Selective Total-Ordering Group Communication on Single High-Speed Channel

Takayuki Tachikawa and Makoto Takizawa
Dept. of Computers and Systems Engineering
Tokyo Denki University
Ishizaka, Hatoyama, Hiki-gun, Saitama 350-03, JAPAN
e-mail {tachi, taki}@takilab.k.dendai.ac.jp

Abstract

In the group communication, multiple processes have to receive messages in some order. In this paper, we discuss a group communication protocol which supports a selective total-ordered (ST) and atomic delivery of messages to the destinations in a group of processes interconnected by a high-speed channel, where the processes may fail to receive messages due to the buffer overruns. That is, each process receives messages destined to it in the sending order and any two common destinations of messages are received in the same order. Its execution is controlled in a distributed scheme, i.e. no master controller.

1 Introduction

In distributed applications like groupware [6], group communication among multiple processes is required in addition to conventional one-to-one communication provided by OSI [8] and TCP/IP [5]. In the group communication, messages sent by one process have to be delivered to either all the destinations or none in the group, i.e. *atomic delivery*. In addition, each process has to receive messages sent by the processes in some order. Group communications have been studied in [3, 4], [7], [9], [11, 12], [13]-[16], and [17]-[21]. [17] presents a reliable broadcast protocol which uses the one-to-one communication. An important problem in the group communication is which process coordinates the cooperation of multiple processes in the group. Most approaches [4, 7, 17] adopt the centralized control scheme, where one master process decides on the atomic and ordered delivery of messages. ISIS [3] uses a decentralized one where a sender of each message controls it. We adopt the *distributed* approach where every process makes a decision on it by itself. [18]-[21] present the distributed protocols.

Let us consider a group D composed of clients C_1 and C_2 and database servers S_1, S_2, and S_3 where data object x is stored in S_1 and S_2, and y is stored in S_2 and S_3. Suppose that C_1 would write x and y, and C_2 would write only x. C_1 sends a write operation op_1 to S_1, S_2, and S_3, and C_2 sends write op_2 to S_1 and S_2. Thus, each process sends each message to any subset of the group at any time. [13] discusses a *selective order-preserving* (SP) protocol where each server receives operations from each client in the sending order while S_1 may receive op_1 after op_2 and S_2 may receive op_1 before op_2. If S_1 and S_2 could receive op_1 and op_2 in the same order, it is easy to realize the serializability [2]. It is a *selective total-ordering* (ST) service where every common destinations of messages receive the messages in the same order. In this paper, we discuss a distributed protocol which provides the ST service for the group by using the high-speed network [1] where each process may fail to receive messages due to the buffer overrun. In addition, every process can send messages asynchronously. In ISIS[3], each message p is sent to a pre-defined group and all the processes in the group receive p. The processes in the intersection of multiple groups can receive messages in the ST order. In the ST protocol, each process can send messages to any subset of the group at any time.

In section 2, we model broadcast communication services. In section 3, we present a data transmission procedure. In section 4, failure recovery mechanisms are discussed. Finally, we discuss the evaluation of the ST protocol in section 5.

2 Service Model

2.1 Service properties

A communication system is composed of *application*, *system*, and *network* layers [Figure 1]. A cluster C [18, 19] is a set of n (≥ 2) system *service access points* (SAPs), i.e. $\{S_1,...,S_n\}$. Each application process A_i takes group communication service through S_i which is supported by a system process E_i ($i = 1,...,n$). $E_1,...,E_n$ cooperate with each other to support the group communication service for C by using the network layer. C is referred to as *supported* by $E_1,...,E_n$ ($C = \langle E_1,...,E_n\rangle$) and *support* $A_1,...,A_n$. The network layer supports the system layer with high-speed data transmission. Since the transmission speed is faster than the processing speed of E_i, E_i may fail to receive messages sent in the network layer. In this paper, we assume that each E_i loses the whole message if E_i fails to receive it, i.e. unconditional loss.

Let T_i denote a process at some layer. The underlying service used by T_i is modeled as a set of logs. A log L is a sequence of messages $< p_1 \ldots p_m$], where p_1 and p_m are the top ($top(L)$) and the last ($last(L)$), respectively. p_h *precedes* p_k ($p_h \rightarrow_L p_k$) in L if $h < k$. Each T_i has a sending log SL_i and a receipt log RL_i,

0-8186-6685-4/94 $4.00 © 1994 IEEE

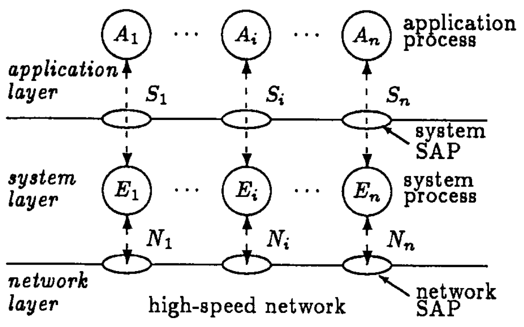


Figure 1: Layers

which are sequences of messages sent and received by E_i, respectively.

RL_i is *order-preserved* iff $p \rightarrow_{RL_i} q$ if $p \rightarrow_{SL_j} q$ for some E_j. RL_i is *information-preserved* iff RL_i includes all the messages in $SL_1,...,SL_n$. RL_i is *preserved* iff RL_i is order- and information-preserved. RL_i and RL_j are *order-equivalent* iff for every pair of p and q in both RL_i and RL_j, $p \rightarrow_{RL_i} q$ iff $p \rightarrow_{RL_j} q$. RL_i and RL_j are *information-equivalent* iff they include the same messages. RL_i and RL_j are *equivalent* iff they are order- and information-equivalent.

In a high-speed network, system processes may fail to receive messages due to the buffer overrun because the transmission speed is faster than the processing speed [1]. A *one-channel* (1C) service is a model of the high-speed network where every RL_i is order-equivalent. Figure 2 shows an example of the 1C service for $C = \langle T_1, T_2, T_3\rangle$ where all the processes receive messages in the same order while RL_2 and RL_3 are not information-preserved because T_2 and T_3 fail to receive c and q, respectively.

RL_1:	$<a\ x\ b\ c\ p\ y\ d\ z\ q\ ]$	SL_1:	$<a\ b\ c\ d\ ]$
RL_2:	$<a\ x\ b\ p\ y\ d\ z\ q\ ]$	SL_2:	$<p\ q\ ]$
RL_3:	$<a\ x\ b\ c\ p\ y\ d\ z\ ]$	SL_3:	$<x\ y\ z\ ]$

Figure 2: 1C service

An *order-preserving* (OP) service is one where every RL_i is preserved. A *total ordering* (TO) service is an OP one where every RL_i is order-equivalent. In the OP service, every T_i receives messages from each T_j in the sending order without message loss. Every T_i receives messages in the same order in the TO service.

2.2 Selective broadcast service

In a cluster C supporting application processes A_1, ..., A_n, each A_i sends a message to the destinations (not necessarily all the processes) in C. RL_i is *selectively information-preserved* iff RL_i includes all the messages sent to E_i. RL_i is *selectively preserved* iff RL_i is order-preserved and selectively information-preserved. A *selective broadcast* (SB) service is one where every RL_i is selectively information-preserved.

RL_1:	$<c\ x\ z\ p\ ]$	RL_1:	$<x\ c\ p\ z\ ]$
RL_2:	$<x\ a\ q\ b\ y\ ]$	RL_2:	$<a\ x\ b\ y\ q\ ]$
RL_3:	$<a\ x\ c\ z\ q\ ]$	RL_3:	$<a\ x\ c\ z\ q\ ]$
	(a) SP service		(b) ST service

SL_1: $<a_{\{2,3\}}\ b_{\{2\}}\ c_{\{1,3\}}\ ]$
SL_2: $<p_{\{1\}}\ q_{\{2,3\}}\ ]$
SL_3: $<x_{\{1,2,3\}}\ y_{\{2\}}\ z_{\{1,3\}}\ ]$

Figure 3: SP and ST services

[Definition] An SB service S is *selectively order-preserving* (SP) iff every RL_i is selectively preserved. S is *selectively total ordering* (ST) iff every RL_i is selectively preserved and is order-equivalent. □

Here, every RL_i in the ST service is referred to as *selectively totally ordered*. Let $p.DST$ ($\subseteq C$) be a set of destination application processes $\{A_{d_1},...,A_{d_m}\}$ of p. Here, p can be written as $p_{\{d_1,...,d_m\}}$. p^i denotes that p is sent by A_i. In the SP service[13], if $p \rightarrow_{SL_j} q$, $p \rightarrow_{RL_i} q$ for every $A_i \in p.DST \cap q.DST$. For p and q sent by different processes, p is sent by A_i and q is sent by A_j ($i \neq j$), every pair of A_k and A_h in $p.DST \cap q.DST$ receive both p and q in the same order in the ST service. For example, $a_{\{2,3\}}$ and $x_{\{1,2,3\}}$ are received by the common destinations A_2 and A_3 in the same order in Figure 3(b).

A *logical precedence* relation ($\Rightarrow$) among p and q is defined as follows.

[Definition] p *logically precedes* q ($p \Rightarrow q$) iff (1) $p \rightarrow_{SL_i} q$ and $p.DST \cap q.DST \neq \phi$, or $p \rightarrow_{RL_i} q$ for some E_i, (2) $p = q$, or (3) for some message r, $p \Rightarrow r \Rightarrow q$. □

$\Rightarrow$ is reflexive and transitive. Since $x \Rightarrow c$ from $s \rightarrow_{RL_1} c$ and $c \Rightarrow z$ from $c \rightarrow_{RL_3} z$ in Figure 3(b), $x \Rightarrow z$. If $\Rightarrow$ is anti-symmetric, $\Rightarrow$ is *consistent*. Otherwise, $\Rightarrow$ is *inconsistent*. $\Rightarrow$ is *complete* iff either $p \Rightarrow q$ or $q \Rightarrow p$ if $p.DST \cap q.DST \neq \phi$ for every p and q. In the ST service, $\Rightarrow$ has to be consistent and complete.

2.3 Distributed atomic receipt

In this paper, we adopt the distributed control scheme to realize the atomic receipt of each message p among all the destinations in the system cluster $C = \langle E_1, ..., E_n\rangle$. There are three criteria levels [18, 19] by which each E_i decides whether to atomically receive p from E_k in C:

(1) *Acceptance*: E_i receives p.

(2) *Pre-acknowledgment*: E_i knows that every destination of p has accepted p.

(3) *Acknowledgment*: E_i knows that p has been pre-acknowledged by every destination of p.

Even if E_i pre-acknowledges p, another E_j may think that some E_k has not accepted p because E_j fails to receive the acceptance confirmation of p from E_k. Hence, the third level is required.

3 ST Protocol

We would like to discuss a data transmission procedure of the ST protocol for a cluster $C = \langle E_1,...,E_n \rangle$ by using the 1C service.

3.1 Variables

There are two kinds of messages, i.e. *DT* (*data*) and *RR* (*receive ready*) ones. DT is used to send data in C. If E_i has no data, E_i transmits RR to all the processes every pre-defined period. Each DT message p has the following fields ($j = 1, ..., n$):

- $p.SRC$ = source process E_i which transmits p.
- $p.DST$ = set of destination processes of p.
- $p.TSEQ$ = total sequence number of p.
- $p.PSEQ_j$ = partial sequence number for E_j
- $p.ACK_j$ = $TSEQ$ of a message which E_i expects to receive next from E_j
- $p.BUF$ = number of buffers available in E_i.
- $p.DATA$ = data.

$E_i \in p.DST$ means that E_i is a destination of p. For every pair of p and q from E_i, $p.TSEQ < q.TSEQ$ if $p \rightarrow_{SL_i} q$. If $E_j \in p.DST \cap q.DST$ and $p \rightarrow_{SL_i} q$, $p.PSEQ_j < q.PSEQ_j$. If $p \rightarrow_{SL_i} q$, $E_j \notin p.DST$, and there is no message r such that $p \rightarrow_{SL_i} r \rightarrow_{SL_i} q$ and $E_j \in r.DST$, then $p.PSEQ_j = q.PSEQ_j$. $p.ACK_j$ informs every process in C that E_i has accepted every message q from E_j where $q.TSEQ < p.ACK_j$. RR has the same fields as DT except that RR does not have DST and DATA fields because RR is sent to all the processes in C without data.

Each E_i has the following variables ($j = 1,...,n$):

- $TSEQ$ = total sequence number of a message which E_i expects to broadcast next.
- $PSEQ_j$ = partial sequence number of a message which E_i expects to send to E_j next.
- $TREQ_j = TSEQ$ of message which E_i expects to receive next from E_j.
- $PREQ_j = PSEQ_j$ of message which E_i expects to receive next from E_j.
- $AL_{hj} = TSEQ$ of message which E_i knows that E_j expects to receive next from E_h ($h = 1,...,n$).
- $PAL_{hj} = TSEQ$ of message which E_i knows that E_j expects to pre-acknowledge next from E_h ($h = 1,...,n$).
- BUF_j = number of buffers in E_j which E_i knows.

E_i obtains initial $TSEQ$, $PSEQ_j$, and BUF_j for every E_j in the cluster establishment [18, 19]. Let $minAL_j$ and $minBUF$ denote minimum ones in $AL_{j1},...,AL_{jn}$ and in $BUF_1,...,BUF_n$, respectively. E_i has a sending log SL_i and three receipt logs RRL_i, PRL_i, and ARL_i to store messages accepted, pre-acknowledged, and acknowledged, respectively.

3.2 Transmission and acceptance

E_i transmits a message p by the following action. Here, W gives the window size and H ($\geq W$) is a constant.

```
[Transmission action]
if $minAL_i \leq TSEQ <$
$minAL_i + min(W, minBUF / (H \times n))$, {
$p.TSEQ := TSEQ$; $TSEQ := TSEQ + 1$;
for ($j = 1,...,n$) {
        $p.PSEQ_j := PSEQ_j$;
        if ($E_j$ is a destination of $p$) {
            $PSEQ_j := PSEQ_j + 1$;
            $p.DST := p.DST \cup \{ E_j \}$; }
}
$p.ACK_h := TREQ_h$ ($h = 1,...,n$);
$p$ is broadcast at the network SAP $N_i$.} □
```

On receipt of p from E_j, E_i performs the following ACC action.

```
[Acceptance (ACC) action]
if (1) $p.TSEQ = TREQ_j$ or (2) $p.PSEQ_i = PREQ_j$
  and $E_i \in p.DST$, {
$TREQ_j := p.TSEQ + 1$;
$AL_{hj} := p.ACK_h$ ($h = 1,...,n$);
if ($E_i \in p.DST$) {
        $PREQ_j := p.PSEQ_i + 1$;
        $p$ is enqueued into $RRL_i$;
} else $p$ is discarded; } □
```

Even if E_i fails to receive g from E_j, E_i does not need to receive g unless $E_i \in g.DST$. For example, suppose that E_j broadcasts a, b, and c and E_i accepts a as shown in Figure 4. Here, $TREQ_j = 4$ and $PREQ_j = 3$ in E_i. On receipt of c, E_i detects a loss of b because $TREQ_j$ (= 4) < $c.TSEQ$ (= 5). Since $c.PSEQ_i = PREQ_j$ (= 3) and $E_i \in c.DST$, E_i knows that $E_i \notin b.DST$ [Figure 4 (a)]. If $E_i \notin c.DST$ and $c.PSEQ_i = 4$, there must be some message b destined to E_i, where $b.PSEQ_i = 3$ [Figure 4 (b)].

3.3 Pre-acknowledgment procedure

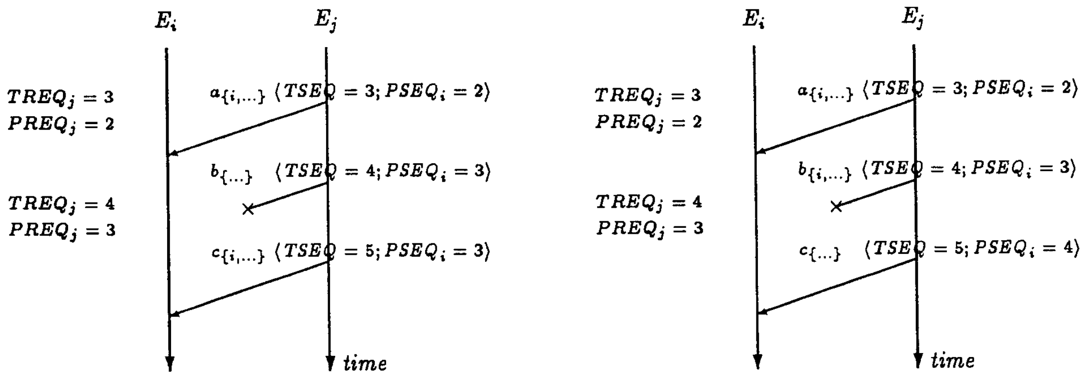
Let $minAL_j(p)$ be a minimum number in $\{ AL_{jh} \mid E_h \in p.DST \}$. E_i knows that every destination of p has accepted p if $p.TSEQ \leq minAL_j(p)$. Hence, p is pre-acknowledged in E_i by the PACK action.

```
[Pre-acknowledgment (PACK) action]
while (for $p$ (= $top(RRL_i)$), $p.TSEQ < minAL_j(p)$
        where $p.SRC = E_j$)
{    $p$ is dequeued from $RRL_i$;
     $p$ is enqueued into $PRL_i$;
     $PAL_{hj} := p.ACK_h$ ($h = 1,...,n$); } □
```

3.4 Acknowledgment procedure

Here, let $minPAL_j(p)$ be a minimum number in $\{ PAL_{jh} \mid E_h \in p.DST \}$. PAL_{jh} means that E_i knows that E_j has pre-acknowledged messages from E_h whose $TSEQ < PAL_{jh}$. Hence, E_i knows that every destination of p has pre-acknowledged p, i.e. p is acknowledged in E_i if $p.TSEQ < minPAL_j(p)$.

```
[Acknowledgment (ACK) action]
while (for $p$(=$top(PRL_i)$), $p.TSEQ < minPAL_j(p)$
        where $p.SRC$=$E_j$)
{    $p$ is dequeued from $PRL_i$;
     $p$ is enqueued into $ARL_i$;  } □
```



E_i
E_j
$TREQ_j = 3$
$PREQ_j = 2$
$TREQ_j = 4$
$PREQ_j = 3$
$a_{\{i,...\}}$ $\langle TSEQ = 3; PSEQ_i = 2\rangle$
$b_{\{...\}}$ $\langle TSEQ = 4; PSEQ_i = 3\rangle$
$c_{\{i,...\}}$ $\langle TSEQ = 5; PSEQ_i = 3\rangle$
time
E_i
E_j
$TREQ_j = 3$
$PREQ_j = 2$
$TREQ_j = 4$
$PREQ_j = 3$
$a_{\{i,...\}}$ $\langle TSEQ = 3; PSEQ_i = 2\rangle$
$b_{\{i,...\}}$ $\langle TSEQ = 4; PSEQ_i = 3\rangle$
$c_{\{...\}}$ $\langle TSEQ = 5; PSEQ_i = 4\rangle$
time

(a) $E_i \notin b.DST$ (b) $E_i \in b.DST$

Figure 4: Acceptance condition

If p is timed out in PRL_i, p is moved to ARL_i, i.e. acknowledged. The application process A_i receives the messages by dequeuing ARL_i.

4 Failure Recovery

4.1 Detection of message loss

We assume that processes never malfunction and the cluster C is aborted if any process stops by failure. In the 1C service, each E_i may fail to receive messages due to the buffer overrun. E_i has to receive only messages destined to E_i. As presented in the acceptance procedure, on receipt of p from E_j, if $PREQ_j < p.PSEQ_i$, E_i finds to lose g from E_j such that $PREQ_j \leq g.PSEQ_i < p.PSEQ_i$.

On receipt of q from E_h, for some j ($\neq h$), if $TREQ_j < q.ACK_j$, E_i has not received g from E_j such that $TREQ_j \leq g.TSEQ < q.ACK_j$. However, g may not be destined to E_i. E_i waits for messages from E_j. If g is not pre-acknowledged in a pre-defined time, E_i finds that some process fails to receive g. If E_i is a sender of g, E_i rebroadcasts g.

4.2 Recovery by insertion

Suppose that E_i finds the loss of g (from E_j). There are ***go-back-n*** and ***selective retransmission*** ways to recover from the message loss. In the go-back-n scheme, all the messages following g are removed from all the receipt logs and are retransmitted. We adopt the selective retransmission one since only g is retransmitted. On receipt of g, E_i has to keep RL_i selectively totally ordered after putting g in RL_i, i.e. $\Rightarrow$ be consistent. One way is that E_i finds where to insert g in RL_i and then puts g there.

[Definition] Suppose that E_i fails to receive g. A ***failure section*** $f_i(g)$ is an ordered pair $\langle p,\ q\rangle$ where

(1) $p \rightarrow_{RL_i} q$,

(2) $p \Rightarrow g$ and there is no message r in RL_i such that $p \Rightarrow r \Rightarrow g$, and

(3) $g \Rightarrow q$ and there is no r in RL_i such that $g \Rightarrow r \Rightarrow q$. □

[Proposition 1] Suppose that E_i fails to receive g and g is retransmitted. If g is inserted only in $f_i(g){=}\langle p,\ q\rangle$, RL_i is selectively totally ordered. □
[Proof] From the definition, $p \Rightarrow g \Rightarrow q$. Hence, if g is inserted between p and q, $\Rightarrow$ is consistent. If g is inserted in the outside of $\langle p,\ q\rangle$, say, before p, $\Rightarrow$ is inconsistent because $g \Rightarrow p$ and $p \Rightarrow g$. □

Figure 5 shows that E_3 fails to receive c in Figure 3(b). Here, since $x \Rightarrow z \Rightarrow q$ in RL_3 and $x \Rightarrow c \Rightarrow p \Rightarrow z$ in RL_1, $x \Rightarrow c \Rightarrow z$. $f_3(c)$ is $\langle x,\ z\rangle$. This means that c can be inserted between x and z in RL_3. If E_3 puts c in the outside of $f_3(c)$, e.g. $c \rightarrow_{RL_3} x$, $\Rightarrow$ is inconsistent since $x \rightarrow_{RL_1} c$, i.e. not selectively totally ordered.

$$\begin{array}{ll} RL_1: & <\ x\ c\ p\ z\ ] \\ RL_2: & <\ a\ x\ b\ y\ q\ ] \\ RL_3: & <\ a\ \langle x\ z\rangle\ q\ ] \end{array}$$

Figure 5: Failure section of c

Here, let a and b be messages destined to E_i. $a \Rightarrow b$ is ***self-decidable*** in E_i if (1) $a.SRC = b.SRC$ and $a.SEQ < b.SEQ$, or (2) there is some c in RL_i such that $a \Rightarrow c \Rightarrow b$. This means that E_i can decide on the receipt order of a and b by using the sequence numbers if E_i could receive a and b. $a \Rightarrow b$ is ***decidable*** if (1) it is self-decidable in some process or (2) there is some c such that $a \Rightarrow c$ and $c \Rightarrow b$ are decidable.

If E_i fails to receive g from E_j, E_j retransmits g and E_i receives g. As stated before, the problem is where to put g in RL_i. If E_i can decide where to put g in RL_i without communicating with another process, E_i is ***independently recoverable*** from the loss of g. E_i can independently recover from the loss of g if either

$p \Rightarrow g$ or $g \Rightarrow p$ is self-decidable for every p in RL_i. Otherwise, E_i is *dependently recoverable.*

[Example 1] Let $a_{\{1,2\}}$, $b_{\{2,3\}}$, and $c_{\{1,3\}}$ be messages.
(1) Suppose that E_1, E_2, and E_3 fail to receive a, b, and c, respectively. If a, b, and c are sent by the same process in this sequence, each E_i can independently recover from the message loss by using the sequence number. For example, on receipt of c, $b \Rightarrow c$ is decidable in E_3 since $b.SEQ < c.SEQ$.
(2) Suppose that a, b, and c are sent by different processes. Suppose E_1 receives a and b in $a \rightarrow_{RL_1} b$, E_2 receives b and c in $b \rightarrow_{RL_2} c$, and E_3 fails to receive c and c is retransmitted. $a \Rightarrow c$ is decidable in E_3 because E_3 could obtain $a \Rightarrow b$ from E_1 and $b \Rightarrow c$ from E_2 by communicating with E_1 and E_2.

Suppose that E_1, E_2, and E_3 fail to receive a, b, and c, respectively. Here, a, b, and c are retransmitted, and are received by E_1, E_2, and E_3, respectively. Suppose that E_1 puts a after c, i.e. $c \Rightarrow a$, E_2 puts b after a, i.e. $a \Rightarrow b$, and E_3 puts c after b, i.e. $b \Rightarrow c$. In result, $\Rightarrow$ is inconsistent because $c \Rightarrow a \Rightarrow b \Rightarrow c$. Here, $c \Rightarrow a$, $a \Rightarrow b$, and $b \Rightarrow c$ are not decidable. Some additional synchronization mechanism among E_1, E_2, and E_3 is required to make an agreement on the consistent $\Rightarrow$ among a, b, and c. □

4.3 Recovery by sorting

Another way for recovering from message loss is to sort RL_i after putting g in RL_i. In Figure 5, c is retransmitted and E_3 puts c on the last of RL_3 on receipt of c. $\Rightarrow$ is inconsistent since $z \rightarrow_{RL_3} c$ and $c \rightarrow_{RL_1} z$. In order to resolve the inconsistency, the messages can be sorted, e.g. by the process number and $PSEQ$. Here, E_1 and E_3 obtain the same $c \Rightarrow z$.

Suppose that some process fails to receive g. Let $T(g)$ be a set $\{r \mid g \Rightarrow r$, or $q \Rightarrow r$ if $f_i(g) = \langle p, q\rangle$ for every $E_i \in q.DST\}$ which gives messages which have to be sorted if $\Rightarrow$ with g is changed. For example, $T(c) = \{c, p, q, z\}$ since $c \Rightarrow p \Rightarrow z$ and $z \Rightarrow q$ in Figure 5. Here, a *sort point*, $sort_i(g)$ of E_i for g means a message in RL_i from which to the last of RL_i messages are sorted. Here, $sort_i(g)$ is defined to be p in RL_i such that p in $T(g)$ and there is no r in RL_i such that $r \Rightarrow p$ and r in $T(g)$. For example, $sort_1(c) = c$, $sort_2(c) = q$, and $sort_3(c) = z$ in Figure 5. A tuple $\langle sort_1(g), ..., sort_n(g)\rangle$ is a *sort line*, *sline* (g) for g. Figure 6 shows that $sline(c)$ is $\langle c, q, z\rangle$ for c, where $\|\alpha$ means that α is the sort point.

Let l_1 be $sline(g_1) = \langle s_{11}(g_1), ..., s_{1n}(g_1)\rangle$ and l_2 be $sline(g_2) = \langle s_{21}(g_2), ..., s_{2n}(g_2)\rangle$. A *join* of l_1 and l_2, $l_1 \wedge l_2$ is $\langle s_1, ..., s_n\rangle$ where each s_i is s_{1i} if $s_{1i} \rightarrow_{RL_i} s_{2i}$, s_{2i} otherwise. If there are multiple messages g_1, ..., g_n lost by some processes, a sort line $sline(\{g_1, ..., g_n\})$ is $sline(g_1) \wedge ... \wedge sline(g_n)$.

From the definition of the sort line, the following proposition is straightforward.

[Proposition 2] A sublog including messages preceding the sort point in RL_i is selectively totally ordered. □

The sublogs $< x$], $< a\ x\ b\ y$], $< a\ x$] of RL_1, RL_2, RL_3 in Figure 6 are selectively totally ordered.

E_1 RL_1: $< x_{\{123\}} \parallel c_{\{13\}}\ p_{\{1\}}\ z_{\{13\}}$]
E_2 RL_2: $< a_{\{23\}}\ x_{\{123\}}\ b_{\{2\}}\ y_{\{2\}} \parallel q_{\{23\}}$]
E_3 RL_3: $< a_{\{23\}}\ x_{\{123\}} \parallel z_{\{13\}}\ q_{\{23\}}$]

Figure 6: Example of sort line

4.4 Recovery procedure

Since the high-speed network is more reliable, it is the most case that one message g is lost by one process E_i. In this case, g is retransmitted, and only E_i inserts g just before the sort point. It is referred to as a ***single-loss***. Thus, E_i can independently recover from the loss of g. On the other hand, if multiple messages are lost, i.e. ***multi-loss***, some additional synchronization protocol is required to make the logical precedence relation $\Rightarrow$ consistent after the messages lost are inserted in the logs by the processes losing them, i.e. dependent recovery is required as presented in Example 1. There are two ways of dependent recovery, ***synchronization*** and ***sorting***. The synchronization requires more communications than the sorting method. Hence, the following strategy is used in this paper:

(1) first, all the processes agree on the sort line, and

(2) then the insertion method is used for a single-loss, otherwise the sort method is used.

First, we would like to discuss how all the processes can agree on the sort line. Suppose that E_i fails to receive p from E_j.

[Agreement procedure]

(1) E_i finds that E_i fails to receive p (from E_j) such that $p.TSEQ > TREQ_j$ and $p.PSEQ > PSEQ_j$. E_i broadcasts RST(reset) message r_i where $r_i.TREQ_h := TREQ_h$ $(h = 1, ..., n)$.

(2) On receipt of r_i, each E_k stops the data transmission. E_k finds a set $P_h = \{q \mid q \in RRL_k, q.SRC = E_h$, and $q.TSEQ \geq r_i.TREQ_h\}$ for each E_h. Let P be $P_1 \cup \cdots \cup P_n$. E_k finds f in P such that $f \rightarrow_{RRL_k} q$ for every q in P. If P is ϕ, let f be $\perp$. For each E_h, let $TREQ_h$ be a minimum value in $T_h = \{t.TSEQ \mid f \rightarrow_{RRL_k} t$ and $t.SRC = E_h\}$ if $f \neq \perp$. E_k broadcasts an RST_PAK rp_k where $rp.TREQ_h := TREQ_h$ $(h = 1, ..., n)$.

(3) On receipt of rp_i, if rp_i arrives before rp_j, E_k stores the arrival order of rp_i, i.e. $E_i < E_j$ in $ORDR$. $TREQ_h := rp_i.TREQ_h$ if $TREQ_h > rp_i.TREQ_h$ $(h = 1, ..., n)$.

(4) On receipt of rp_1, ..., rp_n, E_k finds a set $Q_h = \{q \mid q \in RRL_k, q.SRC = E_h$, and $q.TSEQ \geq TREQ_h\}$ for each E_h. Let Q be $Q_1 \cup \cdots \cup Q_n$. E_k finds f in Q such that $f \rightarrow_{RRL_k} q$ for every q in Q. If Q is ϕ, let f be $\perp$. E_k broadcasts RST_ACK ra_k where $ra_k.TREQ_h := TREQ_h$, $ra_k.PREQ_h := PREQ_h$, $ra_k.PSEQ_h := PSEQ_h$, and $ra_k.ORDR := ORDR$ $(h = 1, ..., n)$.

(5) On receipt of ra_i, $PR_{ij} := ra_i.PREQ_j$ and $PS_{ij} := ra_i.PSEQ_j$ $(j = 1, ..., n)$. If $ra_i.TREQ_h \neq$

$TREQ_h$ for some h or $ra_i.ORDR \neq ORDR$, E_k broadcasts ABORT message to abort C.

(6) On receipt of ra_1, ..., ra_n, each E_k has the same $TREQ_1$, ...,$TREQ_n$. Then, f denotes a sort point of E_k. □

[Example 2] In Figure 6, E_3 fails to receive c. Suppose that $TSEQ$s of a, b, and c are 1, 2, and 3, $TSEQ$s of p and q are 1 and 2, and $TSEQ$s of x, y, and z are 1, 2, and 3, respectively. Here, E_1, E_2, and E_3 have the following $TREQ = < TREQ_1, TREQ_2, TREQ_3 >$.

$$\begin{array}{ll} E_1 & TREQ = < 4,3,4 >. \\ E_2 & TREQ = < 4,3,4 >. \\ E_3 & TREQ = < 3,3,4 >. \end{array}$$

First, E_3 broadcasts RST r_3 where $TREQ = < 3,3,4 >$ when E_3 finds the loss of c.

On receipt of r_3, E_1 obtains $P_1 = \{c\}$ since $c.TSEQ = r_3.TREQ = 3$, and $P_2 = P_3 = \phi$, i.e. $P = \{c\}$. E_2 and E_3 obtain $P = \phi$. Here, each E_i has the following f_i and RRL_i whose $||$ denotes f_i.

$$\begin{array}{ll} RRL_1\text{: } < x \; || \; c \; p \; z \;] & f_1 = c. \\ RRL_2\text{: } < a \; x \; b \; y \; q \; || \;] & f_2 = \perp. \\ RRL_3\text{: } < a \; x \; z \; q \; || \;] & f_3 = \perp. \end{array}$$

E_1 broadcasts RST_PAK rp_1 whose $TREQ = < 3,1,3 >$ because $c \rightarrow_{RRL_1} p \rightarrow_{RRL_1} z$, $c.TSEQ = 3$, $p.TSEQ = 1$, and $z.TSEQ = 3$. E_2 broadcasts rp_2 whose $TREQ = < 4,3,4 >$. E_3 broadcasts rp_3 whose $TREQ = < 3,3,4 >$.

On receipt of rp_1, rp_2, and rp_3, each E_i obtains $TREQ = < 3,1,3 >$. In E_2, f_2 is q since $q.TSEQ \geq 1$. In E_3, f_3 is z since $z.TSEQ \geq 3$, $q.TSEQ \geq 1$, and $z \rightarrow_{RRL_3} q$. In E_1, $f_1 = c$. Here, the sort line $sort(c)$ is $< f_1, f_2, f_3 > = < c, q, z >$ as shown in Figure 6. Each E_i broadcasts RST_ACK ra_i whose $TREQ = < 3,1,3 >$.

On receipt of ra_1, ra_2, and ra_3, the agreement procedure terminates. Here, every E_i has the same $TREQ$. Hence, all the messages preceding the sort point f_i are acknowledged in E_i. That is, x in RRL_1, a, x, b, and y in RRL_2, and a and x in RRL_3 are acknowledged. Finally, the following RRL_i is obtained.

$$\begin{array}{l} RRL_1\text{: } < c \; p \; z \;] \\ RRL_2\text{: } < q \;] \\ RRL_3\text{: } < z \; q \;]. \end{array}$$ □

[Proposition 3] By the agreement procedure, every process obtains the sort point for messages lost. □
[Proof] Suppose that E_i fails to receive p. E_i broadcasts RST r_i and every E_k receives r_i. In step (2) of the agreement procedure, E_k finds q^h preceding every message from E_h in RRL_k whose $TSEQ \geq r_i.TSEQ_h$ ($h = 1, ..., n$). Let f denote some q^h such that $q^h \rightarrow_{RRL_i} q^j$ for every E_j. This means that $p \Rightarrow f \Rightarrow q^j$ for every E_j and p can be inserted in RRL_k such that $p \rightarrow_{RRL_k} f$. However, there may be some q in RRL_k such that $p \Rightarrow q \rightarrow_{RRL_k} p$ because multiple messages may be lost. Then, E_k broadcasts rp_k where $rp_k.TREQ_h = TREQ_h = q^h.TSEQ_h$ ($h = 1, ..., n$).

In step (3), E_k receives rp_1, ..., rp_n. In (4), $TREQ_h$ denotes the minimum one in $\{ rp_1.TREQ_h, ..., rp_n.TREQ_h \}$ ($h = 1, ..., n$). Since every E_k sends ra_k and receives ra_1, ..., ra_n, E_k has the same $TREQ$. E_k finds q^h preceding every message from each E_h in RRL_k such that $q^h.TSEQ \geq TREQ_h$ ($h = 1, ..., n$). Then, let f denote some q^h in RRL_k such that $q^h \rightarrow_{RRL_k} q^j$ for every E_j. This means that $p \Rightarrow f$ and no message t in RRL_k such that $p \Rightarrow t \Rightarrow f$. That is, f denotes the sort point of E_k for all the messages lost. □

Which recovery method, insertion or sort one is used depends on whether it is a single-loss or not. Let $FAIL$ be the cardinality of $\{ PR_{ij} \mid PR_{ij} < PS_{ij}$ $(i, j{=}1, ..., n)\}$. A single-loss occurs if $FAIL = 1$, and multi-loss if $FAIL > 1$.

[Retransmission (RET) procedure]

(1) E_j retransmits p if E_j has sent p and $p.PSEQ_i \geq PR_{ij}$ for some $E_i \in p.DST$.

(2) On receipt of p where $E_i \in p.DST$, if $PREQ_i \leq p.PSEQ_j$, E_i puts p just before $spoint_i(p)$ in RRL_i if $FAIL = 1$, otherwise E_i puts p into the tail of RRL_i. □

In the agreement procedure, each E_j knows what messages E_j fails to receive and E_j has sent but another fails to receive. Unless E_j receives p from E_i such that $p.PSEQ_j < PS_{ij}$ in a pre-defined time, E_j requests E_i to retransmit p.

If there are multiple messages lost, each process sorts RRL_i by the following sorting rule.

[Sorting rule]

(1) If $p.SRC = E_i$, $q.SRC = E_k$, and $E_i < E_k$ in $ORDR$, then $p \rightarrow_{RRL_j} q$.

(2) If $p.SRC = q.SRC$ and $p.PSEQ_j < q.PSEQ_j$, then $p \rightarrow_{RRL_j} q$. □

[Example 3] (1) Insertion: $RRL_1 =< c^1 \; p^2 \; z^3 \;]$, $RRL_2 =< q^2 \;]$, and $RRL_3 =< z^3 \; q^2 \;]$ are obtained by applying the agreement procedure to Figure 6 as presented in Example 2. Since it is a single-loss, i.e. E_3 loses c, E_1 retransmits c and E_3 inserts c before z in RRL_3, i.e. $RRL_3 = < c \; z \; q \;]$.
(2) Sorting: Suppose that E_2 and E_3 fails to receive y and c, respectively in Figure 3(b) i.e. multi-loss. The sort line $sline(\{c, y\}) = \langle \; c, \; q, \; z \; \rangle$ as shown in Figure 6 is obtained by applying the agreement procedure. Then, $RRL_1 = < c^1 \; p^2 \; z^3 \;]$, $RRL_2 = < q^2 \;]$, and $RRL_3 = < z^3 \; q^2 \;]$. c and y are retransmitted and are appended to the last of RRL_3 and RRL_2, respectively. Suppose that RST_PAKs are received by E_3, E_1, and then E_2, i.e. $E_3 < E_1 < E_2$ in $ORDR$. Each E_i obtains RRL_i as shown in Figure 7 by sorting RRL_i. They are selectively totally ordered. □

$$\begin{array}{lll} RRL_1\colon < c^1_{\{13\}}\ p^2_{\{1\}}\ z^3_{\{13\}}\ ] & & < z^3_{\{13\}}\ c^1_{\{13\}}\ p^2_{\{1\}}\ ] \\ RRL_2\colon < q^2_{\{23\}}\ y^3_{\{2\}}\ ] & \Rightarrow & < y^3_{\{2\}}\ q^2_{\{23\}}\ ] \\ RRL_3\colon < z^3_{\{13\}}\ q^2_{\{23\}}\ c^1_{\{13\}}\ ] & & < z^3_{\{13\}}\ c^1_{\{13\}}\ q^2_{\{23\}}\ ] \end{array}$$

Figure 7: Example of sorting

From Proposition 2, it is clear for the following proposition to hold.

[Proposition 4] A sublog of every RL_i obtained by sorting messages following the sort point after putting all the messages lost by E_i on the last of RL_i is selectively totally ordered. □

[Theorem 5] The ST protocol provides the ST service by using the 1C service.

[Proof] It is clear if there is no message loss. We consider a case that some messages are lost. It is sure that every message loss is detected by the failure condition. From Proposition 3, every process agrees on the sort line by the agreement procedure. There are two cases, i.e. single-loss and multi-loss.
(1) Suppose that the single-loss occurs, say, E_i loses a message g. From Proposition 1, the theorem holds since only E_i inserts g in the sort point.
(2) Suppose that the multi-loss occurs. From Proposition 2 and 3, every receipt log obtained by the sorting method is selectively totally ordered. □

5 Evaluation

We would like to evaluate the ST protocol for a cluster $C = \langle E_1, ..., E_n \rangle$ in terms of the number of messages retransmitted in the presence of the message loss. Let d ($\leq n$) be an average number of destination processes of each message. Let f be a probability that each message is lost by one process. Let RL be a sequence of messages transmitted in the 1C network. Let L be a number of messages in RL. Each E_j accepts only messages destined to E_j from RL. Let F be $(1-f(d/n))^d$, i.e. probability that each message is received by every destination. The probability that each message is lost by at least one destination is $(1 - F)$.

Since the selective retransmission is used in the ST protocol, only messages lost are retransmitted. The expected number R_S of messages retransmitted in RL is given as follows.

$$R_S = L(1 - F). \tag{1}$$

Next, let us consider how many messages are retransmitted in the go-back-n scheme. In the go-back-n scheme, if E_i fails to receive p which is destined to E_i, E_i removes all the messages following p in RRL_i. The messages removed are retransmitted by their source processes. If a message q removed by E_i is accepted by E_j, E_j removes all the messages following q from RL_j. Thus, the removal of messages in one process is propagated to another process. The probability that $p = RL[l]$ is lost by one destination while every $RL[h] (h < l)$ is received by every destination is $F^{l-1}(1 - F)$ $(l \geq 1)$. The expected number of messages retransmitted for l is $F^{l-1}(1 - F)(L - l + 1)$ $(1 \leq l \leq L)$. Hence, the expected number R_G of messages retransmitted is given as follows.

$$R_G = \sum_{l=1}^{L} F^{l-1}(1 - F)(L - l + 1). \tag{2}$$

Figure 8 shows R_S/L and R_G/L for f where ST means $d/n = 0.5$ and TO means $d/n = 1$. Figure 9 shows R_S/L and R_G/L for d/n and $f = 0.005$. Following both figures, the ST protocol with the selective retransmission implies less number of messages retransmitted than the go-back-n. The number of messages retransmitted by the selective scheme is almost $O(d/n)$ while $O((d/n)^c)$ in the go-back-n one.

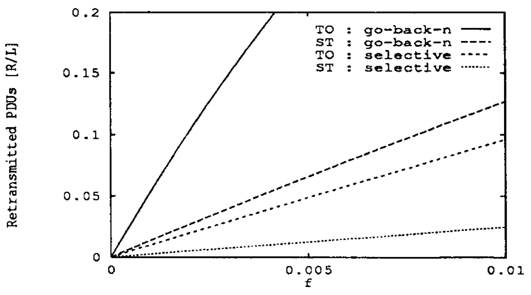


Figure 8: Ratios of retransmitted messages for f

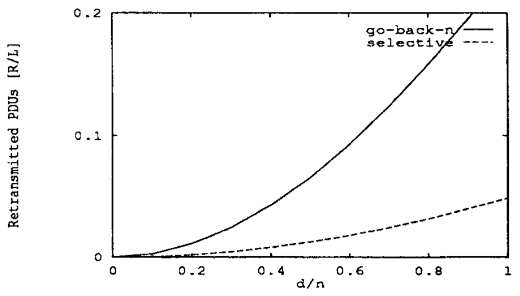


Figure 9: Ratios of retransmitted messages for d/n

Next, let us consider the processing time C to recover from the message loss. We assume that time to insert one message in the receipt log is 1, and time to sort h messages is h. We assume that time for comparing the sequence numbers of the messages in the log is neglectable compared with time to insert the messages in the log. The probability F_0 that only one message is lost by only one process is given $f(d/n)(1-f(d/n))^{nL-1}$. The probability F_1 that multiple messages are lost by more than one process is $1 - F_0 - F^L$. The expected time C of the ST recovery per each process is $F_0/n + F_1 R_G$. Here, let D be the cost of each process for processing messages in RL

without message loss. D is L from the assumption. Figure 10 shows the ratios of C to D for f and d/n. For example, if f is smaller than 0.001, the overhead for recovering from the message loss is below 20% of the normal processing time.

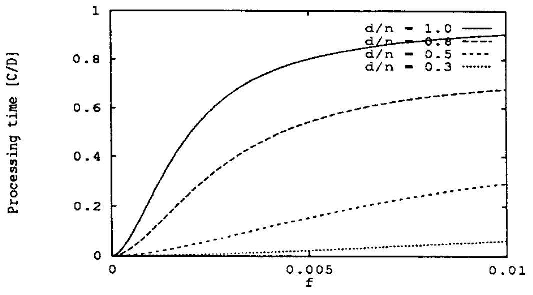


Figure 10: Ratio of the processing time for f

6 Concluding Remarks

In this paper, we have presented a group communication protocol which provides a group of processes, i.e. *cluster* with the selective totally-ordering (ST) service by using the high-speed 1C network. The protocol is based on the distributed control. In the ST service, each message is delivered to any processes in the group and different messages are received by the common destinations in the same order in the presence of the message loss. Furthermore, we have shown the evaluation of the ST protocol. By using the ST protocol, teleconferencing and cooperative work can be easily realized.

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] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addisson Wesley*, 1987.

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

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

[5] Defense Communications Agency, "DDN Protocol Handbook," Vol.1–3, NIC 50004-50005, 1985.

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

[7] Garcia-Molina, H. and Kogan, B., "An Implementation of Reliable Broadcast Using an Unreliable Multicast Facility," *Proc. of the 7th IEEE Symp. on Reliable Distributed Systems*, 1988, pp.428-437.

[8] International Standards Organization, "OSI – Connection Oriented Transport Protocol Specification," ISO 8073, 1986.

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

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

[11] 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.

[12] 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.

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

[14] Nakamura, A. and Takizawa, M., "Design of Reliable Broadcast Communication Protocol for Selectively Partially Ordering PDUs," *Proc. of the IEEE COMPSAC91*, 1991, pp.673-679.

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

[16] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," to appear in *Proc. of the IEEE ICDCS-14*, 1994.

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

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

[19] Takizawa, M., "Design of Highly Reliable Broadcast Communication Protocol," *Proc. of IEEE COMPSAC87*, 1987, pp.731-740.

[20] Takizawa, M. and Nakamura, A., "Partially Ordering Broadcast (PO) Protocol," *Proc. of the IEEE INFOCOM90*, 1990, pp.357-364.

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

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