Causally Ordering Broadcast Protocol

Akihito NAKAMURA and Makoto TAKIZAWA
Dept. of Computers and Systems Engineering
Tokyo Denki University
Ishizaka, Hatoyama, Hiki-gun, Saitama 350-03, JAPAN
E-mail {naka, taki}@takilab.k.dendai.ac.jp

Abstract

The distributed applications require group communications among multiple entities. In the group communication, it is important to discuss in what order each entity in the group can receive data units. In order to realize fault-tolerant systems, the same events have to occur in the same order in each entity. The ordering among the events is known as a causal order. This paper presents a reliable causally ordering broadcast (CO) protocol which provides the same causal ordering of data units for all the entities in the group. In the CO protocol, the data units received are causally ordered by using the sequence numbers of the data units. The CO protocol is based on the fully distributed control scheme, i.e. no master controller, and uses high-speed networks where each entity may fail to receive data units due to the buffer overrun. Furthermore, the CO protocol provides asynchronous data transmission for multiple entities in the group.

1 Introduction

In distributed applications like CSCW (computer-supported cooperative work) [5], reliable group communication service for multiple entities is required. In this paper, a group of entities is referred to as a *cluster* [14, 15]. Data units, i.e. messages, exchanged among the entities are protocol data units (PDUs). If the communication system provides the atomic, ordered, and loss-less delivery of PDUs in the cluster, the applications can be easily realized. In the group communication, each entity can deliver PDUs to the entities and can receive PDUs destined to it from every entity in the cluster. In the high-speed networks [1], the transmission speed is faster than the processing speed of each entity while the data transmission on the network is almost error-free. This means that each entity may fail to receive PDUs. Thus, the PDU loss is considered as the most failure in the networks.

In addition to providing the atomic delivery of PDUs to all the entities in the presence of the PDU loss, a group communication protocol has to provide the application entities with the following kinds of services with respect to the orders in which PDUs are delivered to the application entities:

(1) locally ordering broadcast (LO) service,
(2) causally ordering broadcast (CO) service, and
(3) totally ordering broadcast (TO) service.

In the LO service, PDUs from each entity are received in the same order as they are sent. The PO protocol [16] provides the LO service. That is, if an entity broadcasts two PDUs p and q in this order, every destination entity receives q after p. In the TO service, all the destinations receive PDUs in the same order in addition to the sending order. Most reliable broadcast protocols [3, 4, 6, 7, 9, 10, 14, 15] provide the TO service. In the CO service, PDUs received are ordered by the *happened-before* relation [8]. If p is sent *logically* before q, p is delivered to every destination before q. In the LO and TO services, it is discussed in what order each entity can receive PDUs. In the CO service, it is in addition taken into account how each entity looks at the order of sending and receipt events in another entity. The CO service is required in distributed applications like fault-tolerant systems and CSCW [5]. ISIS [3] supports the CBCAST protocol to provide the CO service for fault-tolerant applications. The CBCAST protocol is implemented on the reliable transport service where every PDU is guaranteed to be delivered to the destination. Each entity assigns a *virtual clock time* for each PDU when it sends the PDU, and PDUs received are causally ordered by the virtual clocks. The virtual clocks have to be synchronized among the entities in a similar way to [8]. Each sender decides on the atomic and ordered receipt of PDUs among all the destinations. The TO protocol [14, 15] provides the CO service by using a one-channel network like Ethernet where each entity receives PDUs in the same order while it may fail to receive some of them.

In this paper, we present a CO protocol which provides the CO service for application entities in a cluster. The CO protocol is implemented on a high-speed network [1], i.e. in the presence of the PDU loss while ISIS assumes to use the reliable network, i.e. no PDU loss. When the PDU loss is detected, only the lost PDUs are retransmitted by using a selective retransmission scheme. The CO protocol uses the sequence numbers of PDUs assigned by each source entity to causally order the PDUs and to detect the PDU loss while ISIS requires more computation to synchronize the virtual clocks. The CO protocol is fully dis-

1063-6927/94 $03.00 © 1994 IEEE

tributed, i.e. every entity decides on the atomic and ordered receipt of PDUs while a sender of each PDU plays a role of controller in ISIS. In addition, the CO protocol provides asynchronous data transmission in the presence of multiple senders.

This paper is organized as follows. In section 2, we model the communication system. In section 3, we present a concept of atomic receipt of PDUs in a cluster. In section 4, we present a data transmission procedure of the CO protocol. In section 5, we discuss the evaluation of the CO protocol.

2 Model

In this section, we present a model of group communication service for multiple entities in the cluster.

2.1 System model

A communication system is composed of *application*, *system*, and *network* layers [Figure 1]. A cluster C [14,15] is a set of n (≥ 2) system *service access points* (SAPs), i.e. $\{S_1,...,S_n\}$. Each application entity A_i takes some communication service through S_i which is supported by a system entity E_i ($i = 1,...,n$). The system entities $E_1,...,E_n$ cooperate with each other by a *system* protocol to support some broadcast service for C by using the underlying network service. C is referred to as *supported* by $E_1,...,E_n$ (written as $C = \langle E_1,...,E_n \rangle$) and *support* $A_1,...,A_n$. The network layer provides a reliable high-speed data transmission service [1] for the system layer through the network SAPs. Since the transmission speed of the network layer is faster than the processing speed of the system entity, the system entity may fail to receive PDUs due to the buffer overrun.

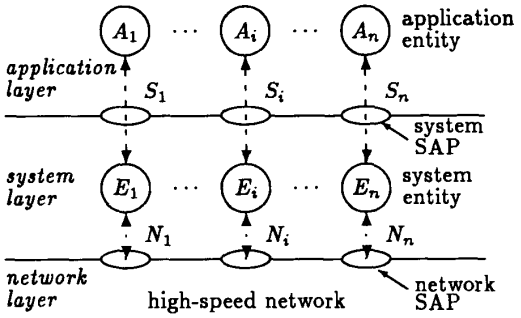


Figure 1: System model

2.2 Ordering properties

We model a communication service provided by C as a set of logs. A log L is a sequence of PDUs, denoted as $< p_1 \ldots p_m >$. Here, p_1 and p_m are the top and the last of L, denoted by $top(L)$ and $last(L)$, respectively. Each E_i has a sending log SL_i and a receipt log RL_i, which are sequences of PDUs sent and received by E_i, respectively ($i = 1,...,n$).

For each E_i and PDU p, $s_i[p]$ and $r_i[p]$ denote the sending and receipt events of p in E_i, respectively. We assume that there exists exactly one sending event $s_j[p]$ for every receipt event $r_i[p]$. The following partially ordered relation, i.e. *happened-before* relation $\rightarrow$ on the events is defined in [8].

[Definition] For every pair of events e_1 and e_2, $e_1 \rightarrow e_2$ iff

(1) e_1 happens before e_2 in E_i,

(2) for some (not necessarily different) E_i and E_j, there exists some PDU p such that $e_1 = s_i[p]$ and $e_2 = r_j[p]$, or

(3) for some event e_3, $e_1 \rightarrow e_3$ and $e_3 \rightarrow e_2$. ■

For every pair of PDUs p and q in RL_i, p *precedes* q *in* RL_i ($p \rightarrow_{RL_i} q$) if $r_i[p] \rightarrow r_i[q]$. That is, E_i receives p before q. p *precedes* q *in* SL_i ($p \rightarrow_{SL_i} q$) if $s_i[p] \rightarrow s_i[q]$. That is, E_i sends p before q.

We define the following properties of the receipt log RL_i by using the happened-before relation. Here, $p.DST$ denotes the set of destination entities of p.

[Definition]

(1) RL_i is referred to as *information-preserved* iff for every E_j and p in SL_j, p is in RL_i if $E_i \in p.DST$.

(2) RL_i is referred to as *local-order-preserved* iff for every pair of p and q in RL_i, $p \rightarrow_{RL_i} q$ if p and q are sent by E_j, i.e. $p \rightarrow_{SL_j} q$, and $E_i \in p.DST \cap q.DST$. □

If RL_i is information-preserved, E_i receives all the PDUs destined to E_i. If RL_i is local-order-preserved, E_i receives PDUs destined to E_i from every E_j in the sending order.

Next, we define a causal relation among the PDUs according to [3].

[Definition] For every pair of PDUs p and q, p *causality-precedes* q (written as $p \prec q$) iff $s_i[p] \rightarrow s_j[q]$. □

The causality-precedence relation $\prec$ is transitive but not symmetric. The relation $\prec$ among PDUs reflects the happened-before relation $\rightarrow$ among the sending events of the PDUs. Intuitively speaking, $p \prec q$ means that p is sent before q in C. p and q are referred to as *causality-coincident* (written as $p \parallel q$) if neither $p \prec q$ nor $q \prec p$. $p \preceq q$ iff $p \prec q$ or $p \parallel q$.

[Definition] A receipt log RL_i is referred to as *causality-preserved* iff for every pair of p and q in RL_i, $p \rightarrow_{RL_i} q$ if $p \prec q$. ■

It is straightforward that RL_i is local-order-preserved if RL_i is causality-preserved.

In Figure 2, $RL_k = < g\ p\ q\ ]$ is causality-preserved. That is, $g \prec p \prec q$ since E_h sends q after receiving p and E_k receives p before receiving q, i.e. $s_j[g] \rightarrow s_j[p] \rightarrow s_h[q]$ and $r_k[g] \rightarrow r_k[p] \rightarrow r_k[q]$. On the other hand, if E_k receives q before p, $RL_k' = < g\ q\ p\ ]$ is not causality-preserved (but local-order-preserved).

2.3 Group communication service

The service provided by C is *reliable* if every RL_i in C is information-preserved and local-order-preserved. Otherwise, the service is *less-reliable*. That is, some entity may fail to receive PDUs or may receive them not in the sending order.

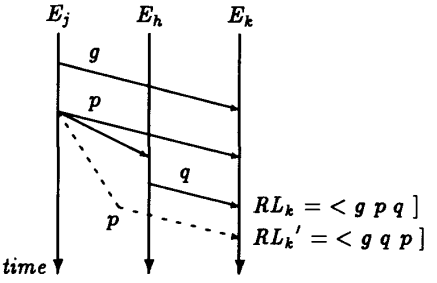


Figure 2: Causality-preserving receipt

[Definition]

(1) A service of C is a *multi-channel* (MC) service iff every RL_i in C is local-order-preserved.

(2) A service of C is a *causally ordering broadcast* (CO) one iff every RL_i in C is information-preserved and causality-preserved. □

The MC service is a model of high-speed communication links where some E_i may not receive some PDUs due to the buffer overrun, i.e. RL_i may not be information-preserved. The MC service can be provided by a system in which computers are fully connected by high-speed communication links.

The CO service is a kind of reliable broadcast service. By using the CO service, the happened-before relation $\rightarrow$ among the sending events of PDUs is preserved in every destination entity of the PDUs. That is, if $s_j[p] \rightarrow s_k[q]$ for p and q, $r_i[p] \rightarrow r_i[q]$ in every common destination E_i of p and q.

3 Atomic Receipt Concept

In this section, we consider the atomic receipt concept in a cluster $C = \langle E_1,...,E_n \rangle$ in a distributed control scheme. We assume that every PDU from each E_j carries the *receipt confirmation* of PDUs which E_j has received already. That is, if E_i receives q from E_j, E_i knows that E_j receives every PDU p where $r_j[p] \rightarrow s_j[q]$. Here, let p and q be PDUs sent by E_k and E_j, respectively. q is referred to as *pre-acknowledge* p for E_j in E_i ($p \Rightarrow_{ji} q$) iff $s_k[p] \rightarrow r_i[p]$ and $s_k[p] \rightarrow r_j[p] \rightarrow s_j[q] \rightarrow r_i[q]$. In Figure 3, E_2 receives a sent by E_1. On receipt of c, E_3 knows that E_2 has received a. Here, $s_1[a] \rightarrow r_2[a] \rightarrow s_2[c] \rightarrow r_3[c]$, i.e. c pre-acknowledges a for E_2 in E_3 ($a \Rightarrow_{23} c$).

There are three criteria levels [14,15] of E_i's atomic receipt for every PDU p in C if the cooperation among the entities is coordinated in the distributed way:

(1) *Acceptance*: E_i receives p.

(2) *Pre-acknowledgment*: E_i knows that every destination of p has accepted p. That is, for every E_j, there exists q such that $p \Rightarrow_{ji} q$.

(3) *Acknowledgment*: E_i knows that p has been pre-acknowledged by every destination of p. That is, for every E_j and E_h, and q where $p \Rightarrow_{hj} q$, there exists g such that $q \Rightarrow_{ji} g$.

The acknowledgment of p by E_i means that E_i knows that every destination of p has known that every destination of p had received p. Even if p is pre-acknowledged in E_i, E_i cannot decide if p is atomically received by all the destination entities. When p is acknowledged in E_i, E_i knows that p is pre-acknowledged by every destination. Also, another E_j knows that p is at least accepted by every destination.

Let us consider a cluster $C = \langle E_1, E_2, E_3, E_4 \rangle$ shown in Figure 3. E_1 broadcasts a in C. Since $a \Rightarrow_{13} b$, $a \Rightarrow_{23} c$, $a \Rightarrow_{33} d$, and $a \Rightarrow_{43} e$, a is pre-acknowledged in E_3 on acceptance of e. On acceptance of f, E_4 knows that b, c, d, and e are accepted by E_3. Suppose that every E_i also receives b, c, d, and e, and then sends g_i ($i = 1, 2, 3, 4$), i.e. $g_3 = f$. If E_4 receives g_i from every E_i, a is acknowledged in E_4.

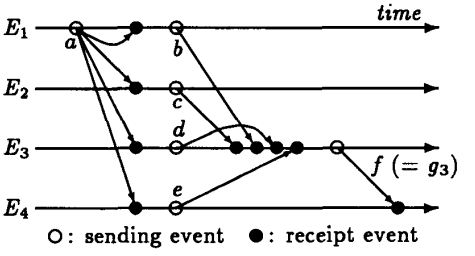


Figure 3: Pre-acknowledgment and acknowledgment

4 Causally Ordering Broadcast Protocol

In this section, we present a causally ordering broadcast (CO) protocol which provides the CO service for $C = \langle E_1,...,E_n \rangle$ by using the MC service. Here, we assume that p is destined to all the entities in C. That is, we do not consider selective group communication [11] in this paper.

4.1 Variables

The PDU format in the CO protocol is shown in Figure 4. $p.F$ denotes a field F of a PDU p. For example, the SEQ field of p is denoted as $p.SEQ$.

- $p.CID$ = cluster identifier of C.
- $p.SRC$ = source entity of p, i.e. the entity who sends p, say E_i.
- $p.SEQ$ = sequence number of p.
- $p.ACK_j$ = sequence number of a PDU which E_i expects to receive next from E_j ($j = 1,...,n$).
- $p.BUF$ = number of available buffer units in E_i.

$p.ACK_j$ means that E_i has accepted every q from E_j where $q.SEQ < p.ACK_j$. Here, $p.ACK$ denotes a tuple $\langle p.ACK_1,...,p.ACK_n \rangle$.

Each E_i has the following variables:

- SEQ = sequence number of a PDU which E_i expects to broadcast next.
- REQ_j = sequence number of a PDU which E_i expects to receive next from E_j ($j = 1,...,n$).

CID	SRC	SEQ	$ACK = \langle ACK_1,...,ACK_n \rangle$	BUF	DATA

Figure 4: PDU format

CID	SRC	LSRC	LSEQ	$ACK = \langle ACK_1,...,ACK_n \rangle$	BUF

Figure 5: RET PDU format

- AL_{kj} = sequence number of a PDU which E_i knows that E_j expects to receive next from E_k (j, k = 1,...,n).
- BUF_j = number of available buffer units in E_j which E_i knows of (j = 1,...,n).

Here, $minAL_k$ and $minPAL_k$ denote the minimum among $AL_{k1},...,AL_{kn}$ and $PAL_{k1},...,PAL_{kn}$, respectively (k = 1,...,n).

4.2 Transmission and acceptance

Each application entity A_i submits a data transmission (DT) request r at the system SAP S_i. When E_i takes r from S_i, E_i checks the following flow condition. Here, W and H ($\geq W$) are constants. W gives the window size.

[Flow condition] $minAL_i \leq SEQ < minAL_i + min(W, minBUF / (H \times n))$. □

If E_i transmits a PDU each time E_i receives a PDU, $O(n^2)$ PDUs are transmitted in C. In order to reduce the number of PDUs transmitted, E_i transmits a PDU after E_i receives at least one PDU from each entity or after some time units, i.e. *deferred confirmation*. By this method, $O(n)$ PDUs are transmitted [11, 13]. Hence, $minBUF$ is divided by $O(n)$ in the flow condition. If the flow condition holds, E_i creates a PDU p for the DT request r and broadcasts it by the following transmission action. Here, an operation $enqueue(L, p)$ means that p is put into the tail of a log L. $broadcast(p)$ means that a data transmission request of p is issued at the network SAP.

[Transmission action of E_i] {
$p.SEQ := SEQ$; $SEQ := SEQ + 1$;
$p.ACK_j := REQ_j$ (j = 1,...,n);
$p.BUF$:= available buffer size in E_i;
$enqueue(SL_i, p)$; $broadcast(p)$; } □

When E_i receives p ($p.SRC = E_j$), E_i checks the following acceptance condition.

[Acceptance (ACC) condition for p] $p.SEQ = REQ_j$ (where $p.SRC = E_j$). □

If p satisfies the ACC condition, E_i performs the following acceptance action. E_i has a receipt log RRL_{ij} to store PDUs accepted from each E_j (j = 1,...,n).

[Acceptance (ACC) action of E_i] {
$REQ_j := p.SEQ + 1$;
$AL_{kj} := p.ACK_k$ (k = 1,...,n);
$BUF_j := p.BUF$;
$enqueue(RRL_{ij}, p)$; } □

On acceptance of p, E_i does not yet know if another entity has also received p. Hence, E_i does not deliver p to the application entity.

4.3 Failure detection and recovery

In the MC service, each E_i may lose some PDU g due to the buffer overrun. E_i can detect the loss of g by checking the following failure condition each time E_i receives a PDU.

[Failure (F) condition]

(1) On receipt of p from E_j, if $REQ_j < p.SEQ_i$, then E_i has not received g from E_j such that $REQ_j \leq g.SEQ_i < p.SEQ_i$ (j = 1,...,n).

(2) On receipt of q from E_k, for some j ($\neq k$), if $REQ_j < q.ACK_j$, then E_i has not received g from E_j such that $REQ_j \leq g.SEQ < q.ACK_j$. □

If E_i detects the loss of g from E_j, E_i requests E_j to retransmit g by sending an RET PDU r. The format of the RET PDU is shown in Figure 5. If E_j receives r, E_j rebroadcasts g.

[Retransmission action]

(1) If E_i detects a lost PDU g from E_j, E_i broadcasts an RET PDU r such that $r.ACK_k = REQ_k$ (k = 1,...,n), $r.LSRC = E_j$, and $r.LSEQ = p.SEQ$ for the F condition (1) or $r.LSEQ = q.ACK_j$ for the F condition (2).

(2) If E_j receives r from E_i, E_j rebroadcasts g such that $r.ACK_j \leq g.SEQ \leq r.LSEQ$. □

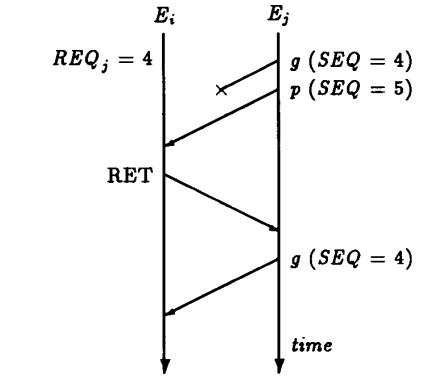
In Figure 6 (a), E_i detects the loss of g on receipt of p, i.e. REQ_j (= 4) $\neq p.SEQ$ (= 5) by the F condition (1). On the other hand, E_i detects the loss of g on receipt of q from E_k in Figure 6 (b). Since REQ_j (= 4) $\neq q.ACK_j$ (= 5), E_i detects the loss of g which E_k has accepted by the F condition (2). In both cases, E_i requests E_j to rebroadcast g by sending an RET PDU r whose $ACK_j = REQ_j$ (= 4), $LSRC = E_j$, and $LSEQ$ = 5.

4.4 Pre-acknowledgment procedure

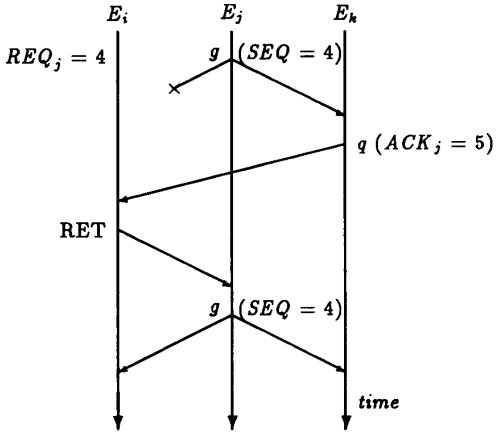
On receipt of a PDU q from E_k, each receipt confirmation $q.ACK_j$ for E_j is stored in AL_{jk} (j = 1,...,n) by the acceptance action. E_i knows that E_k has accepted a PDU p from E_j such that $p.SEQ < AL_{jk}$. The minimum $minAL_j$ among $AL_{j1},...,AL_{jn}$ means that E_i can know that every entity has accepted p from E_j such that $p.SEQ < minAL_j$. Since E_i knows that every entity has accepted p from E_j, p is *pre-acknowledged* in E_i if the following PACK condition holds for p.

[Pre-acknowledgment (PACK) condition] $p.SEQ < minAL_j$ (where $p.SRC = E_j$). □

Next problem is how each entity perceives the



(a) Failure condition (1)



(b) Failure condition (2)

Figure 6: Failure detection and recovery

causality-precedence relation $\prec$ among PDUs received.

[Theorem 4.1] Let p be a PDU sent by E_j. Let q and r be PDUs.

(1) If $p.SRC = q.SRC$, $p \prec q$ iff $p.SEQ < q.SEQ$.
(2) If $p.SRC \neq q.SRC$, $p \prec q$ iff $p.SEQ < q.ACK_j$.

[Proof]

(1) Suppose that $p.SRC = q.SRC$ and $p.SEQ < q.SEQ$. $p.SEQ < q.SEQ$ means that $s_j[p] \rightarrow s_j[q]$, i.e. $p \prec q$. If $p.SEQ < q.SEQ$, $p \prec q$.
(2) Suppose that E_j sends p to E_h and E_k, E_h sends q after receiving q, and E_k receives q after p as shown in Figure 2. First, suppose that $p \prec q$. $p \prec q$ means that $r_h[p] \rightarrow s_h[q]$. On acceptance of p, E_h increments REQ_j by one, i.e. $p.SEQ < REQ_j$. Hence, for p and q such that $r_h[p] \rightarrow s_h[q]$, $p.SEQ < q.ACK_j$ since $REQ_j < q.ACK_j$. Next, suppose that $p.SEQ < q.ACK_j$. E_h sends q after receiving some r from E_j such that $p.SEQ < r.SEQ$. That is, $s_j[p] \rightarrow s_j[r] \rightarrow r_h[r] \rightarrow s_h[q]$. Hence, $p \prec q$. □

This theorem means that each entity can decide if $p \prec q$ by using the sequence numbers. That is, the PDUs can be ordered in $\prec$ while the PDU loss can be detected. By using the virtual clock [2,3], the PDU loss cannot be detected.

[Lemma 4.2] Let p be a PDU sent by E_j and q be a PDU. If $p \prec q$,

(1) $p.ACK_i \leq q.ACK_i$ $(i = 1,...,n)$ if $p.SRC = q.SRC$, and
(2) $p.ACK_j < q.ACK_j$ and $p.ACK_i \leq q.ACK_i$ $(i = 1,...,n,\ i \neq j)$ if $p.SRC \neq q.SRC$.

[Proof]

(1) Suppose that $p.SRC = q.SRC$. Since $p \rightarrow_{SL_j} q$, it is clear that $p.ACK_i \leq q.ACK_i$ $(i = 1,...,n)$.
(2) Next, suppose that $p.SRC \neq q.SRC$ $(= E_k)$. $p.ACK_j \leq p.SEQ$. From Theorem 4.1, $p.SEQ < q.ACK_j$. Hence, $p.ACK_j < q.ACK_j$.
Here, suppose that $p.ACK_h > q.ACK_h$ for some h. This means that E_j receives some PDU a from E_h such that $a.SEQ = p.ACK_h - 1$ and E_k receives some b from E_h such that $b.SEQ = q.ACK_h - 1$. From the assumption, $a.SEQ > b.SEQ$, i.e. $b \prec a$. Since E_k sends q before receiving a, $b \prec q \prec a \prec p$. It contradicts. Hence, $p.ACK_i \leq q.ACK_i$ for $i = 1,...,n$. □

Even if $p \prec q$ holds from Theorem 4.1, there is some PDU lost as presented in Figure 6 if Lemma 4.2 does not hold.

We define a causality-preserved insertion (CPI) operation $L \lhd p$ by which p is inserted to a log L while keeping L causality-preserved. $L \lhd p$ is implemented as follows. Here, it is decided if $p \prec q$ holds for every p and q by using the sequence numbers as presented in Theorem 4.1.

[CPI operation $L \lhd p$]

(1) If L is empty, p is appended to L.
(2) For every q in PRL_i,
(2-1) if $p \prec q$, p is appended to the top of L,
(2-2) if $q \prec p$, p is appended to the tail of L,
(2-3) if $q \parallel p$, p is appended to the tail of L.
(3) Otherwise, p is inserted between q_1 and q_2 such that $q_1 \preceq p \prec q_2$ and no q in L where $q_1 \preceq q \prec q_2$. □

Each E_i has one receipt sublog PRL_i to store the PDUs pre-acknowledged in the causality-precedence order $\prec$. If p is pre-acknowledged in RRL_{ij}, p is moved to PRL_i according to the following PACK action.

[Pre-acknowledgment (PACK) action]

```
For (j = 1,...,n) {
  if (p = top(RRL_ij) satisfies the PACK condition) {
    p := dequeue(RRL_ij);
    PAL_kj := p.ACK_k (k = 1,...,n);
    PRL_i ◁ p;
  }
} □
```

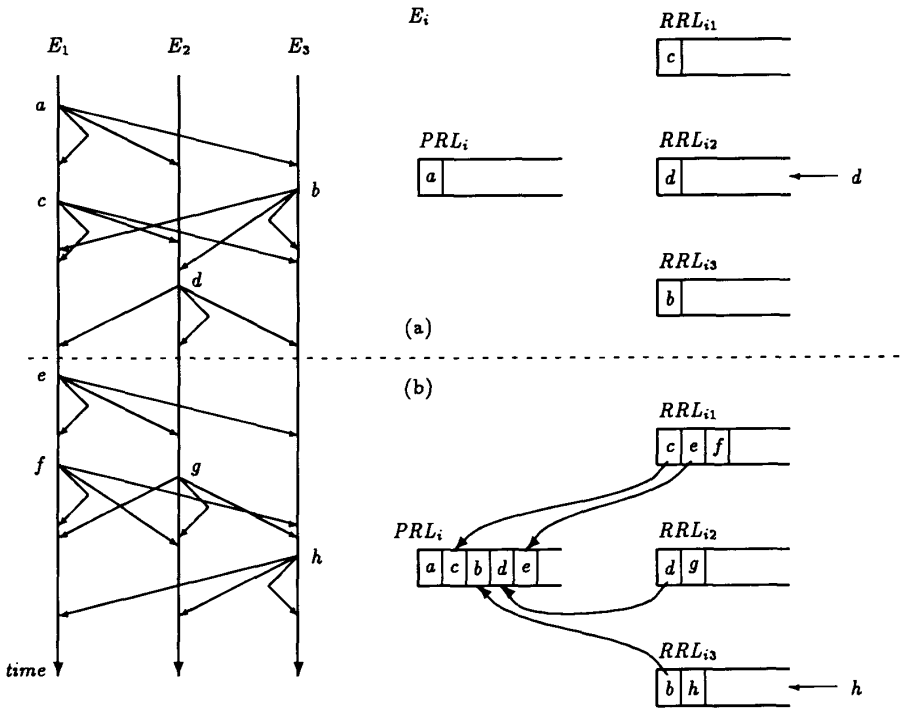


Figure 7: Pre-acknowledgment and the CPI operation in E_i

[Example 4.1] Let us consider a cluster $C = \langle E_1, E_2, E_3 \rangle$ shown in Figure 7. Suppose that initially $REQ_i = 1$ for each E_i in every entity (i = 1, 2, 3). Table 1 shows SEQ and ACK ($= \langle ACK_1, ACK_2, ACK_3 \rangle$) fields of each PDU used in Figure 7. For example, $d.SEQ = 1$ and $d.ACK = \langle 3, 1, 2 \rangle$ because E_2 has accepted c ($c.SEQ = 2$) from E_1 and b ($b.SEQ = 1$) from E_3 before E_2 sends d. On acceptance of d, $AL_{12} := d.ACK_1 = 3$, $AL_{22} := d.ACK_2 = 1$, $AL_{32} := d.ACK_3 = 2$. REQ and AL are given as follows:

$$REQ = \langle 3, 2, 2 \rangle, \quad AL = \begin{bmatrix} 2 & 3 & 2 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}.$$

Now, a is pre-acknowledged by every E_i because $a.SEQ$ (= 1) $< minAL_1$ ($min\{AL_{11}, AL_{12}, AL_{13}\} = 2$). Here, each E_i has PRL_i and RRL_{ij} (j = 1, 2, 3) as shown in Figure 7 (a). When h is accepted, REQ and AL are changed as follows:

$$REQ = \langle 5, 3, 3 \rangle, \quad AL = \begin{bmatrix} 4 & 4 & 5 \\ 2 & 2 & 3 \\ 2 & 2 & 2 \end{bmatrix}.$$

Here, four PDUs b, c, d, and e are pre-acknowledged. Hence, they are moved to PRL_i. First, c and e are appended to the tail of PRL_i by the CPI operation

PDU	SEQ	ACK
a	1	$\langle 1, 1, 1 \rangle$
b	1	$\langle 2, 1, 1 \rangle$
c	2	$\langle 2, 1, 1 \rangle$
d	1	$\langle 3, 1, 2 \rangle$
e	3	$\langle 3, 2, 2 \rangle$
f	4	$\langle 4, 2, 2 \rangle$
g	2	$\langle 4, 2, 2 \rangle$
h	2	$\langle 5, 3, 2 \rangle$

Table 1: SEQ and ACK fields

$\lhd$ ($PRL_i = <$ a c e]) because $a \prec c$ and $c \prec e$, i.e. $a.SRC = c.SRC = e.SRC = E_1$ and $a.SEQ < c.SEQ < e.SEQ$. Secondly, d is moved to PRL_i. Here, $c \prec d$, because $c.SEQ < d.ACK_1$. As presented in Lemma 4.2, $c.ACK_i < d.ACK_i$ (i = 1, 2, 3). $d \prec e$ because $d.SEQ < e.ACK_2$. Thus, d is inserted between c and e in PRL_i (i = 1, 2, 3). Then, b is inserted between c and d because $c \preceq b \prec d$. □

[Proposition 4.3] Let p and q be PDUs where $p.SRC = E_j$, $q.SRC = E_k$, and $p \prec q$. If q is pre-acknowledged in E_i, p is pre-acknowledged in E_i.

[Proof] Suppose that p is not pre-acknowledged

and q is pre-acknowledged in E_i. Since q is pre-acknowledged, E_i has accepted some g_h from every E_h where $q.SEQ < g_h.ACK_k$. That is, E_i accepts g_j such that $s_j[p] \rightarrow s_j[g_j]$. If E_i fails to receive p, E_i can find it on receipt of g_j by the failure (F) condition. Since g_j is not accepted, q is not pre-acknowledged. If E_i receives p, p is pre-acknowledged. It contradicts the assumption. □

This means that PDUs are pre-acknowledged in each E_i according to the causality-precedence order $\prec$.

4.5 Acknowledgment procedure

The next problem is when each entity delivers PDUs to the application entity. When E_i knows that p has been pre-acknowledged by every entity, p is referred to as *acknowledged* by E_i, i.e. E_i is sure that all the destinations have acknowledged or pre-acknowledged p. We consider how to acknowledge p.

For a PDU q from E_k, each receipt confirmation $q.ACK_j$ for E_j is stored in PAL_{jk} when q is pre-acknowledged (j = 1,...,n). E_i knows that E_k has pre-acknowledged p (from E_j) such that $p.SEQ < PAL_{jk}$. That is, the minimum $minPAL_j$ among PAL_{j1},...,PAL_{jn} means that every entity has pre-acknowledged p such that $p.SEQ < minPAL_j$. Hence, if p in PRL_i satisfies the ACK condition, p is acknowledged in E_i and p is moved to ARL_i which keeps in record of the acknowledged PDUs by the ACK action.

[Acknowledgment (ACK) condition]
$p.SEQ < minPAL_j$ (where $p.SRC = E_j$). □

[Acknowledgment (ACK) action]
While ($p = top(PRL_i)$ satisfies the ACK condition) {
  p := *dequeue*(PRL_i); *enqueue*(ARL_i, p);
} □

An application entity A_i receives the acknowledged PDUs in ARL_i by dequeuing the top from ARL_i.

[Example 4.2] As presented in Example 4.1, each time a PDU is pre-acknowledged, PAL is changed. For example, when h from E_3 is pre-acknowledged, PAL_{13} := $h.ACK_1 = 5$, $PAL_{23} := h.ACK_2 = 3$, and PAL_{33} := $h.ACK_3 = 2$, and the following PAL is obtained:

$$PAL = \begin{bmatrix} 4 & 4 & 5 \\ 2 & 2 & 3 \\ 2 & 2 & 2 \end{bmatrix}.$$

Here, a, b, c, d, and e are acknowledged because $a.SEQ < c.SEQ < e.SEQ < minPAL_1$ (= 4), $d.SEQ < minPAL_2$ (= 2), and $b.SEQ < minPAL_3$ (= 2). Here, $a \prec b \preceq c \prec d \prec e$ (where $b \parallel c$). □

[Proposition 4.4] Let p and q be PDUs pre-acknowledged by E_i such that $p \prec q$. If q is acknowledged, p is acknowledged.

[Proof] Suppose that $p.SRC = E_j$ and $q.SRC = E_k$. From Proposition 4.3, since $p \prec q$, p precedes q in PRL_i. Hence, the ACK action is applied to p before q and PRL_i is causality-preserved, i.e. $p \rightarrow_{PRL_i} q$. □

[Theorem 4.5] The CO protocol provides the CO service by using the MC service.

[Proof] If there is no PDU loss, it is straightforward from Proposition 4.3 and 4.4. If some PDU g is lost, g is detected by the failure (F) condition and rebroadcast by the retransmission action. From Proposition 4.3, if $p \prec q$, q is not pre-acknowledged unless p is pre-acknowledged. From Proposition 4.4, q is acknowledged after p is acknowledged. □

5 Evaluation

In this section, we consider the performance of the CO protocol for a cluster $\mathcal{C} = \langle E_1,...,E_n \rangle$. In the implementation, if there is no data, each entity does not send a PDU each time one PDU is received in order to reduce the number of PDUs transmitted in the cluster. After receiving at least one PDU from every entity or some predefined time, each entity sends a PDU, i.e. deferred acknowledgment. Let W be a window size of each entity. Hence, each PDU p is acknowledged when $2nW$ PDUs are received after p is received, i.e. nW PDUs for pre-acknowledgment and another nW for acknowledgment. This means that the required buffer size is $O(n)$. Since each PDU carries n receipt confirmations in the ACK field as shown in Figure 4, the length of PDU is $O(n)$.

Let $\mathcal{R}$ be the maximum propagation delay time among the entities. That is, it takes $\mathcal{R}$ time units from the transmission of a PDU p by the source entity until the receipt of p by the destination entity which is the farthest one from the source. If all the PDUs which carry the receipt confirmation for p are broadcast in parallel, it takes $\mathcal{R}$ from the acceptance of p until the pre-acknowledgment of p. Thus, it takes $2\mathcal{R}$ time units to acknowledge p after its acceptance.

If some PDUs are lost, only the PDUs lost are retransmitted, i.e. the selective retransmission is adopted. In addition, no synchronization among the entities is needed to find where to store the PDUs retransmitted in the receipt logs and the data transmission is not stopped while the PDU loss is being recovered. In general, protocols which provide the TO service [14, 15, 17] use the *go-back-n* retransmission scheme where all PDUs preceding the lost PDU are retransmitted. More PDUs are being transmitted in the high-speed network than the conventional network. Hence, the selective retransmission is required to provide high-throughput data transmission in the high-speed network.

The CO protocol uses the sequence numbers SEQ and ACK_j (j = 1,...,n) carried by the PDUs to causally order the PDUs as presented in Theorem 4.1. SEQ is incremented by one each time a PDU is transmitted. PDU loss can be detected by using SEQ. More computation to synchronize the virtual clock is required and the PDU loss cannot be detected by the virtual clocks in ISIS.

The CO protocol is implemented in a user process of the Sun SPARC2 workstation. The Ethernet is used as the MC network. Each workstation has one CO entity and one application entity. In this evaluation, each application entity sends data transmission (DT) requests to the CO entity continuously like the file transfer. Figure 8 shows the processing time (T_{CO}) per each PDU of each entity and the transmission delay

(T_{ap}) among the application entities for a number n of the entities in a cluster. This figure shows that the processing overhead of each entity is $O(n)$.

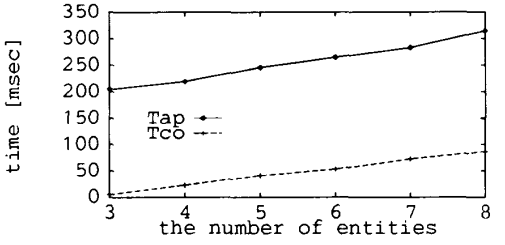


Figure 8: Processing time and delay time

6 Concluding Remarks

In this paper, we have discussed the CO (causally ordering broadcast) protocol which uses the high-speed multi-channel (MC) network. Here, each system entity may fail to receive PDUs due to the buffer overrun and may receive PDUs from different entities in different orders. By using the CO protocol, every application entity in the cluster can receive PDUs in the causality-precedence order. The cooperation of the entities supporting the cluster is coordinated in the distributed control scheme. That is, the entities decide on the atomic and causally ordered receipt of the PDUs by itself while broadcasting the PDUs asynchronously in the cluster. The CO protocol uses the sequence numbers of the PDUs to order PDUs received in the causality-precedence sequence while the PDU loss can be detected by the sequence numbers. We have shown the evaluation of the CO protocol and also shown that the processing overhead of each entity to acknowledge each PDU is $O(n)$ through the implementation.

References

[1] Abeysundara, B. W. and Kamal, A. E., "High-Speed Local Area Networks and Their Performance: A Survey," *ACM Computing Surveys*, Vol.23, No.2, 1991, pp.221-264.

[2] Birman, K. P. and Joseph, T. A., "Reliable Communication in the Presence of Failures," *ACM Trans. on Computer Systems*, Vol.5, No.1, 1987, pp.47-76.

[3] Birman, K. P., Schiper, A., and Stephenson, P., "Lightweight Causal and Atomic Group Multicast," *ACM Trans. on Computer Systems*, Vol.9, No.3, 1991, pp.272-314.

[4] Chang, J. M. and Maxemchuk, N. F., "Reliable Broadcast Protocols," *ACM Trans. on Computer Systems*, Vol.2, No.3, 1984, pp.251-273.

[5] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," *Comm. ACM*, Vol.34, No.1, 1991, pp.38-58.

[6] Garcia-Molina, H. and Spauster, A., "Ordered and Reliable Multicast Communication," *ACM Trans. on Computer Systems*, Vol.9, No.3, 1991, pp.242-271.

[7] Kaashoek, M. F. and Tanenbaum, A. S., "Group Communication in the Amoeba Distributed Operating System," *Proc. of the 11th IEEE ICDCS*, 1991, pp.222-230.

[8] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, Vol.21, No.7, 1978, pp.558-565.

[9] Luan, S. W. and Gligor, V. D., "A Fault-Tolerant Protocol for Atomic Broadcast," *IEEE Trans. on Parallel and Distributed Systems*, Vol.1, No.3, 1990, pp.271-285.

[10] Melliar-Smith, P. M., Moser, L. E., and Agrawala, V., "Broadcast Protocols for Distributed Systems," *IEEE Trans. on Parallel and Distributed Systems*, Vol.1, No.1, 1990, pp.17-25.

[11] Nakamura, A. and Takizawa, M., "Reliable Broadcast Protocol for Selectively Ordering PDUs," *Proc. of the 11th IEEE ICDCS*, 1991, pp.239-246.

[12] Nakamura, A. and Takizawa, M., "Design of Reliable Broadcast Communication Protocol for Selectively Partially Ordered PDUs," *Proc. of the IEEE COMPSAC'91*, 1991, pp.673-679.

[13] Nakamura, A. and Takizawa, M., "Priority-Based Total and Semi-Total Ordering Broadcast Protocols," *Proc. of the 12th IEEE ICDCS*, 1992, pp.178-185.

[14] Takizawa, M., "Cluster Control Protocol for Highly Reliable Broadcast Communication," *Proc. of the IFIP Conf. on Distributed Processing*, 1987, pp.431-445.

[15] Takizawa, M., "Design of Highly Reliable Broadcast Communication Protocol," *Proc. of the IEEE COMPSAC'87*, 1987, pp.731-740.

[16] Takizawa, M. and Nakamura, A., "Partially Ordering Broadcast (PO) Protocol," *Proc. of the 9th IEEE INFOCOM*, 1990, pp.357-364.

[17] Takizawa, M. and Nakamura, A., "Reliable Broadcast Communication," *Proc. of IPSJ Int'l Conf. on Information Technology (InfoJapan)*, 1990, pp.325-332.

[18] Tanenbaum, A. S., "Computer Networks (2nd ed.)," *Englewood Cliffs, NJ: Prentice-Hall*, 1989.