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, takij}@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 centralised 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 C1 and C2 and database servers S1, S2, and S3 where data object z is stored in S1 and S2, and y is stored in S2 and S3. Suppose that C1 would write z and y, and C2 would write only z. C1 writes a write operation op1 to S1, S2, and S3, and C2 sends write op2 to S1 and S2. 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 S1 may receive op1 after op2 and S2 may receive op2 before op1. If S1 and S2 could receive op1 and op2 in the same order, it is easy to realise 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., (S1, ..., Sn). Each application process Ac takes group communication service through Sg, which is supported by a system process Es (i = 1, ..., n). Es, ..., En cooperate with each other to support the group communication service for C by using the network layer. C is referred to as supported by E1, ..., En (C = (E1, ..., En)) and support A1, ..., An. The network layer supports the system layer with high-speed data transmission. Since the transmission speed is faster than the processing speed of Es, Ei may fail to receive messages sent in the network layer. In this paper, we assume that each Ei loses the whole message if Ei fails to receive it, i.e. unconditional loss.

Let T denote a process at some layer. The underlying service used by T is modeled as a set of logs. A log L is a sequence of messages < p1, ..., pm >, where p1 and pm are the top (top(L)) and the last (last(L)), respectively. Let p_h precedes p_k (p_h ->_L p_k) in L if h < k. Each T has sending log SL and a receipt log RL.
which are sequences of messages sent and received by $E_i$, respectively.

$RL_i$ is order-preserved iff $p \rightarrow_{RL_i} q$ if $p \rightarrow_{SL_i} q$ for some $E_i$. $RL_i$ is information-preserved iff $RL_i$ includes all the messages in $SL_i$. $RL_i$ 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 $C = \{T_1, T_2, T_3\}$ 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 \times b \times c \times p \times y \times d \times z \times q \times j $ |
| $RL_2$: | $< a \times b \times p \times y \times d \times z \times q \times q \times p \times g \times q $ |
| $RL_3$: | $< a \times b \times c \times p \times y \times d \times z \times j $ |

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 $T_i$ 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.

$RL_i$: $< c \times z \times p \times j $  \hspace{1cm} $RL_j$: $< z \times c \times p \times p \times j $  
$RL_k$: $< z \times a \times q \times b \times y \times j $  \hspace{1cm} $RL_m$: $< a \times z \times c \times q \times j $  

(a) SP service \hspace{1cm} (b) ST service

$SL_1$: $< a(2,3) \times b(2) \times c(1,3) \times j $  \hspace{1cm} $SL_2$: $< p(1) \times q(1,3) \times j $  
$SL_3$: $< z(1,3) \times y(2) \times z(1,3) \times j $  

Figure 3: SP and ST services
3 ST Protocol

We would like to discuss a data transmission procedure of the ST protocol for a cluster \( C = \{ E_1, \ldots, E_n \} \) by using the IC 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 \( (j = 1, \ldots, n) \):

- \( p.SRC \) = 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 \( p \rightarrow \subseteq \text{SL}_i \ q \). If \( E_i \in p.DST \cap q \) \( \text{DST} \) and \( p \rightarrow \subseteq \text{SL}_i \ q \) \( p.TSEQ < q.TSEQ \). If \( p \rightarrow \subseteq \text{SL}_i \ q \) \( E_j \notin p \) \( \text{DST} \), and there is no message \( r \) such that \( p \rightarrow \subseteq \text{SL}_i \ r \rightarrow \subseteq \text{SL}_i \ q \) and \( E_j \in r \) \( \text{DST} \), 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 \( DATA \) fields because \( RR \) is sent to all the processes in \( C \) without data.

Each \( E_i \) has the following variables \( (j = 1, \ldots, n) \):

- \( TSEQ \) = 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 \( E_j \).
- \( AL_j = TSEQ \) of message which \( E_i \) expects to receive next from \( E_h (h = 1, \ldots, n) \).
- \( PAL_j = TSEQ \) of message which \( E_i \) knows that \( E_i \) expects to pre-acknowledge next from \( E_h (h = 1, \ldots, n) \).
- \( BUF_j \) = 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 \( minAL \) and \( minBUF \) denote minimum ones in \( AL_1, \ldots, AL_n \) and \( BUF_1, \ldots, BUF_n \), 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]
\[
\begin{align*}
\text{if } & minAL_i < TSEQ < minAL_i + min(W, \minBUF / (H \times n)), \{ \\
& p.TSEQ := TSEQ; \quad TSEQ := TSEQ + 1; \text{ for } (j = 1, \ldots, n) \{ \\
& \quad p.PSEQ_j := PSEQ_j; \text{ if } (E_i \text{ is a destination of } p) \{ \\
& \quad \quad PSEQ_j := PSEQ_j + 1; \\
& \quad \quad p.DST := p.DST \cup \{ E_j \}; \} \\
& \quad p.ACK_h := TREQ_h (h = 1, \ldots, n); \\
& p \text{ is broadcast at the network SAP } N_i. \} \Box
\end{align*}
\]

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

[Acceptance (ACC) action]
\[
\begin{align*}
\text{if } & (1) \ p.TSEQ = TREQ_j \text{ or } (2) \ p.PSEQ_h = PREQ_j \text{ and } E_i \in p.DST \{ \\
& \quad p.TREQ_j := p.TSEQ + 1; \quad PREQ_j := p.PSEQ_h + 1; \text{ if } E_i \notin p.DST \text{ and } c.PSEQ_h = \Box
\end{align*}
\]

Even if \( E_i \) fails to receive \( q \) from \( E_j \), \( E_i \) does not need to receive \( q \) unless \( E_i \in p.DST \). For example, suppose that \( E_i \) broadcasts \( a, b, \) and \( c \) and \( E_i \) accepts \( a \) as shown in Figure 4. Here, \( TREQ_q = 4 \) and \( PSEQ_q = 3 \) in \( E_i \). On receipt of \( c \), \( E_i \) detects a loss of \( b \) because \( TREQ_q = 4 < c.TSEQ_q = 5 \). Since \( c.PSEQ_q = PREQ_q \), \( = 3 \) 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 \( c.PSEQ_3 = 4 \), there must be some message \( b \) destined to \( E_i \), where \( b.PSEQ_3 = 3 \) [Figure 4 (b)].

3.3 Pre-acknowledgment procedure

Let \( minAL_i (p) \) be a minimum number in \{ \( AL_j (p) \mid E_i \in p.DST \} \). \( E_i \) knows that every destination of \( p \) has accepted \( p \) if \( p.TSEQ < minAL_i (p) \). Hence, \( p \) is pre-acknowledged in \( E_i \) by the PACK action.

[Pre-acknowledgment (PACK) action]
\[
\begin{align*}
\text{while } & (p = top(RRL_i), \ p.TSEQ < minAL_i (p)) \text{ where } p.SRC = E_i; \\
& \{ \text{p is dequeued from } RRL_i; \quad \text{p is enqueued into } PRL_i; \quad \text{PAL} \quad := p.ACK_h (h = 1, \ldots, n); \} \Box
\end{align*}
\]

3.4 Acknowledgment procedure

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

[Acknowledgment (ACK) action]
\[
\begin{align*}
\text{while } & (p = top(PRRL_i), \ p.TSEQ < minPAL_i (p) \text{ where } p.SRC = E_i); \\
& \{ \text{p is dequeued from } PRL_i; \quad \text{p is enqueued into } ARRL_i \}; \} \Box
\end{align*}
\]
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 I/C 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 \( \text{PREQ}_j < p, \text{PSEQ}_i \), \( E_i \) finds to lose \( q \) from \( E_j \) such that \( \text{PREQ}_j \leq g, \text{PSEQ}_i < p, \text{PSEQ}_i \).

On receipt of \( q \) from \( E_h \), for some \( j \neq h \), if \( \text{TREQ}_j < q, \text{ACK}_j \), \( E_i \) has not received \( q \) from \( E_j \) such that \( \text{TREQ}_j \leq q, \text{TSEQ}_j < q, \text{ACK}_j \). 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 \( q \), \( E_i \) 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 \((p, q)\) where

1. \( p \in RL_i \), \( q \).
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 \( f_i(g) = (p, q) \), then \( RL_i \) is selectively totally ordered. \( \square \)

[ 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 \((p, q)\), say, before \( p \Rightarrow g \Rightarrow q \). \( \square \)

Figure 5 shows that \( E_2 \) fails to receive \( c \) in Figure 3(b). Here, since \( z \Rightarrow x \Rightarrow q \in RL_3 \) and \( x \Rightarrow c \Rightarrow p \Rightarrow z \) in \( RL_1 \), \( z \Rightarrow c \Rightarrow z \). \( f_2(c) \) is \((x, z)\). This means that \( c \) can be inserted between \( x \) and \( z \) in \( RL_3 \). If \( E_2 \) puts \( c \) in the outside of \( f_2(c) \), e.g. \( c \Rightarrow RL_2 \), \( \Rightarrow \) is inconsistent since \( z \Rightarrow RL_3 \), \( c \), i.e. not selectively totally ordered.

\( RL_1 : \langle x, c, p, z \rangle \)
\( RL_2 : \langle a, x, b, y, q \rangle \)
\( RL_3 : \langle a, x, z, q \rangle \)

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) \( a, \text{SRC} = b, \text{SRC} \) and \( a, \text{SEQ} < b, \text{SEQ} \), 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) \( a \) is self-decidable in some process or (2) \( b \) is decidable.

If \( E_i \) fails to receive \( q \) from \( E_j \), \( E_i \) 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 \( RL_i \), independently recoverable from the loss of \( g \). \( E_i \) can independently recover from the loss of \( g \) if either
p \Rightarrow g \text{ or } g \Rightarrow p \text{ is self-decidable for every } p \text{ in } RL_i.
Otherwise, } E_i \text{ is dependently recoverable.}

[Example 1] Let } a_{(1,2)}, b_{(2,3)}, \text{ and } c_{(1,3)} \text{ be messages.}

(1) Suppose that } E_1, E_2, \text{ and } E_3 \text{ fail to receive } a, b, \text{ and } c, \text{ respectively. If } a, b, \text{ and } c \text{ are sent by the same process in this sequence, each } E_i \text{ can independently recover from the message loss by using the sequence number. For example, on receipt of } b, \text{ if } b \Rightarrow c \text{ is decidable, } E_i \text{ solves } b.SEQ < c.SEQ.

(2) Suppose that } a, b, \text{ and } c \text{ are sent by different processes. Suppose } E_1 \text{ receives } a \text{ and } b \text{ in } a \rightarrow RL_i, b \text{ in } E_2 \text{ receives } b \text{ and } c \text{ in } b \rightarrow RL_i, c \text{, and } E_3 \text{ fails to receive } c \text{ and } c \text{ is retransmitted. } a \Rightarrow c \text{ is decidable in } E_3 \text{ because } E_2 \text{ could obtain } a \Rightarrow b \text{ from } E_1 \text{ and } b \Rightarrow c \text{ from } E_2 \text{ by communicating with } E_1 \text{ and } E_2. \text{ Suppose that } E_1, E_2, \text{ and } E_3 \text{ fail to receive } a, b, \text{ and } c, \text{ respectively. Here, } a, b, \text{ and } c \text{ are retransmitted, and are received by } E_1, E_2, \text{ and } E_3, \text{ respectively. Suppose that } E_1 \text{ puts } a \text{ after } a, \text{ i.e. } a \Rightarrow b, \text{ and } E_2 \text{ puts } c \text{ after } b, \text{ i.e. } b \Rightarrow c. \text{ In result, } a \Rightarrow b \text{ is inconsistent because } a \Rightarrow c \Rightarrow a \Rightarrow b \Rightarrow c. \text{ Here, } c \Rightarrow a, a \Rightarrow b, \text{ and } b \Rightarrow c \text{ are not decidable. Some additional synchronization mechanism among } E_1, E_2, \text{ and } E_3 \text{ is required to make an agreement on the consistent } a \Rightarrow b, c \Rightarrow a, \text{ and } b \Rightarrow c. \square

4.3 Recovery by sorting

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

Suppose that some process fails to receive } g. \text{ Let } T(g) = \{ r | g \Rightarrow r \text{ or } g \Rightarrow r \text{ if } f_r(g) = (p, q) \text{ for every } E_0 \in q.DST \} \text{ which gives messages which have to be sorted if } \Rightarrow \text{ with } g \text{ is changed. For example, } T(c) = \{ c, p, q, z \} \text{ since } c \Rightarrow p \Rightarrow z \text{ and } z \Rightarrow g \text{ in Figure 5. Here, a sort point, } sort(c) \text{ of } E_i \text{ for } g \text{ means a message in } RL_i, \text{ from which to the last of } RL_i \text{ messages are sorted. Here, } sort_i(c) \text{ is defined to be } p \text{ in } RL_i \text{ such that } p \text{ is in } T(g) \text{ and there is no } r \text{ in } RL_i \text{ such that } r \Rightarrow p \text{ and } r \text{ in } T(g). \text{ For example, } sort_1(c) = c, sort_2(c) = q, \text{ and } sort_3(c) = z \text{ in Figure 5. A tuple } (sort_1(c), ..., sort_n(c)) \text{ is a sort line, } \text{ for } g \text{ Figure 6 shows that } \text{ for } c \Rightarrow z \text{ where } a \text{ is the sort point.}

Let } l_i \text{ be } \text{ for } g. \text{ Figure 6 shows that } \text{ for } c \Rightarrow z \text{ where } a \text{ is the sort point. Let } l_i \text{ be } \text{ for } g. \text{ Figure 6 shows that } \text{ for } c \Rightarrow z \text{ where } a \text{ is the sort point.}

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

The sublogs } x, y, z \text{ of } RL_1, RL_2, RL_3 \text{ in Figure 6 are selectively totally ordered.}

\[ E_1 \quad RL_1 : \quad \langle x_{(123)} \mid c_{(13)} \mid P_1 \rangle \quad x_{(123)} \]
\[ E_2 \quad RL_2 : \quad \langle a_{(23)} \mid z_{(23)} \mid \langle y_{(2)} \mid q_{(23)} \rangle \mid q_{(23)} \rangle \]
\[ E_3 \quad RL_3 : \quad \langle a_{(23)} \mid z_{(23)} \mid 2_{(13)} \mid q_{(23)} \rangle \]

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 \text{ is lost by one process } E_i. \text{ In this case, } g \text{ is retransmitted, and only } E_i \text{ inserts } g \text{ just before the sort point. It is referred to as a single-loss. Thus, } E_i \text{ can independently recover from the loss of } g. \text{ 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 \text{ 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 \text{ fails to receive } p \text{ from } E_j.

[Agreement procedure]

(1) } E_i \text{ finds that } E_j \text{ fails to receive } p \text{ (from } E_j) \text{ such that } p.\text{TSEQ} < TREQ_j \text{ and } p.\text{PSEQ} > PSEQ_j. \text{ E broadcasts } \text{RST} \text{(reset) message } r_i \text{ where } r_i.TREQ_i = TREQ_1 \langle h = 1, ..., n \rangle.

(2) On receipt of } r_i, \text{ each } E_k \text{ stops the data transmission. } E_k \text{ finds a set } P_k = \{ q \in RRL_k \mid q.SRC = E_i \text{ and } q.TSEQ \geq r_i.TREQ_i \} \text{ for each } E_k. \text{ Let } P = P_1 \cup ... \cup P_n. \text{ } E_k \text{ finds } f \text{ in } P \text{ such that } f \rightarrow RRL_k, q \text{ for every } q \in P. \text{ If } f \neq \bot, \text{ E broadcasts } \text{RST} \text{ message } r_i.TREQ_i \text{ for } h = 1, ..., n.

(3) On receipt of } r_i, \text{ if } r_i \text{ arrives before } r_i, \text{ E stores the arrival order of } r_i, \text{ i.e. } E_i \text{ in } E_j. \text{ E broadcasts } r_i.\text{TREQ} \langle h = 1, ..., n \rangle.

(4) On receipt of } r_i, ..., r_m, \text{ E finds a set } Q_h = \{ q \in RRL_k \mid q.SRC = E_i \text{ and } q.TSEQ \geq TREQ_h \} \text{ for each } E_i. \text{ E broadcasts } \text{RST} \text{ message } r_i.TREQ_i \text{ for } h = 1, ..., n.

(5) On receipt of } r_i, \text{ if } r_i \text{ arrives before } r_i, \text{ E stores the arrival order of } r_i, \text{ i.e. } E_i \text{ in } E_j. \text{ E broadcasts } r_i.\text{TREQ} \langle h = 1, ..., n \rangle.

On receipt of } r_i, ..., r_m, \text{ E finds a set } Q_h = \{ q \in RRL_k \mid q.SRC = E_i \text{ and } q.TSEQ \geq TREQ_h \} \text{ for each } E_i. \text{ E broadcasts } \text{RST} \text{ message } r_i.TREQ_i \text{ for } h = 1, ..., n.

On receipt of } r_i, ..., r_m, \text{ E finds a set } Q_h = \{ q \in RRL_k \mid q.SRC = E_i \text{ and } q.TSEQ \geq TREQ_h \} \text{ for each } E_i. \text{ E broadcasts } \text{RST} \text{ message } r_i.TREQ_i \text{ for } h = 1, ..., n.

On receipt of } r_i, ..., r_m, \text{ E finds a set } Q_h = \{ q \in RRL_k \mid q.SRC = E_i \text{ and } q.TSEQ \geq TREQ_h \} \text{ for each } E_i. \text{ E broadcasts } \text{RST} \text{ message } r_i.TREQ_i \text{ for } h = 1, ..., n.

On receipt of } r_i, ..., r_m, \text{ E finds a set } Q_h = \{ q \in RRL_k \mid q.SRC = E_i \text{ and } q.TSEQ \geq TREQ_h \} \text{ for each } E_i. \text{ E broadcasts } \text{RST} \text{ message } r_i.TREQ_i \text{ for } h = 1, ..., n.

On receipt of } r_i, ..., r_m, \text{ E finds a set } Q_h = \{ q \in RRL_k \mid q.SRC = E_i \text{ and } q.TSEQ \geq TREQ_h \} \text{ for each } E_i. \text{ E broadcasts } \text{RST} \text{ message } r_i.TREQ_i \text{ for } h = 1, ..., n.
TREQq for some h or rₐ.ORDR ≠ ORDₗ, Eₖ broadcasts ABORT message to abort C.

(6) On receipt of rₐ₁,..., rₐₙ, each Eₖ has the same TREQ₁, ..., TREQₙ. Then, f denotes a sort point of Eₖ. □

[Example 2] In Figure 6, E₃ fails to receive C. Suppose that TSEQₗ of a, b, and c are 1, 2, and 3, TSEQₗ of p and q are 1 and 2, and TSEQₗ of x₁, y₁, and z are 1, 2, and 3, respectively. Here, E₁, E₂, and E₃ have the following TREQ = < TREQ₁, TREQ₂, TREQ₃ >.

E₁ TREQ = < c, 4, 3, 4 >.
E₂ TREQ = < c, 4, 3, 4 >.
E₃ TREQ = < c, 3, 3, 4 >.

First, E₃ broadcasts RST rₐ₁ where TREQ = < 3, 3, 4 > when E₃ fails the loss of c.
On receipt of rₐ₁, E₁ obtains Pₐ₁ = {c} since c.TSEQ = rₐ₁.TREQ = 3, and Pₐ₁ = Pₐ = {c}, i.e. Pₐ₂ = {c}. E₂ and E₃ obtain Pₐ = {c}. Here, E₁ has the following fₐ₁ and RRL₉ whose || denotes fₐ₁.

\[ \text{RRL₉: } \begin{align*} &x \mid c \ p \ z \ \\ &f₁ = c. \end{align*} \]
\[ \text{RRL₁₉: } \begin{align*} &a \ z \ b \ q \ \\ &f₂ = L. \end{align*} \]
\[ \text{RRL₃₉: } \begin{align*} &a \ z \ q \ \\ &fₐ₃ = L. \end{align*} \]

E₁ broadcasts RST.PAK rₐ₁ whose TREQ = < 3, 1, 3 > because c → RRL₉, p → RRL₉, z, c.TSEQ = 3, p.TSEQ = 1, and z.TSEQ = 3. E₂ broadcasts rₐ₂ whose TREQ = < 4, 3, 4 >. E₃ broadcasts rₐ₃ whose TREQ = < 3, 3, 4 >.

On receipt of rₐ₁, rₐ₂, and rₐ₃, each Eₖ obtains TREQ = < 3, 1, 3 >. In E₂, f₂ is q since q.TSEQ > 1. In E₃, fₐ₃ is z since z.TSEQ > 3, q.TSEQ > 1, and z.RRL₉. q. In E₁, f₁ = c. Here, the sort line sort(c) is < f₁₉, f₂₉, fₐ₃ > = < c, q, z > as shown in Figure 6. Each Eₖ broadcasts RST.ACK rₐ₉ whose TREQ = < 3, 1, 3 >.

On receipt of rₐ₁, rₐ₂, and rₐ₃, the agreement procedure terminates. Here, E₁ has the same TREQ = < 3, 1, 3 >. Hence, all the messages preceding the sort point fₐ₉ are acknowledged in E₁. That is, in RRL₉, a, z, b, and y in RRL₉, and a and z in RRL₉ are acknowledged. Finally, the following RRL₉ is obtained.

\[ \text{RRL₉: } \begin{align*} &c \ p \ z \ \\ &RRL₁₉: \begin{align*} &c \ q \ z \ \\ &RRL₃₉: \begin{align*} &z \ q \ . \end{align*} \end{align*} \]

[Proposition 3] By the agreement procedure, every process obtains the sort point for messages lost. □

[Proof] Suppose that Eₖ fails to receive p. Eₖ broadcasts RST rₐ₁ and every Eₖ receives rₐ₁. In step (2) of the agreement procedure, Eₖ finds qₗ₉ preceding every message from Eₖ in RRL₉ whose TSEQ ≥ rₐ₁.TSEQₗ (h = 1, ..., n). Let f denote some qₗ₉ such that qₗ₉ → RRL₉, qₗ₉ for every Eₖ. This means that p ⇒ qₗ₉ for every Eₖ, and p can be inserted in RRL₉ such that p → RRL₉, f. However, there may be some qₗ₉ in RRL₉ such that p ⇒ qₗ₉ because multiple messages may be lost. Then, Eₖ broadcasts rₐ₉ where rₐ₉.TREQ₉ = TREQ₉ = qₗ₉.TSEQ₉ (h = 1, ..., n).

In step (3), Eₖ receives rₐ₁, ..., rₐ₉. In (4), TREQ₉ denotes the minimum one in { rₐ₁.TREQ₉, ..., rₐ₉.TREQ₉ } (h = 1, ..., n). Since every Eₖ sends rₐ₉ and receives rₐ₁, ..., rₐ₉, Eₖ has the same TREQ₉. Eₖ finds qₗ₉ preceding every message from each Eₖ in RRL₉ such that qₗ₉.TSEQ ≥ TREQ₉ (h = 1, ..., n). Then, let f denote some qₗ₉ in RRL₉ such that qₗ₉ → RRL₉, qₗ₉ for every Eₖ. This means that p ⇒ qₗ₉ and no message t in RRL₉ such that p ⇒ t ⇒ f. That is, f denotes the sort point of Eₖ 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 { Pᵢⱼ | Pᵢⱼ < Pₛᵢⱼ (i, j = 1, ..., n) }. A single-loss occurs if FAIL = 1, and multi-loss if FAIL > 1.

[Retransmission (RET) procedure]

(1) Eᵢ retransmits p if Eᵢ has sent p and Pₐⱼ ≥ Pᵢⱼ for some Eⱼ ∈ p.DST.

(2) On receipt of p where Eᵢ, p.DST, if TSEQᵢ < Pₛᵢⱼ, Eᵢ puts p just before span(p, Pₛᵢⱼ) in RRLᵢ if FAIL = 1, otherwise Eᵢ puts p into the tail of RRLᵢ. □

In the agreement procedure, each Eᵢ knows what messages Eᵢ fails to receive and Eᵢ has sent but another fails to receive. Unless Eᵢ receives p from Eⱼ such that p.Pₐⱼ < Pₛᵢⱼ in a pre-defined time, Eᵢ requests Eⱼ to retransmit p.

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

[Sorting rule]

(1) If p.SRC = Eᵢ, q.SRC = Eⱼ, and Eᵢ < Eⱼ in ORDᵢ, then p → RRLᵢ q.

(2) If p.SRC = q.SRC and p.Pₐⱼ < q.Pₐⱼ, then p → RRLᵢ q. □

[Example 3] (1) Insertion: RRL₁₉ = < c₁ p₁ z₁ >, RRL₂₉ = < q₁ >, and RRL₃₉ = < z₁ q₁ > are obtained by applying the agreement procedure to Figure 6 as presented in Example 2. Since it is a single-loss, i.e. E₁ loses c₁, E₁ retransmits c and E₁ inserts c before z in RRL₉, i.e. RRL₉ = < c z₁ >.

(2) Sorting: Suppose that E₁ and E₃ fail to receive y and c, respectively in Figure 3(b) i.e. multi-loss. The sort line span{c, z} = { c, q, z } as shown in Figure 6 is obtained by applying the agreement procedure. Then, RRL₁₉ = < c₁ p₁ z₁ >, RRL₂₉ = < q₁ >, and RRL₃₉ = < z₁ q₁ > are retransmitted and are appended to the last of RRL₉ and RRL₁₉, respectively. Suppose that RST.PAKs are received by E₃, E₁, and then E₂, i.e. E₃ < E₁ < E₂ in ORD₉. Each Eᵢ obtains RRLᵢ as shown in Figure 7 by sorting RRLᵢ. They are selectively totally ordered. □
From Proposition 2, it is clear for the following proposition to hold.

[Proposition 4] A sublog of every RL obtained by sorting messages following the sort point after putting all the messages lost by \( E_i \) on the last of RL; is selectively totally ordered. \( \Box \)

[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. \( \Box \)

5 Evaluation

We would like to evaluate the ST protocol for a cluster \( C = (E_1, \ldots, E_n) \) in terms of the number of messages retransmitted in the presence of 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 \( E_i \) accepts only messages destined to \( E_i \) 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 \( R_d \) of messages retransmitted is given as follows.

\[
R_d = \sum_{l=1}^{L} F^{l-1}(1 - F)(L - l + 1). \tag{2}
\]

Figure 8 shows \( R_s / L \) and \( R_d / L \) for \( f \) where \( ST \) means \( d/n = 0.5 \) and \( TO \) means \( d/n = 1 \). Figure 9 shows \( R_s / L \) and \( R_d / L \) 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 \( O((d/n)^2) \) in the go-back-n one.
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 $f$](image)

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