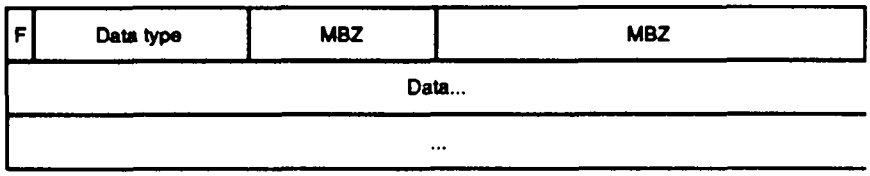


Figure 8: The Data option

Data: 7 bits
The message type is data(71)

A data part may contain new data to be added to the already existing, or a command to be executed on the existing data.

4.2.7 Retransmit

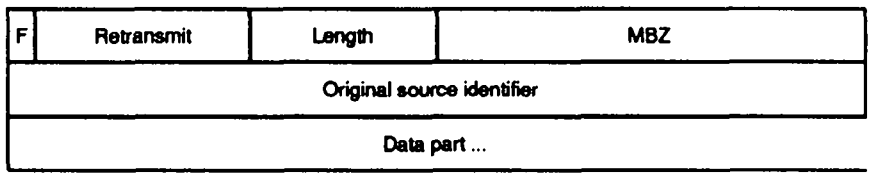


Figure 9: The Retransmit option

The original packet is placed in the data part of this option. The retransmissions are done point to point. The insertion of control options in the same packet is optional.

Final-flag (F): 1 bit
This value is set to one because this is the unique option of the message. The options contained in the original message are stored in the data part.

Retransmit: 7 bits
The message type is retransmit(72)

Length: 15 bits
This field indicates the length of the data part.

Original Source identifier: 32 bits
The identifier of the original source of the message.

Data part:
Contains the original RMP packet.

5 Adaptation to network conditions

The protocol has been implemented for a white board application and tested in a local area environment over UDP/IP multicast. Inital results show acceptable performance: the protocol works correctly for this application, requiring reliable, ordered and relatively slow data exchange. However, when this protocol was tested in lossy environement (over the MBONE), it was clear that a rate based flow control mechanism [13] is missing for the following reasons:

- Consider that the amount of data sent on the network exceeds de size of the input buffer. Many packets will be lost. Consider that there is a maximum number of packets stored in the protocol layer for retransmission. If the number of sent packets exceeds this limit, the sender station will no more remember it's first packets which may have been lost. In that case the sender will no more be able to retransmit it's message, if the Nack did not arrive before the old packet has been discarded.

- Consider that the number of lost packets is greater that the number of Nack options a packet can contain. This leads to problems as well.

For these reasons, special care must be taken when large amounts of data transfers are expected (e.g. for the refresh state). We are working on the design of a flow control mechanism to improve this protocol.

On the other hand, at this point, the joining station has to wait, either the correct reception of the current state (refresh) or a few seconds with no answer to its join message (in which case it decides that it must be alone in the group) before giving back the hand to the application. The joining procedure may also be done in a passive way: the station could simply listen and ask a refresh directly to the first active station.

We are also studying a mechanism to avoid the use of join messages, because their loss may lead to a joining station thinking to be alone in the group although it is not. It will then notice that all active stations have huge sequence numbers, and will expect a retransmission of all the "missing packets".

6 Conclusion

In this paper we presented a reliable multicast protocol which makes efficient use of the underlying network multicast facilities. This protocol has been implemented for a white board application and tested in a local environment. The test results are satisfactory in such low loss rate environment. A flow control mechanism is currently under study in order to have efficient operation of the protocol over the multicast back-bone over the Internet.

References

[1] J.M. Chang and N.F. Maxemchuk, "Reliable Broadcast Protocols," *ACM Transaction on Computer Systems*, Vol. 2, No 3, August 1984.

[2] H. Garcia-Molina and A. Spauster, "Message Ordering in a Multicast Environment," *Technical Report, CS-TR-161-88,* Dept. of Computer Science, Princeton University, June 1988.

[3] T.A. Joseph and K.P. Birman, "Reliable Communication in the Presence of Failures," *ACM Transaction on Computer Systems*, Vol. 5, No. 1, February 1987.

[4] S-W. Luan and V. D. Gligor, "A Fault-Tolerant Protocol for Atomic Broadcast," *Proc. IEEE 7th Reliable Distributed Systems Symposium*, pp. 112-126, 1988.

[5] F. Schneider, D. Gries, and R. Schlichting, "Fault-Tolerant Broadcasts," *Science of Computer Programming*, Vol. 4, No 1, April 1984, pp. 1-15.

[6] S. Armstrong, A. Freier and K. Marzullo, "Multicast Transport Protocol," *RFC 1301*, February 1992.

[7] S. E. Deering and D. P. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Trans. on Computer Systems*, Vol.8, May 1990, pp. 85-110.

[8] T.A. Joseph and K.P. Birman, "Reliable Broadcast Protocols," *Distriuted Systems, Sape Mullender (Ed)*, Addison-Wesley, pp. 293-317, 1989.

[9] B. Rajagopalan, "Failsafe Routing and Multicasting in Dynamic Internets," *PhD Thesis,* University of Illinois at Urbana-Champaign, 1991.

[10] Henning Schulzrinne, "A Transport Protocol for Real-Time Applications," *Internet Draft*, September 15, 1993.

[11] David L. Mills, "Network Time Protocol," *RFC 1305*, March 1992.

[12] David L. Mills, "Internet Time Synchronization," *RFC 1129*, September 1989.

[13] R. Braudes and S. Zabele, "Requirements for Multicast Protocols," *RFC 1458*, May 1993.

Unité de Recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399