A Robust Causal Order Protocol

George Marconi de A. Lima, Raimundo J. de A. Macêdo

Universidade Federal da Bahia
Laboratório de Sistemas Distribuídos - LaSiD
Prédio do CPD-UFBA, Campus de Ondina, 40.170-110 Salvador, BA, Brasil
{gmlima, macedo}@lasid.ufba.br

Abstract—

This paper presents a robust causal order protocol implemented for the BCG platform (Reliable Group Communication Base) developed at LaSiD/UFBA (Distributed System Laboratory at UFBA). It was designed using a symmetric message recovery approach in which any process of the group can detect and retransmit missing messages providing a reliable message recovery mechanism. Its algorithm, based on certain properties and structures of the BCG, makes the proposed protocol more flexible than similar protocols published to date. Data collected from experiments are also reported.

Keywords— **robust network protocols, message recovering, group communication protocols, causal ordering.**

I. INTRODUCTION

The group communication paradigm has been extensively used and recommended to provide support for reliable distributed applications [BSS91, CV95, CZ85, EMS95, Mac94, VRV93], such as load balance and fault tolerance. In such a paradigm processes are organised in sets, called groups, and all messages are multicast to the whole group. In order to preserve consistency, the group communication protocols have to offer certain services such as message ordering, membership and flow control which provide transparent co-operation among processes in the group as well as reduce complexity at the application level.

We can divide a group communication protocol into three main layers according to their goals. The lowest is the communication layer, called multicast layer, responsible for message transmission/receiving operations. The highest layer is the application layer and between both these layers the group communication layer[1] is found. Problems such as message ordering, membership agreement, etc. are solved in this middle layer. It is interesting to note that in this layered model if a problem is solved in a lower layer the upper ones need not worry about it, simplifying their functions.

If the missing message problem is solved in the multicast layer it is called a reliable multicast layer (i.e. it guarantees that there is no missing message). Because of this guarantee we can design weaker group protocols above it. In general a reliable multicast protocol works using stability of messages defined at the receiving time. In other words, if a message is known to be stable by the sender (it was received by all receivers) then the sender can discard the message. However, as a received message is kept in local buffers until delivered to the application, the sender does not know if a received message is delivered in reality. The majority of multicast protocols assume that received messages will always be delivered.

On the other hand, a situation where more flexibility is required may occur. For example, consider that the local buffer of a receiver process is full and a message, m′, with higher priority associated to it arrives. An already lower priority received message which still has not been delivered could be dropped so that the receiver process is able to get m′. In fact, some researchers argue that delivered messages must be kept in local buffers until known to be delivered by all members of the group [MES96, EMS95] (stability defined at the delivery time). Another situation where a flexible protocol is required is when some missing messages are no longer necessary. For instance, in some multimedia applications the messages have a lifetime associated to them. Thus if the time from sending to the delivery operations is greater than its lifetime the messages do not actually have to be delivered [BMR94]. Such messages have to be discarded instead of being considered lost. The problem is that there is no necessary information to solve this problem in the multicast layer. Consequently if the missing message problem is solved in this layer, recovery of unnecessary discarded messages will waste computation time.

However, if the missing message problem is solved in the middle layer we can use the semantic of group communication ordering protocols to recover missing messages as well as make the multicast layer simpler and faster. In order to do this the stability of messages defined at delivery time can be used. Furthermore this approach makes the protocols more flexible. We can, for example, discard already received messages (if necessary) or not worry about expired messages (if they exist)[2].

[1] This is a simplified view but it is useful to our objectives. Some group communications protocols have their middle layer composed by other layers such as in [RBM96].

[2] In this paper we do not treat these problems. Only the missing message problem is treated.

This paper presents a new causal order protocol which recover missing messages at the group communication layer. It was designed as a symmetric message recovery procedure where any process of the group can detect and retransmit missing messages providing a flexible and reliable recovery mechanism. Its algorithm was based on certain properties and structures of BCG protocols and on message stability information[3] exchanged among the processes by the causal order protocol of BCG [Mac95, LM97, LM99] (the relative causal order protocol). The BCG is a distributed group communication platform, implemented in C++ over a network of Unix workstations at LaSiD/UFBA, with the objective of providing a suitable environment for the design of reliable distributed applications.

This text is organised as follows. Section II presents basic concepts about multicast protocols and the missing message problem, characterising them. Section III describes BCG architecture and section IV summarises the BCG causal order protocol. Our recovery algorithm is presented in section V. Finally section VI concludes this paper.

II. BACKGROUND.

According to the way messages are recovered, the group communication protocols can be classified into two main categories: the *sender-initiated* and *receiver-initiated*. In the first the responsibility for ensuring reliable delivery lies with the sender who receives *positive acknowledge* messages (ACK) from the receivers keeping information about the state of communication up to date. Missing messages are detected only at the sender through the absence of ACKs in a timeout period. The principal disadvantage of this approach is its poor scalability and throughput because the amount of information kept by the sender is dependent on the group size. As well as this the volume of ACK messages can cause ACK implosion. In contrast, the receiver-initiated approach is based on *negative acknowledge* messages (NACK) sent by the receivers when they detect losses of messages. In this case the sender retransmits any of the reported missing messages. Although this second type has better scalability (it is independent of the group size), the sender cannot know which messages have been received and cannot discard them for long periods. In order to overcome the limitations of both approaches there are several proposed improvements. [Bar98] classifies them into different groups according to their characteristics. See a brief summary below:

- **Implosion avoidance optimisation**. There are four categories: *tree-based*, where receivers are organised according to a tree structure to minimise the number of ACKs because each receiver only sends a feedback message to its parent; *period-based* scheme, where periodically receivers send feedback information related to a block of messages instead of individual messages; and *delay-based*, where NACK messages are multicast to the group when a missing message is detected by a receiver. In this approach any receiver can retransmit, however, to avoid retransmission implosion, delays are associated with the processes. The latter approach is known as *polling-based* scheme where only a subset of receivers is responsible for sending ACKs to the sender, reducing the number of ACK messages.
- **Organisation (or model)**. The organisation can be *centralised*, *hierarchic* or *symmetrically distributed*. In the centralised organisation the sender handles feedback information (ACK or NACK) from all receivers and this can cause implosion problems and thus scalability problems. Scalability can be increased in the hierarchic model (*tree-based* optimisation) given its characteristics as explained above. In the symmetrically distributed model, SRM [FJMLZ95], an example of this, any receivers which have received a message as well as the sender are able to retransmit it. After a receiver detects missing messages it multicasts a NACK and waits for the messages. This latter approach uses the *many-to-many* group communication paradigm while the others use *one-to-many*.

Using this classification our recovery protocol was based on the *symmetrically distributed* model, providing decentralised error recovery as well as using the *periodic scheme* to avoid the problem of implosion. Periodically receivers multicast an ACK message to confirm all received messages. Based on stability and ordering information any process in the group is able to detect and retransmit missing messages. In fact, these periodic messages already exist in an asynchronous group communication protocol as ´I am alive´ messages to implement membership and failure detection services [Mac94].

III. AN OVERVIEW OF THE BCG

BCG provides several group communication services, allowing message exchange through distributed applications under the group communication paradigm (Fig.1). In the BCG core, causal [Mac95], total [Mac94, MS95, EMS95] ordering, membership [EMS95, GM98] and flow control [MES95] protocols are implemented. The multicast layer implements the communication system using as network communication subsystem UDP or TCP/IP protocols. The UDP protocol is considered only for causal order protocol over which we have implemented a message recovery mechanism.

[3] In this text *stability* refers to stability at delivery time.

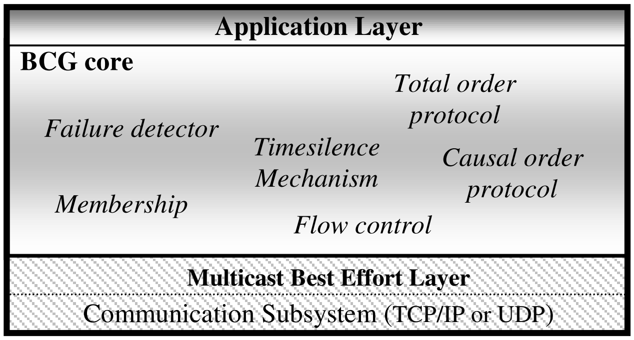

Fig.1 The Architecture of the BCG.

The protocols and services of the BCG were designed using the distributed asynchronous system model where a set of processes may be distributed in distinct processors or sites and there are no known bounds for message transmission and processing times. Originally the following were the assumptions: (1) only process crash faults exist (i.e. processes stop functioning); (2) once a message has been sent, eventually it is received at the destination; (3) the multicast layer provides FIFO order between any pairs of processes. Assumption (1) is related to process fault model while the others are related to communication layer. Here we have widened assumption (2), assuming message omission faults, and eliminated assumption (3). Obviously in an asynchronous model the concept of omission faults is not precise because lost and late message cannot be distinguished. Thus, we regard either missed or too late messages as *communication omission faults* (or simply *omission faults*) and use a predefined timeout value to detect them.

We assume the process crashes are dully treated by the membership service of BCG. That is, a process crash will always be detected and the crashed process removed from the group membership. Furthermore, group view changes and message delivery will be reported in a mutually consistent way to all functioning group processes (i.e. synchronous view semantics). For details refer to [Mac94, EMS95, GM98]. As in this paper we do not directly address process crashes (as this is handled by the membership protocol mentioned), we assume that a message sent by a functioning group member will arrive at its destinations after a finite number of retransmissions.

We also consider that periodic messages, called null messages, are multicast to the group by the timesilence mechanism (Fig.1). Such messages are necessary for the failure detection and membership services to work properly [EMS95, Mac94]. Our proposed message recovery protocol takes advantage of these null messages, used as ACK messages, to detect missing messages and minimise the implosion of ACKs.

Before describing our protocol in the following sections we will present some concepts of the BCG causal protocol, *causal blocks,* and *block matrix* [MES93, Mac95]. Causal blocks and block matrix are structures designed to maintain ordering and reliability information for group communication [MES93, Mac94].

A. An Overview of the Causal Blocks Model

Consider a group of processes $g = \{p_1 , p_2 , ... , p_n\}$. Each process p_i has a logical clock BC_i (*Block Counter*) which is initialised with zero by all processes of g when g is created. Transmitted messages are timestamped with *block numbers* (i.e. Block Counter values at sending time), and, as is the case in Lamport´s Logical Clock [Lam78], timestamping using Block Counters will respect causality [Mac95]. The two events under which BC_i is incremented is $send_i(m)$ and $deliver_i(m)$:

(R1.0) Just before $send_i(m)$: $BC_i = BC_i + 1$; $m.b = BC_i$.
(R1.1) Just before $deliver_i(m)$: $BC_i = \max\{BC_i ,m.b\}$.

According to the rules above, as shown in [Mac94, Mac95], any distinct messages m and m′ are related such as: $send(m) \rightarrow send(m') \Rightarrow m.b < m'.b$[4]. As well as this, any distinct messages multicast with the same block number are necessarily concurrent and these messages must have been multicast by distinct processes.

Let us consider a bi-dimensional matrix, BM, called block matrix, kept by all processes p_i which belong to g. Each row β of BM_i, called Causal Block $BM_i[\beta]$, represents the transmitted or received messages by p_i, such that their block numbers are equal to β. The number of columns of BM corresponds to $|g|$ (the size of g). Whenever a process p_i sends or receives a multicast message m with a new block number β' it sets $BM_i[\beta'][m.s]$ = '+', where m.s represents the sender of m, in order to represent the send/receive operations, respectively.

Fig.2 shows a BM for a group with six processes. Supposing that this is the BM of p_1 it indicates that the last transmitted message has a block number equal to 5 and the last received messages from p_2, p_3, p_4, p_5 and p_6 have, respectively, 6, 3, 4, 5 and 2 as their block numbers.

BC	p_1	p_2	p_3	p_4	p_5	p_6
1	+			+		
2		+			+	+
3			+			
4	+			+	+	
5	+				+	
6		+				

Fig.2 A BM for a group with six processes.

[4] The symbol $\rightarrow$ corresponds to the happen-before relation [Lam78].

Based on the rules above and on this representation it is interesting to note that:

- In a Causal Block only concurrent messages are represented.
- Consider m and m', represented in BM[β] e BM[β'], respectively, and $m \rightarrow m'$ then $\beta < \beta'$.

For more detailed description see [Mac94].

IV. CAUSAL ORDER PROTOCOL

Consider that any transmitted message m is timestamped not only with its block number, m.b, but also with the block number of the last delivered message from each of the group members at the time m is sent. When m is received at a destination process p, it can be delivered immediately after the messages represented by those block numbers (messages which causally happened before m) have also been delivered. In order to do so each process $p_i \in g = \{p_1, p_2, ..., p_n\}$ maintains a vector of block numbers of last delivered messages (LDV - Last Delivered Vector) whose size is equal to $|g|$. Thus the LDV and the block number are the timestamps of every transmitted message m (m.ldv and m.b, respectively). Assuming that m.s represents the identifier of the process which sent m, the following rules, executed by p_i, are responsible for updating LDV_i:

(R2.0)	Just before *$send_i(m)$*:	m.ldv $\leftarrow LDV_i$[5].
(R2.1)	Just after *$send_i(m)$*:	$LDV_i[i] \leftarrow$ m.b.
(R2.2)	Just after *$deliver_i(m)$*:	LDV[m.s] $\leftarrow$ m.b.

As the LDV represents the causal relation among distributed events, in order to deliver any received message m, a process $p_i \in g$ must compare its LDV with m.ldv according to the following rule:

Delivery Condition Rule: m can be delivered by p_i if $m.ldv[j] \leq LDV_i[j]$ for all $p_j \in g$.

The properties and proofs of the protocol can be seen in [Mac94, Mac95].

V. ADDING ROBUSTNESS TO THE CAUSAL ORDER PROTOCOL

The causal relation among messages is represented by the LDV as seen in the section above. In fact, such a vector brings information about stability and can be used for adding robustness in the causal order protocol. The basic idea is to maintain the LDV of the last received message. In order to do this each process $p_i \in g$ keeps a matrix of LDV, termed MLDV[6], updated as follows:

(R3.0)	At $receive_i(m)$: then	if $m.b > \max(m'.b) \mid m'.s = m.s$ $MLDV_i[m.s] \leftarrow$ m.ldv.
(R3.1)	At $transmitter_i(m)$:	$MLDV_i[i] \leftarrow$ mi.ldv.
(R3.2)	At $deliver_i(m)$:	$MLDV_i[i][m.s] \leftarrow$ m.bn.

It is interesting to note that while LDV_i represents knowledge of p_i about the delivered messages in g, $MLDV_i$ represents the 'view' of p_i about the LDV knowledge of other processes in g. Our recovery algorithm is based on this 'view'. However, this is insufficient for the design of a robust protocol. More information is necessary if we want to know if a message has been missed. This can be obtained through knowledge about received messages.

Definition[7]: Consider a group $g = \{p_1, p_2, ..., p_n\}$. Take the greatest block number, β, called MB (maximum block number), such that for all j ($p_j \in g$), $1 \leq j \leq n$, the j^{th} entry of BM[β] either (1) has ´+´ or (2) is a blank and there exists $\beta' > \beta$ such that j^{th} entry of BM[β'] has a ´+´.

The MB value, calculated by all processes in the group g and sent together with any transmitted message, is used to maintain a MBV (MB Vector) which has one entry for each process of g. Such a vector is updated at receiving/transmission operations as follows:

(R4.0)	At $receiver_i(m)$: then	if $m.b > \max(m'.b) \mid m'.s = m.s$ $MBV_i[m.s] \leftarrow$ m.mb.
(R4.1)	At $transmitter_i(m)$:	$MBV_i[i] \leftarrow$ m.mb.

According to the rules above, MBV_i maintains the p_i's knowledge of the MB values of all processes in g. Thus, any process $p_i \in g$ using its knowledge about delivered (MLDV) and received messages (MBV) can suspect losses of messages.

(P1.0) *Property*: If a message m, sent by a process $p_k \in g$, was not received by any $p_j \in g$ then there exist some processes $p_i \in g$ ($i \neq j$) which received m such that eventually $MLDV_i[j][k] < m.b < \min(MBV_i)$.

Proof: Consider a group $g = \{p_1, p_2, ..., p_n\}$. The proof will be in two parts.

---

[5] The symbol $\leftarrow$ represents an attribution operation.

[6] Although LDV_i and $MLDV_i[i]$ have the same meaning, we will use both to maintain compatibility with the original causal order protocol [Mac94, Mac95].

[7] This definition corresponds to the block completion definition [Mac94]. However, we cannot use the same term because we have assumed different hypothesis (section III).

Part 1. $\underline{MLDV_i[j][k] < m.b}$: if a message m with block number m.b sent by p_k was not received by p_j then $LDV_j[k] < m.b$ and $k \neq j$. Due to the timesilence mechanism and as the LDV values timestamp any transmitted message (by R2.0), p_i will eventually receive a message from p_j (section III) making $MLDV_i[j]$ equal to LDV_j (by R3.0).

Part 2. $\underline{m.b < \min(MBV_i)}$: As every process pi always transmits messages (timesilence mechanism) and the Causal Block advances during transmission (by R1.0) $MBV_i[i]$ will also advance. As rule R4.0 advances $MBV_i[k]$ ($i \neq k$), eventually $m.b < \min(MBV_i)$.

Based on the property above a process can compare its MLDV and min(MBV) values to suspect if other processes have missed any message. If so, the process can retransmit them. This is the basic idea of the recovery algorithm described below.

(R5.0) ***Retransmission Condition Rule***: any message $m_k \in BM_i$ ($m.s = p_k$) such that $MLDV_i[j][k] < \min(m_k.bn) < \min(MBV_i)$ and $i \neq k \neq j$, can be retransmitted by $p_i \in g = \{p_1, p_2,...,p_n\}$.

As we can see the detection of missing messages is dependent on the activity of the group. That is, missing messages are only detected when processes receive a message informing the state of the process which missed them. Therefore, if the group is very active, missing messages are detected fast otherwise the recovery and detection can be slow. Thus the null message transmission procedure has a fundamental role in the performance of the recovery mechanism. However, if there is a high frequency of transmissions of null messages it can increase the network load. It is, therefore, necessary to balance time of recovery and timesilence mechanism.

B. The Algorithms

The figures 3 to 6 represent the algorithm of the proposed protocol. Each procedure is an independent thread which runs concurrently in a process and every process in a group runs the same algorithms in a symmetric approach. Fig.3 illustrates the detection and retransmission procedure. As soon as this thread detects a missing message (line 3) it sets a random timeout to send such a message (line 4). This timeout is an optimisation to avoid retransmission implosion because all processes in the same group can detect the same faults. This timeout can be cancelled before the retransmission (the lines 7 and 8 of Fig.4). With a suitable adjustment of this timeout the number of retransmissions can be decreased.

It is interesting to note that only the message with the smallest block number is retransmitted. The reason for this is that such a message happened before the others (causal relation). Therefore there is no guarantee that these messages have not been received (MLDV is a view of delivered messages). Once the first message is recovered a retransmission procedure will be restarted if another missing message is detected.

```
Procedure send_missing_message

1. Do it forever
2.     if min(MBV) has changed
3      ∀ m_k ∈ BM_i | R5.0 is valid
4.         after a random timeout send m with minimum
           block number
5. enddo
```

Fig.3 Message detection and retransmission procedure.

The receiving, transmitting and delivery procedures, are showed in the figures 4 to 6. Their algorithms, together with the procedure above implement the rules viewed in the earlier sections.

```
Procedure receiver
1. Do it forever
2.     receive m from the transport layer
3.   if there is any timeout to retransmit m then
       set a new timeout
4.   else
5.     if m.b > max(m′.b) | m′.s = m.s ∀ received m′
         then
6.         MLDV_i[m.s] ← m.ldv; MBV_i[m.s] ← m.mb
7.         if there is any timeout for any m′ and
             R5.0 is not valid then
8.             cancel the timeout
9.         if m ∉ BM_i then
10.            mark BM_i[m.b] with a ´+´; BM_i[m.b] ← m
11.            start delivery procedure.
12. enddo
```

Fig.4 Recovery procedure.

The receiving procedure receives messages from the transport layer, executes certain actions and starts the delivery procedure. First of all, in order to minimise the number of retransmissions, if there is any pending timeout for the received message a new timeout is set (line 3) and the received message is discarded. The timeout is only cancelled in line 8 if the retransmission condition rule is no longer valid. As messages do not arrive in FIFO order, only receiving of messages with the greatest block number can update MLDV (line 5). Finally received messages are put in BM and the delivery procedure is started as well. Due to message retransmissions, these final actions are performed if received messages are not yet present in BM.

The transmission and delivery procedures, Fig.5 and Fig.6, are simpler and very similar to the original causal protocol [Mac95].

```
Procedure transmitter

1. BC_i = BC_i + 1; m.b = BC_i
2. m.ldv ← LDV_i
3. MLDV_i[i][i] ← LDV_i[i] ← BC_i
4. BM_i ← m
5. multicast m through the multicast layer
```

Fig.5 Transmission procedure.

As we can see in these algorithms, the protocol can easily be modified to gain flexibility according to the semantic of the application. For instance, consider a distributed multimedia application. If a process discovers that a message does not make sense (i.e. its lifetime is expired) it can advance its LDV without delivering the message and the recovery procedure will not be started.

```
Procedure deliver
1. if ∃ m ∈ BM_i | m is not delivered Λ m.ldv ≥ LDV_i
then
2.     deliver m to p_i
3.     BC_i ← max{BC_i, m.b}
4.     MLDV_i[i][m.s] ← LDV_i[m.s] ← m.b
5.     signal deliver procedure recursively
```

Fig 6. Delivery procedure.

C. *Correctness*

In order to prove the correctness of the protocol we have to prove the theorem below:

(T1.0) Every missing message is detected, eventually recovered and delivered without violating causality.
Proof: [Mac94] has proved that any sent message will be eventually delivered by the original causal order protocol. The lemmas below prove the other parts of the theorem.

The lemmas L1.0 and L1.1 show the liveness property of the protocol, i.e., all missing messages are eventually detected and recovered. With regard to the safety property we will see that neither the message retransmission procedure nor the multicast layer assumptions have any effect on the message delivery order (L1.3). Finally lemma L1.2 shows that any detected missing message is a valid message. As stated before we assume that a membership and a timesilence services exit in order to guarantee that functioning members always have a mutual consistent view of the group membership.

(L1.0) The existence of a missing message is eventually detected (*liveness*).
Proof: The detection of a missing message m by a process $p_i \in g = \{p_1, p_2, ..., p_n\}$ happens when $MLDV_i[j][k] < \min(m.bn) < \min(MBV_i)$, supposing that m was originally sent by $p_k \in g$ and was not received by some $p_j \in g$. According to property P1.0 the relation above is eventually true, and so the detection is also eventually true. In the worst case there is only one detecting process, the original sender process ($p_k = p_i$).

(L1.1) *Lemma:* A missing message is eventually received and delivered (*liveness).*
Proof: Recalling our assumptions, any transmitted message arrive at its destination after a finite number of retransmissions. As a detecting process retransmits a given message after it detects the missing (Fig.3), at least one of the retransmissions will arrive successfully, otherwise the group is not a connected one.

(L1.2) *Lemma*: A detected missing message m is a valid message (was originally sent by a process belonging to g) (*validity*).
Proof: Consider a group $g = \{p_1, p_2,...,p_n\}$ and suppose that m was detected as a missing message by $p_i \in g$. Thus, $MLDV_i[j][k] < m.bn < \min(MBV_i)$ (by R5.0) for some $p_j \in g$ and $p_k \in g$. This means that p_i has received m from p_k and delivered it. As m was delivered by p_i, m is a valid message.

(L1.3) *Lemma:* A received message is delivered without violating causality (*safaty*).
Proof: Consider a group $g = \{p_1, p_2,...,p_n\}$. Suppose by absurd that a message m, such that $m' \to m$, was delivered by $p_j \in g$ and m′ was not yet delivered by p_j. Now suppose without losing of generality that m′ was sent by $p_i \in g$ with $i \neq j$. If p_j delivered m, $LDV_j \geq m.ldv$ and so $LDV_j[i] \geq m.ldv[i]$. As $m' \to m$, $m'.ldv[i] \leq m.ldv[i]$. By rule

R2.2 it is known that $LDV_j[i]$ only advances when p_j delivers messages sent by p_i. Therefore m′ must have been delivered by p_j before m (contradiction).

VI. IMPLEMENTATION

The protocol proposed was implemented in C++ over a network of Unix workstations. An experiment with a group of three processes was configured where the timesilence mechanism was set to transmit messages at intervals of 500ms for all processes. One of these processes was chosen as the sender and the others the receivers. The receiver processes were also able to retransmit messages when they detected loss of messages.

Messages with block numbers between 10 and 15 were dropped at one of the receivers. The figure 7 shows when these missing messages were received by this receiver. The first line of the figure is the BC values of such a receiver[8]. The line named *Rcv* represents which messages were recovered (indicted by their block numbers). Thus we can see that message 10 was recovered when the BC was 17, i.e., after receiving message 16. It interesting to note that message 10 was recovered twice. This is because both the transmitter and the other receiver retransmitted it.

BC	17	18	19	20	21	22	23	24	25	26	27
Rcv	10	10	11			12	13		14	15	

Fig.7. Illustration of the message recovery protocol for a group of three processes.

VII. CONCLUSIONS

The missing message problem has an important influence on distributed protocol implementation. The performance and complexity of distributed protocols depend on the decision of where and how to detect and recover missing messages. To our best knowledge, the protocols developed to date [MRTW97, PSLB97, YGS95, BSS91] were designed to recover missing messages based on information at the receiving time. As we have discussed earlier, recovery based on this kind of information can be too restrictive for certain types of applications. We have contributed to the area by proposing and implementing a new message recovery mechanism based on stability defined at the delivery time. According to the semantic of application, the protocol can be configured to ignore expired messages or discard already received messages by properly manipulating the MLDV vector. This can be very useful for soft real-time applications which have lifetimes or priorities associated to transmitted messages.

Moreover, our protocol was designed as a symmetric recovery procedure in which any process of the group can detect and retransmit missing messages. This characteristic contrasts with the majority of message recovery protocols proposed to date where the responsibility of the message retransmissions generally lies only with the sender (unique point of failure). Although SRM [FJMLZ95] is also symmetric, it does not take advantage of message delivery information at the group communication service level (e.g. causal order protocol level).

[8]As our objective is only to illustrate the behaviour of the recovery procedure instead of to quantify its performance, the measurements are based on logical time.

ACKNOWLEDGEMENTS

This work was partially supported by LOCUS/ProTeM-CC project (Phase III) and the second author is also supported by CNPq Grant 300013/87-6 (BCG Research Project).

REFERENCES

[ACM95] G. A. Alvarez, F. Cristian and S. Mishra. *On-Demand Asynchronous Atomic Broadcast*. 5th IFIP Working Conf. on Dependable Computing for Critical Applications, 1995.

[BJ87] K. Birman, T. A. Joseph. *Reliable Communication in the Presence of Failures*. ACM TDCS, (5,1), pp. 47-76, 1987.

[Bar98] Barcellos, A. M. P. *PRMP: A Scaleable Polling-based Reliable Multicast Protocol*. Ph. D. Thesis. Newcastle upon Tyne, September 1998.

[BMR94] R. Baldoni, A. Mostefaoui and M. Raynal, *Causal Deliveries in Unreliable Networks with Real-Time Delivery Constraints.* Technical Report 2427, INRIA. December 1994.

[BSS91] K. Birman, A. Schiper and P. Stephenson. *Lighweight Causal and Atomic Group Multicast*. ACM Transaction on Computing Systems, (9,3), pp. 272-314, August 1991.

[CV95] F. J. N. Cosquer and P. Veríssimo. *The Impact of Group Communication Paradigms on Groupware Support*. Proc. of the 5th Workshop on Future Trends of Distributed Computing Systems. Korea, 1995.

[CZ85] D. Cheriton, W. Zwaenepoel. *Distributed Process Groups in V Kernel*. ACM TOCS, (3,2), pp. 97-107, 1985.

[EMS95] P. D. Ezhilchelvan, R. A Macêdo and S. K. Shrivastava. *Newtop: A Fault-Tolerant Group Communication Protocol.* Proc. of the 15th International Conference on Distributed Computing Systems, IEEE Computer Society, pp. 296-306, Canadá , June 1995.

[FJMLZ95] S. Floyd, V. Jacobson, S. McCanne, C. Liu, L. Zhang. *A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing*. ACM SIGCOMM'95. Cambridge, USA, August 1995.

[GM98] F. G. P. Greve and R. J. A. Macêdo. *The BCG Membership Service Performance Analysis*. In Proc. of XVI Simpósio Brasileiro de Redes de Computadores. Rio de Janeiro, May 1998.

[Lam78] L. Lamport. *Time, Clocks and Ordering of Events in a Distributed System*. CACM, (21, 7), pp. 558-265, July 1978,.

[LM97] G. M. de A. Lima and R. J. A. Macêdo. *Avaliação de Desempenho do protocolo BCGcausal* Technical Report, RTI-003/97. November 1997.

[LM99] G. M. de A. Lima and R. J. A. Macêdo. *Avaliação de Desempenho de Protocolos de Ordenação Causal para Comunicação em Grupo*. XXV Conferencia Latinoamericana de Informática, CLEI99, 1999.

[Mac94] R. J. de A. Macêdo. *Fault Tolerant Group Communication Protocols for Asynchronous Systems*. Ph.D. Thesis, Computing Science Department, University of Newcastle upon Tyne, UK, 1994.

[Mac95] R. J. de A. Macêdo. *Causal Order Protocols for Group Communication*. SBRC95, Belo Horizonte-MG, pp. 265-283, May 1995.

[MES96] R. J. A. Macêdo and P. D. Ezhilchelvan, S. K. Shrivastava. *Buffer Overflow Avoidance Techniques for Groups Communication Protocols*. SBRC96, pp. 633-652, Fortaleza-CE, May 1996.

[MES93] R. J. A. Macêdo and P. D. Ezhilchelvan, S. K. Shrivastava. Modeling *Group Communication Using Causal Blocks*. 5th European Workshop on Dependable Computing, Lisbon, February, 1993.

[MRTW97]K. Miller, K. Robertson, A. Tweedly and M. White. *Starbus Multicast File Transfer Protocol (MFTP) Specification*. Internet draft (expired), January 1997.

[PBS89] L. L. Peterson, N. C. Buchholz and R. D. Schlichting. *Preserving and Using Context Information in Interprocess Comunication*. ACM Transaction on Computing Systems, (7,3), pp. 217-246, August 1989.

[PSLB97] S. Paul, K. Sabnani, J. Lin and S. Bhattacharryya. *Reliable Multicast Transport Protocol (RMTP)*. IEEE Journal on Selected Areas in Communications, Vol. 13, No. 3, April 1997.

[RBM96] R. Renesse, K. Birman and S. Maffeis. *HORUS, A Flexible Group Communication System*. Communication of the ACM, April 1996.

[VRV93] P. Veríssimo, L. Rodrigues and Werner Vogels. *Group Orientation: a Paradigm for Modern Distributed Systems*. ESPRIT Basic Research Project First Year Report, volume 1, October 1993.

[YGS95] R. Yavatkar, J. Griffioen and M. Sudan. *A Reliable Dissemination for Interactive Collaborative Applications*. ACM Multimedia, 1995.

A Robust Causal Order Protocol

George Marconi de A. Lima, Raimundo J. de A. Macêdo

Universidade Federal da Bahia
Laboratório de Sistemas Distribuídos - LaSiD
Prédio do CPD-UFBA, Campus de Ondina, 40.170-110 Salvador, BA, Brasil
{gmlima, macedo}@lasid.ufba.br

Abstract—

This paper presents a robust causal order protocol implemented for the BCG platform (Reliable Group Communication Base) developed at LaSiD/UFBA (Distributed System Laboratory at UFBA). It was designed using a symmetric message recovery approach in which any process of the group can detect and retransmit missing messages providing a reliable message recovery mechanism. Its algorithm, based on certain properties and structures of the BCG, makes the proposed protocol more flexible than similar protocols published to date. Data collected from experiments are also reported.

Keywords— **robust network protocols, message recovering, group communication protocols, causal ordering.**

I. Introduction

The group communication paradigm has been extensively used and recommended to provide support for reliable distributed applications [BSS91, CV95, CZ85, EMS95, Mac94, VRV93], such as load balance and fault tolerance. In such a paradigm processes are organised in sets, called groups, and all messages are multicast to the whole group. In order to preserve consistency, the group communication protocols have to offer certain services such as message ordering, membership and flow control which provide transparent co-operation among processes in the group as well as reduce complexity at the application level.

We can divide a group communication protocol into three main layers according to their goals. The lowest is the communication layer, called multicast layer, responsible for message transmission/receiving operations. The highest layer is the application layer and between both these layers the group communication layer[1] is found. Problems such as message ordering, membership agreement, etc. are solved in this middle layer. It is interesting to note that in this layered model if a problem is solved in a lower layer the upper ones need not worry about it, simplifying their functions.

If the missing message problem is solved in the multicast layer it is called a reliable multicast layer (i.e. it guarantees that there is no missing message). Because of this guarantee we can design weaker group protocols above it. In general a reliable multicast protocol works using stability of messages defined at the receiving time. In other words, if a message is known to be stable by the sender (it was received by all receivers) then the sender can discard the message. However, as a received message is kept in local buffers until delivered to the application, the sender does not know if a received message is delivered in reality. The majority of multicast protocols assume that received messages will always be delivered.

On the other hand, a situation where more flexibility is required may occur. For example, consider that the local buffer of a receiver process is full and a message, m′, with higher priority associated to it arrives. An already lower priority received message which still has not been delivered could be dropped so that the receiver process is able to get m′. In fact, some researchers argue that delivered messages must be kept in local buffers until known to be delivered by all members of the group [MES96, EMS95] (stability defined at the delivery time). Another situation where a flexible protocol is required is when some missing messages are no longer necessary. For instance, in some multimedia applications the messages have a lifetime associated to them. Thus if the time from sending to the delivery operations is greater than its lifetime the messages do not actually have to be delivered [BMR94]. Such messages have to be discarded instead of being considered lost. The problem is that there is no necessary information to solve this problem in the multicast layer. Consequently if the missing message problem is solved in this layer, recovery of unnecessary discarded messages will waste computation time.

However, if the missing message problem is solved in the middle layer we can use the semantic of group communication ordering protocols to recover missing messages as well as make the multicast layer simpler and faster. In order to do this the stability of messages defined at delivery time can be used. Furthermore this approach makes the protocols more flexible. We can, for example, discard already received messages (if necessary) or not worry about expired messages (if they exist)[2].

[1] This is a simplified view but it is useful to our objectives. Some group communications protocols have their middle layer composed by other layers such as in [RBM96].

[2] In this paper we do not treat these problems. Only the missing message problem is treated.

This paper presents a new causal order protocol which recover missing messages at the group communication layer. It was designed as a symmetric message recovery procedure where any process of the group can detect and retransmit missing messages providing a flexible and reliable recovery mechanism. Its algorithm was based on certain properties and structures of BCG protocols and on message stability information[3] exchanged among the processes by the causal order protocol of BCG [Mac95, LM97, LM99] (the relative causal order protocol). The BCG is a distributed group communication platform, implemented in C++ over a network of Unix workstations at LaSiD/UFBA, with the objective of providing a suitable environment for the design of reliable distributed applications.

This text is organised as follows. Section II presents basic concepts about multicast protocols and the missing message problem, characterising them. Section III describes BCG architecture and section IV summarises the BCG causal order protocol. Our recovery algorithm is presented in section V. Finally section VI concludes this paper.

II. BACKGROUND.

According to the way messages are recovered, the group communication protocols can be classified into two main categories: the *sender-initiated* and *receiver-initiated*. In the first the responsibility for ensuring reliable delivery lies with the sender who receives *positive acknowledge* messages (ACK) from the receivers keeping information about the state of communication up to date. Missing messages are detected only at the sender through the absence of ACKs in a timeout period. The principal disadvantage of this approach is its poor scalability and throughput because the amount of information kept by the sender is dependent on the group size. As well as this the volume of ACK messages can cause ACK implosion. In contrast, the receiver-initiated approach is based on *negative acknowledge* messages (NACK) sent by the receivers when they detect losses of messages. In this case the sender retransmits any of the reported missing messages. Although this second type has better scalability (it is independent of the group size), the sender cannot know which messages have been received and cannot discard them for long periods. In order to overcome the limitations of both approaches there are several proposed improvements. [Bar98] classifies them into different groups according to their characteristics. See a brief summary below:

- **Implosion avoidance optimisation**. There are four categories: *tree-based*, where receivers are organised according to a tree structure to minimise the number of ACKs because each receiver only sends a feedback message to its parent; *period-based* scheme, where periodically receivers send feedback information related to a block of messages instead of individual messages; and *delay-based*, where NACK messages are multicast to the group when a missing message is detected by a receiver. In this approach any receiver can retransmit, however, to avoid retransmission implosion, delays are associated with the processes. The latter approach is known as *polling-based* scheme where only a subset of receivers is responsible for sending ACKs to the sender, reducing the number of ACK messages.
- **Organisation (or model)**. The organisation can be *centralised*, *hierarchic* or *symmetrically distributed*. In the centralised organisation the sender handles feedback information (ACK or NACK) from all receivers and this can cause implosion problems and thus scalability problems. Scalability can be increased in the hierarchic model (*tree-based* optimisation) given its characteristics as explained above. In the symmetrically distributed model, SRM [FJMLZ95], an example of this, any receivers which have received a message as well as the sender are able to retransmit it. After a receiver detects missing messages it multicasts a NACK and waits for the messages. This latter approach uses the *many-to-many* group communication paradigm while the others use *one-to-many*.

Using this classification our recovery protocol was based on the *symmetrically distributed* model, providing decentralised error recovery as well as using the *periodic scheme* to avoid the problem of implosion. Periodically receivers multicast an ACK message to confirm all received messages. Based on stability and ordering information any process in the group is able to detect and retransmit missing messages. In fact, these periodic messages already exist in an asynchronous group communication protocol as ´I am alive´ messages to implement membership and failure detection services [Mac94].

III. AN OVERVIEW OF THE BCG

BCG provides several group communication services, allowing message exchange through distributed applications under the group communication paradigm (Fig.1). In the BCG core, causal [Mac95], total [Mac94, MS95, EMS95] ordering, membership [EMS95, GM98] and flow control [MES95] protocols are implemented. The multicast layer implements the communication system using as network communication subsystem UDP or TCP/IP protocols. The UDP protocol is considered only for causal order protocol over which we have implemented a message recovery mechanism.

[3] In this text *stability* refers to stability at delivery time.

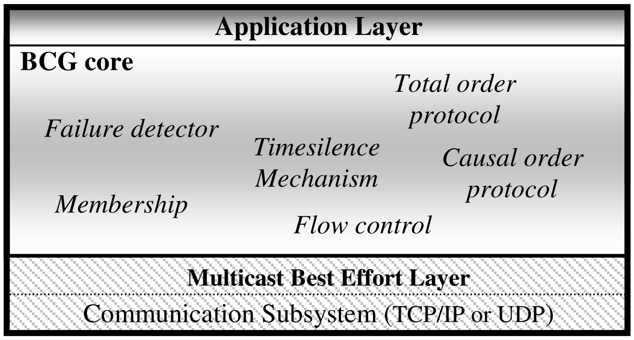

Fig.1 The Architecture of the BCG.

The protocols and services of the BCG were designed using the distributed asynchronous system model where a set of processes may be distributed in distinct processors or sites and there are no known bounds for message transmission and processing times. Originally the following were the assumptions: (1) only process crash faults exist (i.e. processes stop functioning); (2) once a message has been sent, eventually it is received at the destination; (3) the multicast layer provides FIFO order between any pairs of processes. Assumption (1) is related to process fault model while the others are related to communication layer. Here we have widened assumption (2), assuming message omission faults, and eliminated assumption (3). Obviously in an asynchronous model the concept of omission faults is not precise because lost and late message cannot be distinguished. Thus, we regard either missed or too late messages as *communication omission faults* (or simply *omission faults*) and use a predefined timeout value to detect them.

We assume the process crashes are dully treated by the membership service of BCG. That is, a process crash will always be detected and the crashed process removed from the group membership. Furthermore, group view changes and message delivery will be reported in a mutually consistent way to all functioning group processes (i.e. synchronous view semantics). For details refer to [Mac94, EMS95, GM98]. As in this paper we do not directly address process crashes (as this is handled by the membership protocol mentioned), we assume that a message sent by a functioning group member will arrive at its destinations after a finite number of retransmissions.

We also consider that periodic messages, called null messages, are multicast to the group by the timesilence mechanism (Fig.1). Such messages are necessary for the failure detection and membership services to work properly [EMS95, Mac94]. Our proposed message recovery protocol takes advantage of these null messages, used as ACK messages, to detect missing messages and minimise the implosion of ACKs.

Before describing our protocol in the following sections we will present some concepts of the BCG causal protocol, *causal blocks,* and *block matrix* [MES93, Mac95]. Causal blocks and block matrix are structures designed to maintain ordering and reliability information for group communication [MES93, Mac94].

A. An Overview of the Causal Blocks Model

Consider a group of processes $g = \{p_1, p_2, \ldots, p_n\}$. Each process p_i has a logical clock BC_i (*Block Counter*) which is initialised with zero by all processes of g when g is created. Transmitted messages are timestamped with *block numbers* (i.e. Block Counter values at sending time), and, as is the case in Lamport´s Logical Clock [Lam78], timestamping using Block Counters will respect causality [Mac95]. The two events under which BC_i is incremented is $send_i(m)$ and $deliver_i(m)$:

(R1.0) Just before $send_i(m)$: $BC_i = BC_i + 1$; $m.b = BC_i$.
(R1.1) Just before $deliver_i(m)$: $BC_i = \max\{BC_i, m.b\}$.

According to the rules above, as shown in [Mac94, Mac95], any distinct messages m and m′ are related such as: $send(m) \rightarrow send(m') \Rightarrow m.b < m'.b$[4]. As well as this, any distinct messages multicast with the same block number are necessarily concurrent and these messages must have been multicast by distinct processes.

Let us consider a bi-dimensional matrix, BM, called block matrix, kept by all processes p_i which belong to g. Each row β of BM_i, called Causal Block $BM_i[\beta]$, represents the transmitted or received messages by p_i, such that their block numbers are equal to β. The number of columns of BM corresponds to $|g|$ (the size of g). Whenever a process p_i sends or receives a multicast message m with a new block number β′ it sets $BM_i[\beta'][m.s]$ = ‘+’, where m.s represents the sender of m, in order to represent the send/receive operations, respectively.

Fig.2 shows a BM for a group with six processes. Supposing that this is the BM of p_1 it indicates that the last transmitted message has a block number equal to 5 and the last received messages from p_2, p_3, p_4, p_5 and p_6 have, respectively, 6, 3, 4, 5 and 2 as their block numbers.

BC	p_1	p_2	p_3	p_4	p_5	p_6
1	+			+		
2		+			+	+
3			+			
4	+			+	+	
5	+				+	
6		+				

Fig.2 A BM for a group with six processes.

[4] The symbol → corresponds to the happen-before relation [Lam78].

Based on the rules above and on this representation it is interesting to note that:

- In a Causal Block only concurrent messages are represented.
- Consider m and m', represented in BM[β] e BM[β'], respectively, and $m \rightarrow m'$ then $\beta < \beta'$.

For more detailed description see [Mac94].

IV. Causal Order Protocol

Consider that any transmitted message m is timestamped not only with its block number, m.b, but also with the block number of the last delivered message from each of the group members at the time m is sent. When m is received at a destination process p, it can be delivered immediately after the messages represented by those block numbers (messages which causally happened before m) have also been delivered. In order to do so each process $p_i \in g = \{p_1, p_2, ..., p_n\}$ maintains a vector of block numbers of last delivered messages (LDV - Last Delivered Vector) whose size is equal to $|g|$. Thus the LDV and the block number are the timestamps of every transmitted message m (m.ldv and m.b, respectively). Assuming that m.s represents the identifier of the process which sent m, the following rules, executed by p_i, are responsible for updating LDV_i:

(R2.0)	Just before *$send_i(m)$*:	m.ldv $\leftarrow LDV_i$[5].
(R2.1)	Just after *$send_i(m)$*:	$LDV_i[i] \leftarrow$ m.b.
(R2.2)	Just after *$deliver_i(m)$*:	LDV[m.s] $\leftarrow$ m.b.

As the LDV represents the causal relation among distributed events, in order to deliver any received message m, a process $p_i \in g$ must compare its LDV with m.ldv according to the following rule:

Delivery Condition Rule: m can be delivered by p_i if $m.ldv[j] \leq LDV_i[j]$ for all $p_j \in g$.

The properties and proofs of the protocol can be seen in [Mac94, Mac95].

V. Adding Robustness To The Causal Order Protocol

The causal relation among messages is represented by the LDV as seen in the section above. In fact, such a vector brings information about stability and can be used for adding robustness in the causal order protocol. The basic idea is to maintain the LDV of the last received message. In order to do this each process $p_i \in g$ keeps a matrix of LDV, termed MLDV[6], updated as follows:

(R3.0)	At $receive_i(m)$: then	if m.b > max(m'.b) | m'.s = m.s $MLDV_i$[m.s] $\leftarrow$ m.ldv.
(R3.1)	At $transmitter_i(m)$:	$MLDV_i[i] \leftarrow$ mi.ldv.
(R3.2)	At $deliver_i(m)$:	$MLDV_i[i]$[m.s] $\leftarrow$ m.bn.

It is interesting to note that while LDV_i represents knowledge of p_i about the delivered messages in g, $MLDV_i$ represents the 'view' of p_i about the LDV knowledge of other processes in g. Our recovery algorithm is based on this 'view'. However, this is insufficient for the design of a robust protocol. More information is necessary if we want to know if a message has been missed. This can be obtained through knowledge about received messages.

Definition[7]: Consider a group $g = \{p_1, p_2, ..., p_n\}$. Take the greatest block number, β, called MB (maximum block number), such that for all j ($p_j \in g$), $1 \leq j \leq n$, the j^{th} entry of BM[β] either (1) has ´+´ or (2) is a blank and there exists $\beta' > \beta$ such that j^{th} entry of BM[β'] has a ´+´.

The MB value, calculated by all processes in the group g and sent together with any transmitted message, is used to maintain a MBV (MB Vector) which has one entry for each process of g. Such a vector is updated at receiving/transmission operations as follows:

(R4.0)	At $receiver_i(m)$: then	if m.b > max(m'.b) | m'.s = m.s MBV_i[m.s] $\leftarrow$ m.mb.
(R4.1)	At $transmitter_i(m)$:	$MBV_i[i] \leftarrow$ m.mb.

According to the rules above, MBV_i maintains the p_i's knowledge of the MB values of all processes in g. Thus, any process $p_i \in g$ using its knowledge about delivered (MLDV) and received messages (MBV) can suspect losses of messages.

(P1.0) *Property*: If a message m, sent by a process $p_k \in g$, was not received by any $p_j \in g$ then there exist some processes $p_i \in g$ ($i \neq j$) which received m such that eventually $MLDV_i[j][k] < m.b < \min(MBV_i)$.

Proof: Consider a group $g = \{p_1, p_2, ..., p_n\}$. The proof will be in two parts.

---

5 The symbol $\leftarrow$ represents an attribution operation.

6 Although LDV_i and $MLDV_i[i]$ have the same meaning, we will use both to maintain compatibility with the original causal order protocol [Mac94, Mac95].

7 This definition corresponds to the block completion definition [Mac94]. However, we cannot use the same term because we have assumed different hypothesis (section III).

Part 1. $\underline{MLDV_i[j][k] < m.b}$: if a message m with block number m.b sent by p_k was not received by p_j then $LDV_j[k] < m.b$ and $k \neq j$. Due to the timesilence mechanism and as the LDV values timestamp any transmitted message (by R2.0), p_i will eventually receive a message from p_j (section III) making $MLDV_i[j]$ equal to LDV_j (by R3.0).

Part 2. $\underline{m.b < \min(MBV_i)}$: As every process pi always transmits messages (timesilence mechanism) and the Causal Block advances during transmission (by R1.0) $MBV_i[i]$ will also advance. As rule R4.0 advances $MBV_i[k]$ $(i \neq k)$, eventually $m.b < \min(MBV_i)$.

Based on the property above a process can compare its MLDV and min(MBV) values to suspect if other processes have missed any message. If so, the process can retransmit them. This is the basic idea of the recovery algorithm described below.

(R5.0) ***Retransmission Condition Rule***: any message $m_k \in BM_i$ $(m.s = p_k)$ such that $MLDV_i[j][k] < \min(m_k.bn) < \min(MBV_i)$ and $i \neq k \neq j$, can be retransmitted by $p_i \in g = \{p_1, p_2,...,p_n\}$.

As we can see the detection of missing messages is dependent on the activity of the group. That is, missing messages are only detected when processes receive a message informing the state of the process which missed them. Therefore, if the group is very active, missing messages are detected fast otherwise the recovery and detection can be slow. Thus the null message transmission procedure has a fundamental role in the performance of the recovery mechanism. However, if there is a high frequency of transmissions of null messages it can increase the network load. It is, therefore, necessary to balance time of recovery and timesilence mechanism.

B. The Algorithms

The figures 3 to 6 represent the algorithm of the proposed protocol. Each procedure is an independent thread which runs concurrently in a process and every process in a group runs the same algorithms in a symmetric approach. Fig.3 illustrates the detection and retransmission procedure. As soon as this thread detects a missing message (line 3) it sets a random timeout to send such a message (line 4). This timeout is an optimisation to avoid retransmission implosion because all processes in the same group can detect the same faults. This timeout can be cancelled before the retransmission (the lines 7 and 8 of Fig.4). With a suitable adjustment of this timeout the number of retransmissions can be decreased.

It is interesting to note that only the message with the smallest block number is retransmitted. The reason for this is that such a message happened before the others (causal relation). Therefore there is no guarantee that these messages have not been received (MLDV is a view of delivered messages). Once the first message is recovered a retransmission procedure will be restarted if another missing message is detected.

```
Procedure send_missing_message

1. Do it forever
2.     if min(MBV) has changed
3      ∀ m_k ∈ BM_i | R5.0 is valid
4.         after a random timeout send m with minimum
           block number
5. enddo
```

Fig.3 Message detection and retransmission procedure.

The receiving, transmitting and delivery procedures, are showed in the figures 4 to 6. Their algorithms, together with the procedure above implement the rules viewed in the earlier sections.

```
Procedure receiver
1. Do it forever
2.     receive m from the transport layer
3.   if there is any timeout to retransmit m then
       set a new timeout
4.   else
5.     if m.b > max(m′.b) | m′.s = m.s ∀ received m′
         then
6.         MLDV_i[m.s] ← m.ldv; MBV_i[m.s] ← m.mb
7.         if there is any timeout for any m′ and
             R5.0 is not valid then
8.             cancel the timeout
9.         if m ∉ BM_i then
10.            mark BM_i[m.b] with a ´+´; BM_i[m.b] ← m
11.            start delivery procedure.
12. enddo
```

Fig.4 Recovery procedure.

The receiving procedure receives messages from the transport layer, executes certain actions and starts the delivery procedure. First of all, in order to minimise the number of retransmissions, if there is any pending timeout for the received message a new timeout is set (line 3) and the received message is discarded. The timeout is only cancelled in line 8 if the retransmission condition rule is no longer valid. As messages do not arrive in FIFO order, only receiving of messages with the greatest block number can update MLDV (line 5). Finally received messages are put in BM and the delivery procedure is started as well. Due to message retransmissions, these final actions are performed if received messages are not yet present in BM.

The transmission and delivery procedures, Fig.5 and Fig.6, are simpler and very similar to the original causal protocol [Mac95].

```
Procedure transmitter

1. BC_i = BC_i + 1; m.b = BC_i
2. m.ldv ← LDV_i
3. MLDV_i[i][i] ← LDV_i[i] ← BC_i
4. BM_i ← m
5. multicast m through the multicast layer
```

Fig.5 Transmission procedure.

As we can see in these algorithms, the protocol can easily be modified to gain flexibility according to the semantic of the application. For instance, consider a distributed multimedia application. If a process discovers that a message does not make sense (i.e. its lifetime is expired) it can advance its LDV without delivering the message and the recovery procedure will not be started.

```
Procedure deliver
1. if ∃ m ∈ BM_i | m is not delivered Λ m.ldv ≥ LDV_i
then
2.     deliver m to p_i
3.     BC_i ← max{BC_i, m.b}
4.     MLDV_i[i][m.s] ← LDV_i[m.s] ← m.b
5.     signal deliver procedure recursively
```

Fig 6. Delivery procedure.

C. *Correctness*

In order to prove the correctness of the protocol we have to prove the theorem below:

(T1.0) Every missing message is detected, eventually recovered and delivered without violating causality.
Proof: [Mac94] has proved that any sent message will be eventually delivered by the original causal order protocol. The lemmas below prove the other parts of the theorem.

The lemmas L1.0 and L1.1 show the liveness property of the protocol, i.e., all missing messages are eventually detected and recovered. With regard to the safety property we will see that neither the message retransmission procedure nor the multicast layer assumptions have any effect on the message delivery order (L1.3). Finally lemma L1.2 shows that any detected missing message is a valid message. As stated before we assume that a membership and a timesilence services exit in order to guarantee that functioning members always have a mutual consistent view of the group membership.

(L1.0) The existence of a missing message is eventually detected (*liveness*).
Proof: The detection of a missing message m by a process $p_i \in g = \{p_1, p_2, ..., p_n\}$ happens when $MLDV_i[j][k] < \min(m.bn) < \min(MBV_i)$, supposing that m was originally sent by $p_k \in g$ and was not received by some $p_j \in g$. According to property P1.0 the relation above is eventually true, and so the detection is also eventually true. In the worst case there is only one detecting process, the original sender process ($p_k = p_i$).

(L1.1) *Lemma:* A missing message is eventually received and delivered (*liveness).*
Proof: Recalling our assumptions, any transmitted message arrive at its destination after a finite number of retransmissions. As a detecting process retransmits a given message after it detects the missing (Fig.3), at least one of the retransmissions will arrive successfully, otherwise the group is not a connected one.

(L1.2) *Lemma*: A detected missing message m is a valid message (was originally sent by a process belonging to g) (*validity*).
Proof: Consider a group $g = \{p_1, p_2,...,p_n\}$ and suppose that m was detected as a missing message by $p_i \in g$. Thus, $MLDV_i[j][k] < m.bn < \min(MBV_i)$ (by R5.0) for some $p_j \in g$ and $p_k \in g$. This means that p_i has received m from p_k and delivered it. As m was delivered by p_i, m is a valid message.

(L1.3) *Lemma:* A received message is delivered without violating causality (*safaty*).
Proof: Consider a group $g = \{p_1, p_2,...,p_n\}$. Suppose by absurd that a message m, such that $m' \rightarrow m$, was delivered by $p_j \in g$ and m′ was not yet delivered by p_j. Now suppose without losing of generality that m′ was sent by $p_i \in g$ with $i \neq j$. If p_j delivered m, $LDV_j \geq m.ldv$ and so $LDV_j[i] \geq m.ldv[i]$. As $m' \rightarrow m$, $m'.ldv[i] \leq m.ldv[i]$. By rule

R2.2 it is known that $LDV_j[i]$ only advances when p_j delivers messages sent by p_i. Therefore m′ must have been delivered by p_j before m (contradiction).

VI. IMPLEMENTATION

The protocol proposed was implemented in C++ over a network of Unix workstations. An experiment with a group of three processes was configured where the timesilence mechanism was set to transmit messages at intervals of 500ms for all processes. One of these processes was chosen as the sender and the others the receivers. The receiver processes were also able to retransmit messages when they detected loss of messages.

Messages with block numbers between 10 and 15 were dropped at one of the receivers. The figure 7 shows when these missing messages were received by this receiver. The first line of the figure is the BC values of such a receiver[8]. The line named *Rcv* represents which messages were recovered (indicted by their block numbers). Thus we can see that message 10 was recovered when the BC was 17, i.e., after receiving message 16. It interesting to note that message 10 was recovered twice. This is because both the transmitter and the other receiver retransmitted it.

BC	17	18	19	20	21	22	23	24	25	26	27
Rcv	10	10	11			12	13		14	15	

Fig.7. Illustration of the message recovery protocol for a group of three processes.

VII. CONCLUSIONS

The missing message problem has an important influence on distributed protocol implementation. The performance and complexity of distributed protocols depend on the decision of where and how to detect and recover missing messages. To our best knowledge, the protocols developed to date [MRTW97, PSLB97, YGS95, BSS91] were designed to recover missing messages based on information at the receiving time. As we have discussed earlier, recovery based on this kind of information can be too restrictive for certain types of applications. We have contributed to the area by proposing and implementing a new message recovery mechanism based on stability defined at the delivery time. According to the semantic of application, the protocol can be configured to ignore expired messages or discard already received messages by properly manipulating the MLDV vector. This can be very useful for soft real-time applications which have lifetimes or priorities associated to transmitted messages.

Moreover, our protocol was designed as a symmetric recovery procedure in which any process of the group can detect and retransmit missing messages. This characteristic contrasts with the majority of message recovery protocols proposed to date where the responsibility of the message retransmissions generally lies only with the sender (unique point of failure). Although SRM [FJMLZ95] is also symmetric, it does not take advantage of message delivery information at the group communication service level (e.g. causal order protocol level).

[8] As our objective is only to illustrate the behaviour of the recovery procedure instead of to quantify its performance, the measurements are based on logical time.

ACKNOWLEDGEMENTS

This work was partially supported by LOCUS/ProTeM-CC project (Phase III) and the second author is also supported by CNPq Grant 300013/87-6 (BCG Research Project).

REFERENCES

[ACM95] G. A. Alvarez, F. Cristian and S. Mishra. *On-Demand Asynchronous Atomic Broadcast*. 5th IFIP Working Conf. on Dependable Computing for Critical Applications, 1995.

[BJ87] K. Birman, T. A. Joseph. *Reliable Communication in the Presence of Failures*. ACM TDCS, (5,1), pp. 47-76, 1987.

[Bar98] Barcellos, A. M. P. *PRMP: A Scaleable Polling-based Reliable Multicast Protocol*. Ph. D. Thesis. Newcastle upon Tyne, September 1998.

[BMR94] R. Baldoni, A. Mostefaoui and M. Raynal, *Causal Deliveries in Unreliable Networks with Real-Time Delivery Constraints.* Technical Report 2427, INRIA. December 1994.

[BSS91] K. Birman, A. Schiper and P. Stephenson. *Lighweight Causal and Atomic Group Multicast*. ACM Transaction on Computing Systems, (9,3), pp. 272-314, August 1991.

[CV95] F. J. N. Cosquer and P. Veríssimo. *The Impact of Group Communication Paradigms on Groupware Support*. Proc. of the 5th Workshop on Future Trends of Distributed Computing Systems. Korea, 1995.

[CZ85] D. Cheriton, W. Zwaenepoel. *Distributed Process Groups in V Kernel*. ACM TOCS, (3,2), pp. 97-107, 1985.

[EMS95] P. D. Ezhilchelvan, R. A Macêdo and S. K. Shrivastava. *Newtop: A Fault-Tolerant Group Communication Protocol.* Proc. of the 15th International Conference on Distributed Computing Systems, IEEE Computer Society, pp. 296-306, Canadá , June 1995.

[FJMLZ95] S. Floyd, V. Jacobson, S. McCanne, C. Liu, L. Zhang. *A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing*. ACM SIGCOMM'95. Cambridge, USA, August 1995.

[GM98] F. G. P. Greve and R. J. A. Macêdo. *The BCG Membership Service Performance Analysis*. In Proc. of XVI Simpósio Brasileiro de Redes de Computadores. Rio de Janeiro, May 1998.

[Lam78] L. Lamport. *Time, Clocks and Ordering of Events in a Distributed System*. CACM, (21, 7), pp. 558-265, July 1978,.

[LM97] G. M. de A. Lima and R. J. A. Macêdo. *Avaliação de Desempenho do protocolo BCGcausal* Technical Report, RTI-003/97. November 1997.

[LM99] G. M. de A. Lima and R. J. A. Macêdo. *Avaliação de Desempenho de Protocolos de Ordenação Causal para Comunicação em Grupo*. XXV Conferencia Latinoamericana de Informática, CLEI99, 1999.

[Mac94] R. J. de A. Macêdo. *Fault Tolerant Group Communication Protocols for Asynchronous Systems*. Ph.D. Thesis, Computing Science Department, University of Newcastle upon Tyne, UK, 1994.

[Mac95] R. J. de A. Macêdo. *Causal Order Protocols for Group Communication*. SBRC95, Belo Horizonte-MG, pp. 265-283, May 1995.

[MES96] R. J. A. Macêdo and P. D. Ezhilchelvan, S. K. Shrivastava. *Buffer Overflow Avoidance Techniques for Groups Communication Protocols*. SBRC96, pp. 633-652, Fortaleza-CE, May 1996.

[MES93] R. J. A. Macêdo and P. D. Ezhilchelvan, S. K. Shrivastava. Modeling *Group Communication Using Causal Blocks*. 5th European Workshop on Dependable Computing, Lisbon, February, 1993.

[MRTW97] K. Miller, K. Robertson, A. Tweedly and M. White. *Starbus Multicast File Transfer Protocol (MFTP) Specification*. Internet draft (expired), January 1997.

[PBS89] L. L. Peterson, N. C. Buchholz and R. D. Schlichting. *Preserving and Using Context Information in Interprocess Comunication*. ACM Transaction on Computing Systems, (7,3), pp. 217-246, August 1989.

[PSLB97] S. Paul, K. Sabnani, J. Lin and S. Bhattacharryya. *Reliable Multicast Transport Protocol (RMTP)*. IEEE Journal on Selected Areas in Communications, Vol. 13, No. 3, April 1997.

[RBM96] R. Renesse, K. Birman and S. Maffeis. *HORUS, A Flexible Group Communication System*. Communication of the ACM, April 1996.

[VRV93] P. Veríssimo, L. Rodrigues and Werner Vogels. *Group Orientation: a Paradigm for Modern Distributed Systems*. ESPRIT Basic Research Project First Year Report, volume 1, October 1993.

[YGS95] R. Yavatkar, J. Griffioen and M. Sudan. *A Reliable Dissemination for Interactive Collaborative Applications*. ACM Multimedia, 1995.