マルチメディア通信と分散処理 67－2
（１９９４．１２．２）

ε- 因果関係保存グループ通信 (ε-CO) プロトコル

立川 敬行　滝沢 誠
東京電機大学理工学部経営工学科
E-mail {tachi, taki}@takilab.k.dendai.ac.jp

複数のプロセス間のグループ通信では、送信されたメッセージを各プロセスがどのような順序で受信するかが問題となる。また、応用レベルのメッセージは、網上ではパケットに分割され、送信される。これらの分割されたパケットの集合をメッセージのストリームとする。ストリーム内のパケットの順序と各ストリーム間の順序を考える必要がある。本論文では、グループ内の全プロセスに対して、パケット間ではなく、ストリームの受信順序を保障する高信頼なグループ通信プロトコルについて論じる。高速通信網での輻輳とオーバーランによるパケットの紛失が起きても、メッセージとしての受信を行なえるときには、復旧を行なわない方式を考える。本プロトコルは、主制御プロセスの存在しない完全分散型の制御方式に基づいている。

ε-Causally Ordering Group Communication (ε-CO) Protocol

Takayuki Tachikawa and Makoto Takizawa
Tokyo Denki University

The distributed applications require group communication of multimedia data among multiple processes. A message including multimedia data at the application level is decomposed into smaller packets at the communication system level. In this paper, we discuss the ordered, atomic, and non-loss delivery of messages at the application level not at the system level. In some multimedia applications, the application processes do not mind if some packets are lost. The application process specifies the minimum receipt ratio ε (≤ 1) showing at least how many percentages of whole data in each message the destination process has to receive. The communication system delivers the packets to the destinations in the group so as to satisfy the receipt constraint ε. The protocol is based on the fully distributed control scheme, and uses high-speed networks where the process may lose packets due to the buffer overrun and congestion.

1 Introduction

In the distributed applications like computer-supported cooperative work (CSCW) [5], a group of multiple processes are cooperated and multimedia data like video are exchanged among the processes in the group. In this paper, a group of processes is referred to as *cluster*. In the group communication, each application process either delivers messages to all the destinations in the cluster or none of them, i.e. *atomic delivery*. In the high-speed networks [1], packets may be lost due to the network congestion. Since the transmission speed is faster than the processing speed of each process while the data transmission on the network is almost error-free, process may not receive packets transmitted in the network. Thus, in the presence of packet loss, the packets have to be delivered to all the destinations, i.e. *non-loss delivery*.

In addition to providing the atomic and non-loss delivery, the group communication protocol has to provide the application processes with kinds of *ordered* delivery of messages: locally (LO), causally (CO), and totally ordering (TO) delivery. In the LO service, messages from each process are received in the sending order. That is, if a process sends message q after p, every destination process receives q after p. In the TO service [4, 7, 8, 10], all the destinations receive messages in the same order and in the sending order. In the CO service, messages received are ordered by Lamport's *happened-before* relation [9]. That is, if p is sent *logically* before q, p is delivered to every destination before q. The CO service is required in distributed applications like fault-tolerant systems [13] and CSCW [5].

In the distributed applications, multimedia data like video and voice are exchanged among the processes. Application processes transmit multimedia messages to the destinations in the cluster. The *system* process in the communication system takes the message from the application process, decomposes it to smaller packets, and sends and receives the packets by using the high-speed network. The system process may receive the packets out of order and may not receive some packets. The group communication protocols [3, 4, 7, 8, 11, 12, 15] support the atomic, ordered, and non-loss delivery of the packets while the application processes may not care loss of packets. In this paper, the application specifies how much ratio ε (≤ 1) of data in each message have to be received at least by each destination. If the system process could receive more packets than ε of the message even if it does not receive some packets, it is allowed to pass them to the application and does not require the sender to send the lost packets again. The application processes are interested only in the receipt order of messages but not packets.

In section 2, we present a concept of message stream. In section 3, we discuss causal ordered delivery. In section 4, we present the data transmission procedure. Finally, we present the evaluation of the ε-CO protocol in section 5.

2 Message Stream

The communication system is composed of three hierarchical layers, i.e. *application*, *system*, and *network* ones. A cluster C is a set of n (≥ 2) system *service access points* (SAPs), i.e. $\{C_1, ..., C_n\}$. Each application process A_i takes some communication service through C_i which is supported by a system process S_i ($i = 1, ..., n$). $S_1, ..., S_n$ cooperate with one another by a *group communication* protocol to support group communication service for C by using the underlying network. C is written as $C = \langle S_1, ..., S_n \rangle$. The network layer provides high-speed data transmission [1] for the system layer. A data unit exchanged among application processes and among system processes are referred to as *messages* and *packets*, respectively. S_i may not receive packets due to the buffer overrun and congestion.

Application processes $A_1, ..., A_n$ exchange messages including multimedia data through the cluster C. Message a sent by A_i is passed to S_i and S_i is decomposes a into smaller packets $a_1, ..., a_h$ ($h \geq 1$). Here, a_i is referred to as *in a*. For example, one frame of MPEG [6] is decomposed into cells in the ATM network [2]. A *stream* of message a is a collection $\langle a_1, ..., a_h \rangle$ ($h \geq 1$) of the packets decomposed from a. In Figure 1, application process A_i sends message a to A_j and A_j sends b to A_i after receiving a. S_i takes a from A_i and decomposes a into a stream $\langle a_1, a_2, a_3 \rangle$. After receiving a_1, a_2, and a_3, S_j reassembles them into a and passes a to A_j. Similarly S_j decomposes b into $\langle b_1, b_2 \rangle$. S_j sends b_1 and b_2 to S_i.

Packet a_i *depends on* a_j ($a_i \models a_j$) iff a_i cannot be passed to the application process if a_j cannot. For example, if the key for deciphering a_i is included in a_j, the application process cannot accept a_i because a_i cannot be deciphered unless a_j is received, i.e. $a_i \models a_j$. a_i and a_j are *independent* iff neither $a_i \models a_j$ nor $a_j \models a_i$. a_i *precedes* a_j ($a_i \vdash a_j$) iff a_i has to be passed to the application process before a_j. a_i and a_j are *equivalent* iff neither $a_i \vdash a_j$ nor $a_j \vdash a_i$.

[**Assumption**] Packets a_i and a_j in message a are independent and equivalent. □

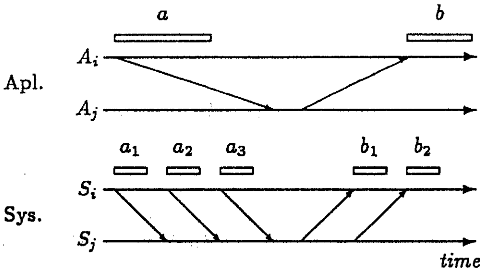


Figure 1: Message stream

In the high-speed network, packets may be lost and may be delivered to the destinations not in the same order. Application processes may not always require the communication system to receive all the data in each message. For example, in MPEG, the application process can playout the video image from the frames received even if some B-frames are lost.

Suppose that A_i sends a to A_j. S_i decomposes a into stream $\langle a_1, ..., a_h \rangle$. It is written as $a = \langle a_1, ..., a_h \rangle$. Let $|a|$ denote number h of the packets in a. S_i sends $a_1, ..., a_h$ to the destination, say S_j by using the high-speed network. S_j reassembles $a_1, ..., a_h$ into a and passes a to A_j. S_j may not receive some packets due to the buffer overrun and congestion. Here, let *received stream* $\rho_{ij}(a)$ be substream of a including packets which S_j receives from S_i, $\rho_{ij}(a) \subseteq a$. The *receipt ratio* $\pi_{ij}(a)$ of a in A_j is $|\rho_{ij}(a)| \,/\, |a|$ (≤ 1).

Here, *Receipt logs* RL_i^A and RL_i^S denote sequences of messages and packets which application and system process A_i and S_i receive, respectively. *Sending logs* SL_i^A and SL_i^S denote sequences of messages and packets which A_i and S_i send, respectively. If message a precedes b in log L, $a \rightarrow_L b$. For example, in RL_i^A, if A_i receives message a before b, $a \rightarrow_{RL_i^A} b$.

[**Definition**] Let a be a message sent to A_j by A_i. For a constant ϵ (≤ 1), RL_j^A is *ϵ-information preserved* iff A_j can accept $\rho_{ij}(a)$ if $\pi_{ij}(a) \geq \epsilon$. □ ε is the minimum receipt ratio which A_i gives to the system process S_i when sending message a. Even if S_i does not receive some packets of a, S_i notifies A_i of the acceptance of a if S_i could receive more packets of a than ε. In the MPEG, for I-frame, $\varepsilon = 1$, and for B-frame, $\varepsilon \leq 1$. For example, S_i decomposes message a issued by A_i into a stream $\langle a_1, ..., a_5 \rangle$ and sends it to S_j, and S_j receives a_1, a_2, a_3, and a_5 but not a_4. $\rho_{ij}(a) = \langle a_1, a_2, a_3, a_5 \rangle$ and $\pi_{ij}(a) = |\rho_{ij}(a)|/|a| = 4/5 = 0.8$. Suppose that A_i requires S_i to send a with $\varepsilon_i = 0.6$. Since $\pi_{ij}(a) = 0.8 > \varepsilon_i$, S_j passes $\rho_{ij}(a)$ to A_j.

3 Causal Ordering of Messages

Here, let $s_i[a]$ and $r_i[a]$ be sending and receipt events of message a in application process A_i, respectively. Precedence relations $\Rightarrow$ [3] among the events and $\prec$ among the messages are defined as follows.

[Definition] For every pair of events e and e', $e \Rightarrow e'$ iff

(1) e happens before e' in A_i,

(2) for some (not necessarily different) A_i and A_j, there exists some message a such that $e = s_i[a]$ and $e' = r_j[a]$, or

(3) for some event e'', $e \Rightarrow e''$ and $e'' \Rightarrow e'$. $\Box$

[Definition] For every pair of messages a and b, *a causally precedes b* ($a \prec b$) iff $s_i[a] \Rightarrow s_j[b]$. $\Box$

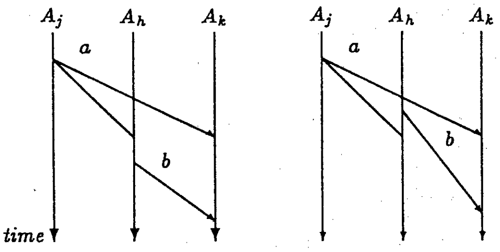
$a \prec b$ means that a is sent before b at the application level. a and b are *causally coincident* ($a \parallel b$) iff neither $a \prec b$ nor $b \prec a$. $a \preceq b$ means that $a \prec b$ or $a \parallel b$.

[Definition] For every pair of messages a and b which A_i receives from A_j, RL_i^A is *local-order preserved* iff $a \rightarrow_{RL_i^A} b$ if $a \rightarrow_{SL_j^A} b$. $\Box$

If RL_i^A is local-order preserved, A_i receives messages from each A_j in the sending order.

[Definition] RL_i^A is *causally preserved* iff for every pair of messages a and b in RL_i^A, $a \rightarrow_{RL_i^A} b$ if $a \prec b$. $\Box$

In Figure 2(1), since A_k receives b after a, RL_k^A is causally preserved. In Figure 2(2), $a \parallel b$ since A_h sends b before receiving a.



(1) (2)

Figure 2: Causality at application level

[Definition] RL_i^S is *serial* if for every message $a = \langle a_1, ..., a_t \rangle$ and $b = \langle b_1, ..., b_u \rangle$,

(1) $a_h \rightarrow_{RL_i^S} a_k$ if $h < k$, and

(2) there is no b_k such that $a_1 \rightarrow_{RL_i^S} b_k \rightarrow_{RL_i^S} a_t$. $\Box$

(1) means that S_i receives packets from each S_j in the sending order. (2) means that packets in a and b are not interleaved.

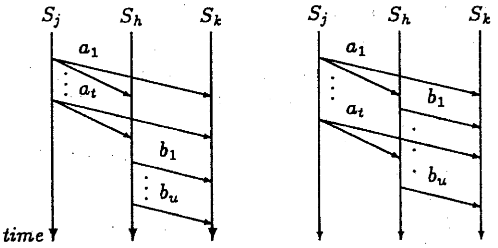
System process S_i takes message a from A_i. $t_i[a]$ denotes the event. S_i decomposes it to a stream and sends the stream to S_j by using the high-speed network. S_j receives the packets from S_i and reassembles them into message, i.e. $\rho_{ij}(a)$. S_j delivers $\rho_{ij}(a)$ to A_j. This event is denoted by $d_j[a]$. Then, A_j receives $\rho_{ij}(a)$ from S_j. $r_j[a]$ denotes this receipt event. Let a and b be message sent by A_i and A_j, respectively.

[Definition] *a causally precedes b* ($a \prec b$) iff $t_i[a] \Rightarrow t_j[b]$. $\Box$

[Definition] RL_h^A is *causally preserved* for each common destination A_h of a and b iff $d_h[a] \Rightarrow d_h[b]$ if $t_i[a] \Rightarrow t_j[b]$. $\Box$

Let us consider Figure 2 and Figure 3. Here, we can assume that $s_i[a]$ and $t_i[a]$ occur simultaneously because S_i sends a as soon as taking a. Figure 3 shows the data transmission among three system processes S_j, S_h, and S_k. In Figure 3(1), S_h sends packet b_1 in b after receiving the last a_t in a. S_k receives a_t before b_1, i.e. $a_t \rightarrow_{RL_k^S} b_1$. Figure 2(1) shows the data transmission among A_j, A_h, and A_k for Figure 3(1). A_h sends b to A_k after receiving a and A_k receives b after a, i.e. $a \prec b$. Next, in Figure 3(2), S_h starts to send the stream of b before receiving all the packets in a. Figure 2(2) shows the application-level data transmission of Figure 3(2). Since A_h sends b not after receiving a, $a \parallel b$.

In Figure 3(2), suppose that S_k receives the packets as $RL_k^S = < a_1 \cdots b_1 \cdots a_t \cdots b_u$]. S_k reassembles them into message and passes it to A_k. S_k may pass a before b because a_1 is received before b_1. If so, the causal ordering of packets in a and b except a_1 and b_1 is meaningless for the application process. In this paper, we would like to discuss how to causally order messages at the application level not at the system level in order to reduce the processing overhead of the communication system.



(1) (2)

Figure 3: Causality at system level

In Figure 4, S_j sends S_h stream $\langle a_1, a_2, a_3, a_4, a_5 \rangle$ of message a. In Figure 4(1), S_h does not receive a_5, and S_h starts to send stream $\langle b_1, b_2 \rangle$ of b to S_k before passing a to A_h. Here, $a \parallel b$. In Figure 4(2), S_h does not receive a_2. If the receipt ratio $\pi_{jh}(a) > \varepsilon$ in S_h on receipt of a_5, S_h passes $\rho_{jh}(a) = \langle a_1, a_3, a_4, a_5 \rangle$ to A_h. After that, S_h takes b from A_h and sends b_1 and b_2 to S_k. Here, $t_j[a] \Rightarrow t_h[b]$ since $d_h[a] \Rightarrow t_h[b]$. Hence, $a \prec b$.

We define the following services supported by

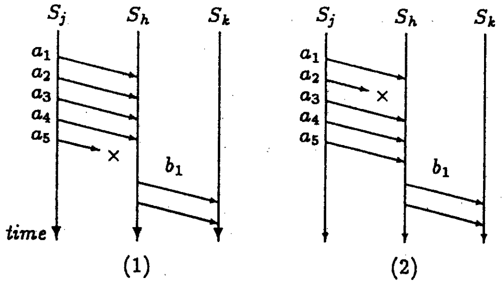


Figure 4: Causality at system level

the system and application layers.

[Definition]

(1) A network service is a *multi-channel* (MC) one iff every RL_i^S is local-order-preserved.

(2) A system service supported by $C = \langle S_1, ..., S_n \rangle$ is ***ε-causally ordered*** (ε-CO) iff every RL_i^A is ε-information-preserved and causally preserved. □

S_i can receive the packets from each process in the sending order in the MC service while S_i may not receive some packets. The MC service is supported by a system where every two computers are connected by logical or physical high-speed links. In this paper, we would like to discuss how to support a group of application processes with ε-CO service by using the MC service.

4 ε-CO Protocol

Here, suppose that a cluster C is $\langle S_1, ..., S_n \rangle$.

4.1 Variables

System process S_i has the following variables to send and receive streams (j, k = 1, ..., n).

- ε = minimum receipt ratio given by A_i.
- SEQ = sequence number of stream which S_i expects to send next or is sending at present, i.e. *stream number*.
- REQ_j = sequence number of stream which S_i has accepted most recently from S_i with respect to ε.
- DLV_j = sequence number of stream from S_j which S_i has passed most recently to A_i.
- AL_{jk} = sequence number of stream which S_i knows S_j accepted most recently from S_k.

S_i has the following variables to send and receive packets (j = 1, ..., n).

- $mSEQ$ = sequence number of packet which S_i expects to send next, i.e. *packet number*.
- $mREQ_j$ = sequence number of packet which S_i expects to receive next from S_j.
- LST_j = number of packets lost in a stream which S_i is receiving from S_j.
- BUF_j = available buffer size of S_j which S_i knows.

Here, let $minAL_j$ denote minimum in AL_{1j}, ..., AL_{nj} (j = 1, ..., n). Let $minBUF$ be minimum in BUF_1, ..., BUF_n.

Let $\langle a_1, ..., a_h \rangle$ be a stream of message a sent by A_i. Each packet a_k in a has the following fields (k = 1, ..., h).

- CID = cluster identifier.
- SRC = source system process of a_k, i.e. S_i.
- SEQ = stream number of a.
- $mSEQ$ = packet number of a_k.

Stream a has the following variables (j = 1, ..., n).

- $mTotal$ = number of packets in a, i.e. $|a| = h$.
- ACK_j = sequence number of stream which S_i has accepted most recently from S_j.
- $DACK_j$ = sequence number of stream from S_j which S_i has passed most recently to A_i.
- BUF = available buffer size of S_i
- SRC = source system process of a_k, i.e. S_i.
- SEQ = stream number of a.

4.2 Transmission and acceptance of packets

S_i decomposes message a from A_i to stream $\langle a_1, ..., a_h \rangle$. Here, let W be a maximum window size and H be a maximum number of packets in stream, i.e. $h \leq H$. Let f be the maximum size of packet. S_i has to have buffer to store at least $O(n)$ packets [?]. S_i sends packet a_k by the following data transmission procedure if the following flow condition holds. Here, *enqueue*(L, a) means that packet a is enqueued into a queue L. *send*(a) means that a is sent to S_1, ..., S_n by using the MC network. S_i has a sending queue SL_i to store packets which S_i sends.

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

```
[Packet transmission procedure] {
for $k = 1$ to $h$ {
    $a_k.SRC := S_i$;  $a_k.SEQ := SEQ$;
    $a_k.mSEQ := mSEQ$;  $mSEQ := mSEQ + 1$;
    $enqueue(SL_i, a_k)$;
    if the flow condition holds, $send(a_k)$;
        else waits; }
$SEQ := SEQ + 1$; } □
```

On receipt of packet a_k from S_j, S_i accepts a_k by the following procedure. S_i stores packets accepted from S_j into the receipt queue RRL_{ij} (j = 1, ..., n).

```
[Acceptance(mACC) procedure] If $a_k.SEQ = REQ_j + 1$ {
if $mREQ_j \neq a_k.mSEQ$ {
    $LST_j := LST_j + (a_k.mSEQ - mREQ_j)$; }
$mREQ_j := a_k.mSEQ + 1$;  $BUF_j := a_k.BUF$;
$enqueue(RRL_{ij}, a_k)$; } □
```

In RRL_{ij}, the packets from S_j are stored in the sending order. On receipt of a_k, if $a_k.mSEQ > mREQ_j$, it is found that S_i does not receive a_h where $a_k.mSEQ > a_h.mSEQ \geq mREQ_j$ but S_i

does not require S_j to retransmit a_h. The number of packets lost is accumulated in LST_j. If LST_j / $|a|$ (= $a_x.mTotal$) $> 1 - \varepsilon$, S_i requires S_j to send the packets lost again.

4.3 Transmission and acceptance of stream

S_i sends the stream a by the following procedure.

```
[Stream transmission procedure] {
$a.SRC := S_i$;  $a.SEQ := SEQ$;
$a.mTotal := h$;  $a.ACK_j := REQ_j$ ($j = 1,...,n$);
$a.DACK_j := DLV_j$ ($j = 1,...,n$);
$a.BUF :=$ available buffer size of $S_i$; } □
```

Here, some packet a_x in a has additional fields *mTotal*, ACK_j, $DACK_j$, and *BUF*. If S_i does not receive a_x from S_j, S_i cannot accept a, i.e. $a_k \models a_x$ ($k \neq x$). Hence, if S_i loses a_x, S_i requires S_j to send a_x again. Another way is that a_x is sent more than one time. Even if one replica of a_x is lost, S_i can receive another replica of a_x.

Here, suppose $a = \langle a_1, ..., a_h \rangle$. If S_i accepts the last packet a_h in a or it takes some time units after S_i accepts packet in a, S_i decides whether to accept the stream.

```
[Acceptance(ACC) procedure] { If $(a.mTotal - LST_j)/ a.mTotal \geq \varepsilon$ for $a$ in $RRL_{ij}$, {
    the packets in $a$ are reassembled into stream $a$;
    $AL_{jk} := a.ACK_k$ ($k = 1, ..., n$);
    $BUF_j := a.BUF$;  $enqueue(CRL_{ij}, a)$:
    $REQ_j := a.SEQ$; $mREQ_j := 1$;
    $LST_j := 0$; } } □
```

If S_i accepts more packets in a than $\varepsilon \times h$, S_i accepts $\rho_{ij}(a)$ from S_j. $\rho_{ij}(a)$ is enqueued into ARL_{ij} as the stream a. The streams from S_j are stored in the sending order in ARL_{ij}.

Unless S_i cannot accept a, S_i requires S_j to send again the packets which S_i has lost. S_j sends RET(retransmission) packet to S_j which carries the list of *SEQ*s of the packets. On receipt of the RET packet from S_j, S_j sends the packets to S_i again. On receipt of the packets retransmitted, S_i stores them into RRL_{ij}.

4.4 Causally ordered and atomic delivery

Suppose that S_i accepts stream $\rho_{ij}(a)$ of message a sent by S_j. Before passing $\rho_{ij}(a)$ to A_i, S_i has to order the streams in the causal precedence order $\prec$. The streams a and b are ordered in $\prec$ by the following condition.

[Causality(C) condition] For every message a and b, $a \prec b$ iff
(1) $a.SEQ < b.SEQ$ ($a.SRC = b.SRC$),
(2) $a.SEQ \leq b.DACK_j$ ($a.SRC$ $(= S_j) \neq b.SRC$). □

If S_i sends a and b, $t_j[a] \Rightarrow t_j[b]$ from condition (1). If not, $d_j[a] \Rightarrow t_j[b]$ from condition (2). Hence, $t_j[a] \Rightarrow t_k[b]$ ($b_1.SRC = S_k$), i.e. $a \prec b$ at the application level. S_i orders streams in $\prec$ by the following procedure. $dequeue(ARL_{ij}, a)$ means that the top a of ARL_{ij} is removed.

```
[Causality Ordering (C) procedure] {
while (the top $a$ in some $ARL_{ij}$ is found such that $a \preceq b$ for the top of every $ARL_{ih}$) {
    $dequeue(ARL_{ij}, a)$; $enqueue(CRL_i, a)$; } } □
```

S_i has to know that all the system processes have accepted a. If a satisfies the following acknowledgment (ACK) condition, S_i considers a as accepted by all the system processes, and passes a to A_i by the following procedure. $deliver(a)$ means that S_i passes a to A_i. Here, a is referred to as *acknowledged* in C.

[Acknowledgment (ACK) condition]
$minAL_j \geq a.SEQ$ (where $a.SRC = S_j$). □

```
[Passing message (P) procedure] {
while (the stream of the top $a$ (where $a.SRC = S_j$) of $CRL_i$ satisfies the ACK condition) {
    $dequeue(CRL_i, a)$ ; $DLV_j := a.SEQ$
    $deliver(a)$: } } □
```

Figure 5 shows the overall flows of messages in the data transmission procedure.

5 Evaluation

The ε-CO protocol supports the causally ordered delivery of application messages and not packets. Here, suppose that message a is decomposed into packets $a_1, ..., a_h$ ($|a| = h$). We compare the ε-CO protocol with the CO protocol supporting the causally ordered delivery of packets at the communication system level with respect to the number of comparison operations executed to deliver a to the application process. Table 1 shows the number of comparisons. In the CO protocol, the fields of every packet are checked for the ACC, ACK, and C procedures. On the other hand, only the top packet a_1 is checked by the ACK and C conditions in the ε-CO protocol. Thus, the processing overhead of a can be more reduced in the ε-CO protocol.

Table 1: Number of comparison operations

procedure protocol	ACC ($O(1)$)	C ($O(n)$)	ACK ($O(n)$)
CO ($\varepsilon = 1$)	h	h	h
ε-CO	h	1	1

ACC = Acceptance condition,
C = Causally preserved condition,
ACK = Acknowledgment condition.

Next, we would like to think about the number of packets retransmitted for packet loss. In the ε-CO protocol, the packets are not retransmitted if the destinations accept more than ε packets in message. While each time loss of packet is detected, the lost packet is retransmitted in the CO protocol. Let h be average number of packets in message. Let σ be probability that packet is lost

$$\begin{array}{ccccccccc}
 & & CRL_i & & ARL_{ij} & & RRL_{ij} & & \\
 & & & \| & <(a)\,(b)\,] \leftarrow & \| & <a_1\,a_2\,\ldots\,a_h\,] \leftarrow & \| & \leftarrow S_1 \\
A_i \leftarrow & \| & <(a)\,] \leftarrow & \| & <(p)\,(q)\,] \leftarrow & \| & <p_1\,p_2\,\ldots\,p_h\,] \leftarrow & \| & \leftarrow S_j \\
 & & & \| & <(x)\,(y)\,] \leftarrow & \| & <x_1\,x_2\,\ldots\,x_h\,] \leftarrow & \| & \leftarrow S_n \\
 & ACK & & C & & ACC & & mACC &
\end{array}$$

a_i : packet a_i, (a) : stream a.

Figure 5: Receipt procedure

by one destination. In the CO protocol, the packets lost are retransmitted. Hence, $\sigma \times h$ packets are retransmitted. On the other hand, if $\sigma \leq (1-\varepsilon)$, there is no packet retransmitted in the ε-CO protocol. If $\sigma > (1-\varepsilon)$, $(\varepsilon - \sigma) \times h$ packets are retransmitted. Figure 6 shows the number of packets retransmitted for σ. The ε-CO protocol implies less retransmissions.

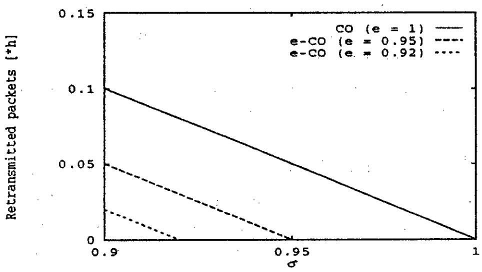


Figure 6: Retransmitted packets for σ

6 Concluding Remarks

In this paper, we have discussed a group communication protocol named ε-CO protocol to exchange multimedia data among application processes by using the high-speed network. The messages are causally ordered at the application level not at the communication system level. Application specifies the minimum receipt ratio ε of messages to be received. The ε-CO protocol is based on the distributed control. By using the ε-CO protocol, the processing overhead of the communication system can be reduced because it is not required to support the atomic, causally ordered, and non-loss delivery of every packet transmitted in the network. The ε-CO protocol implies less processing overhead and less retransmissions than the packet-level group communication protocols.

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] Bae, J. J. and Suda, T., "Survey of Traffic Control Schemes and Protocols in ATM Networks," *Proc. of the IEEE*, Vol.79, No.2, 1991, pp.170-189.

[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] Gall, D., "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. ACM*, Vol.34, No.4, 1991, pp.46-58.

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

[8] 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.

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

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

[12] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," *Proc. of the 14th IEEE ICDCS*, 1994, pp.48-55.

[13] Powell, D., Chereque, M., and Drackley, D., "Fault-Tolerance in Delta-4," *ACM Operating Systems Review*, Vol.25, No.2, 1991, pp.122-125.

[14] Ravindran, K. and Shah, K., "Causal Broadcasting and Consistency of Distributed Shared Data," *Proc. of the 14th IEEE ICDCS*, 1994, pp.40-47.

[15] Tachikawa, T. and Takizawa, M., "Selective Total Ordering Broadcast Protocol," *Proc. of the 2nd IEEE ICNP*, 1994, pp.212-219.

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

[17] Verissimo, P., Rodrigues, L., Baptista, M., "AMp: A Highly Parallel Atomic Multicast Protocol," *Proc. of ACM SIGCOMM*, 1989, pp.83-93.