Distributed Protocol for Selective Intra-Group Communication

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}@taklab.k-dendai.ac.jp

Abstract
In distributed applications, a group of application processes is established and the processes in the group communicate with one another, i.e. intra-group communication. Here, messages have to be reliably and causally delivered to all the destinations. In addition, the processes send messages to any subset of the group at any time. This paper presents an intra-group communication protocol which provides the group of application processes with the selective and causally ordered (SCO) delivery of messages. The SCO protocol is based on the fully distributed control scheme, i.e. no master controller, and uses the high-speed one-to-one network where messages may be lost due to the buffer overrun and congestion.

1 Introduction
Distributed applications like groupware [5] require group communications among multiple application processes. There are two kinds of group communications. The first type [17, 18, 20] is intra-group communication where after a group of processes is established, the processes communicate with one another in the group. This is the extension of the conventional connection concept among two processes to multiple processes. The processes might like to send messages to some processes, not necessarily all in the group. In the selective group communication [20], each process can send messages to any subset of the group at any time [16] discusses a selective sending-order preserving (SOP) protocol where each process can receive messages destined to the process in the sending order. [20] presents a selective totally ordering (STO) protocol where every two common destination processes of every two messages receive the messages in the same order. ISIS [2, 3] supports the second type named multicast communication, where processes can send messages to a group of processes where every process can receive the messages in the causal order. [11, 19] discuss how to deliver messages in the causal order specified at the application level.

In the distributed applications, a group of application processes have to receive messages in the causal order [2, 3, 13, 18, 24]. In this paper, we would like to discuss an intra-group communication protocol which supports a group of application processes with the selective and causally ordered (SCO) delivery of messages. While [17, 18, 20] use the broadcast network and [2, 3] use the reliable one-to-one network, the SCO protocol uses the high-speed one-to-one network like ATM [1]. In the high-speed network, each process may fail to receive messages due to the buffer overrun because the transmission speed of the network is faster than the processing speed of the process, and messages may be lost due to the congestion. In order to take advantage of the high-speed network, the communication protocols are required to be light-weighted. We approach that the communication system supports the application process with SCO delivery by recovering from the message loss and out-of-order delivery of messages by directly using the high-speed network without assuming that there exists one lower layer supporting the non-loss and ordered delivery on the network. According to advances of VLSI technologies, each process can be considered to be reliable in the distributed applications like groupware. Therefore, we can assume that the processes are reliable but messages may be lost. While [3, 12, 14, 15] discuss how to treat the process faults assuming that the network is reliable.

The sender of each message m plays a role of controller in ISIS [2, 3] and Delta-4 [24], and there is one controller named sequencer in Amoeba [9]. The reliable delivery of m means that m is received by every destination process of m. These are non-distributed approaches based on the two-phase commitment [7]. The SCO protocol adopts the distributed approach where each process has to send other processes the acceptance confirmation of messages received while the process sends the acceptance confirmation to only the controller in the non-distributed approach. That is, more messages are transmitted in the distributed approach than the non-distributed one. [17, 18, 20] show that the distributed protocols have O(n) message overhead in the broadcast network and O(n^2) in the one-to-one network for number n of processes in the group. In this paper, we would like to discuss how to reduce the number of messages in the distributed control.
That is, the acceptance confirmation is included in messages, i.e. *piggyback*. In addition, the process does not send the confirmation of messages as soon as the messages are received, i.e. *deferred confirmation*.

In section 2, the basic concepts are defined. In section 3, we discuss the data transmission procedure of the SCO protocol. In section 4, we evaluate the performance of the SCO protocol by comparing with the non-distributed protocols.

2 Basic Concepts

2.1 Layered structure

A communication system is composed of application, system, and network layers [Figure 1]. Each layer supports the higher layer with communication service through *service access points (SAPs)*. The network layer provides the system layer with high-speed data transmission [2]. The messages may be lost in the high-speed network due to the buffer overruns and congestions. A group $G$ [21] of application processes $A_1, \ldots, A_n$ is supported by system processes $S_1, \ldots, S_n$, written as $G = \{ S_1, \ldots, S_n \} \ (n \geq 2)$. $S_1, \ldots, S_n$ cooperate to support the group communication service by using the network. After $G$ is established, each $A_i$ communicates with $A_1, \ldots, A_n$ in $G$.

![Figure 1: System model](image)

2.2 Ordered delivery

Processes $P_1, \ldots, P_n$ at each layer use service provided by the underlying layer as presented in 2.1. It is important to consider in what order the underlying layer delivers messages to the processes. The causal precedence relation "$\prec$" [3] among the messages is defined based on the Lamport’s happened-before relation [2,10].

**Definition** For every pair of messages $m$ and $m'$, $m$ *causally precedes* $m'$ ($m \prec m'$) iff

1. a process $P_i$ sends $m$ before $m'$, or
2. $P_i$ sends $m'$ after receiving $m$, or
3. for some message $m''$, $m \prec m''$ and $m'' \prec m'$.

$m$ and $m'$ are *causally coincident* ($m \parallel m'$) iff neither $m \prec m'$ nor $m' \prec m$. $m \leq m'$ iff $m \prec m'$ or $m \parallel m'$.

**Definition** The underlying layer supports *selective information-preserved* delivery iff all messages are delivered to the destinations.

**Theorem** The underlying layer supports *causally ordered* delivery iff for every pair of messages $m$ and $m'$, every $P_i$ receives $m$ before $m'$ if $m \prec m'$. □

Figure 2 shows the data transmission among four processes $P_1, P_2, P_3$, and $P_4$. $m_4 \prec m_2 \prec m_3 \prec m_4$ because $P_1$ sends $m_2$ after $m_1$, $P_2$ sends $m_3$ after receiving $m_2$, and $P_3$ sends $m_4$ after receiving $m_3$. $P_4$ receives $m_4$ after $m_2$ if the causally ordered delivery is supported.

![Figure 2: Causally ordered delivery](image)

**Definition** The underlying layer supports *selective causally ordered* (SCO) delivery if it is selectively information-preserved and causally ordered. □

The high-speed one-to-one network does not support the SCO delivery, i.e. the messages may be lost or not be causally delivered. In this paper, we would like to discuss how to support a group of application processes with the SCO service by using the high-speed one-to-one network.

2.3 Acceptance levels

The system processes send and receive messages by using the network. We would like to discuss how a system process $S_j$ accepts a message $m$ received from $S_i$ in the presence of message loss.

- $S_j$ simply accepts $m$ iff $S_j$ takes $m$ on receipt of $m$ if $m$ is destined to $S_j$.
- $S_j$ continuously accepts $m$ iff $S_j$ simply accepts $m$ and all the messages which $S_j$ sends before $m$.
- $S_j$ causally accepts $m$ iff $S_j$ simply accepts $m$ and all the messages causally preceding $m$.

Unless $S_j$ simply accepts $m$ destined to $S_i$, $S_i$ loses $m$. If $S_i$ loses a message $m'$ which $S_j$ sends before $m$, $S_i$ does not continuously accept $m$ while $S_j$ simply accepts $m$. The gap in a message sequence, i.e. message loss, can be easily detected by giving a unique sequence number to each message.

**Theorem** $S_j$ does not causally accept a message $m$ unless for every process $S_A$, there is a message $m'$ causally following $m$, i.e. $m \prec m'$, which $S_i$ continuously accepts from $S_A$. □
This theorem means that $S_i$ can causally accept $m$ after continuously accepting at least one message $m'$ from every process, where $m \leq m'$.

- $S_i$ reliably accepts $m$ iff $S_i$ knows that every destination of $m$ simply accepts $m$.

The continuously causal acceptance of $m$ means that $m$ is stably delivered without gap [13]. We assume that each message $m'$ sent by $S_k$ includes the acceptance confirmation of $m$ which $S_k$ has simply accepted before sending $m'$. Here, $m'$ is referred to as confirm $m$. If $S_i$ simply accepts messages confirming $m$ from every destination of $m$, $S_i$ reliably accepts $m$.

- $S_i$ continuously reliably accepts $m$ iff $S_i$ reliably accepts $m$ and all the messages which $S_j$ sends $m$.
- $S_i$ causally reliably accepts $m$ iff $S_i$ reliably accepts $m$ and all the messages causally preceding $m$.

It is straightforward that $S_i$ continuously reliably accepts $m$ if $S_i$ causally reliably accepts $m$. Here, $m$ is referred to as fully accepted by $S_i$ if $S_i$ could deliver $m$ to the application. In the SCO service, $S_i$ fully accepts $m$ if $S_i$ causally reliably accepts $m$. Figure 3 shows the implication relation among the acceptance levels where $\alpha \rightarrow \beta$ shows that $\alpha$ implies $\beta$.

![Figure 3: Acceptance levels](image)

$S_i$ has to causally accept $m$ in the presence of loss of messages causally preceding $m$. $S_i$ knows the loss of $m'$ which $S_j$ sends before $m$ if $S_i$ receives messages which $S_j$ sends after $m'$. By $S_j$'s retransmitting $m'$ to $S_i$, $S_i$ can continuously accept $m$.

2.4 Control schemes

There are three kinds of control schemes, i.e., centralized, decentralized, and distributed ones on how the system processes $S_1, \ldots, S_n$ reliably accept a message $m$ in the presence of message loss. Let $r$ denote the average number of destinations of message. In the centralized protocol [9], there is one controller. In the decentralized one [2, 3, 24], a sender of $m$ plays a role of the controller. They are based on the two-phase commitment protocol [7]. In Figure 4(1), $S_1$ plays a role of controller and sends a message $m_1$ to $S_2$ and $S_3$. $S_2$ and $S_3$ send $m_2$ and $m_3$, confirming $m_1$ to $S_1$. Then, $S_1$ sends $m_4$ to $S_2$ and $S_3$. Here, $S_2$ and $S_3$ reliably accept $m_3$. Totally $3r$ messages are transmitted and it takes three rounds.

In the distributed protocol, if every $S_i$ simply accepts a message $m$ from $S_j$, $S_i$ sends messages confirming $m$ to $S_j$ and all the destinations of $m$. If $S_i$ simply accepts the messages confirming $m$ from all the destinations of $m$, $S_i$ knows that $m$ has been accepted by every destination of $m$, i.e., $S_i$ reliably accepts $m$. Even if some $S_k$ loses messages confirming $m$, $S_k$ can ask another process if it is reliably accepted. If processes are not reliable, the three-phase protocol [16–18, 20–22] is required to support the reliable acceptance, i.e., $m$ is accepted only if all the destinations simply accept $m$. Figure 4(2) shows the distributed protocol. After simply accepting $p$, $S_2$ and $S_3$ send the confirmation of $m$ to the sender $S_1$ and the other destinations. Here, totally $r^2$ messages are transmitted and it takes two rounds.

Thus, the distributed approach requires more messages and less number of rounds than the centralized one. In order to decrease messages in the distributed one without increasing the number of rounds, we adopt the following strategies:

1. the acceptance confirmation of messages accepted is carried back by the message sent, and
2. each $S_i$ does not send the acceptance confirmation as soon as $S_i$ simply accepts.

Here, messages with and without data are referred to as data and control ones, respectively. After simply accepting a data message $m$ from $S_j$, $S_i$ sends a data message $m'$ with the confirmation of $m$ to the sender $S_j$ and the destinations. If $S_i$ has no data and sends back a control message, $O(r^2)$ messages are transmitted. Here, $S_i$ sends the control messages to only processes which $S_i$ has not sent the data messages for some predetermined time units without sending the control ones as soon as accepting $m$ to decrease the number of messages.

![Figure 4: Reliable acceptance](image)

Table 1 shows how to realize the acceptance levels in CBCAST of ISIS [3], CO [18], and SCO protocols. In CBCAST, the network layer supports the continuous acceptance because the network is assumed to be reliable. On the other hand, messages may be lost in the CO and SCO protocols. In order to detect messages lost, the sequence numbers of messages are used.
ISIS uses the vector clock and CO uses the sequence numbers to causally order messages.

3 Data Transmission Procedure

We present the data transmission procedure of the SCO protocol for a group \( G = \{ S_1, \ldots, S_n \} \) by using the high-speed one-to-one network.

3.1 Transmission

Messages are sent to all the processes in \( G \) and only the destinations accept the messages in the SOP [16] and STO [20] protocols since they are sent by using the broadcast network. On the other hand, messages are sent to only the destinations in \( G \) in the SCO protocol since the one-to-one network is used.

\( S_i \) sends a message \( m \) with total sequence number \( sgn \) and local sequence numbers \( lsn_1, \ldots, lsn_n \). Each time \( S_i \) sends a message \( m \), \( sgn \) is incremented by one. Here, if \( m \) is destined to \( S_j \), \( lsn_j \) is incremented by one. \( j = 1, \ldots, n \). \( m \) has a field \( dst \) which denotes the destination processes in \( G \).

\( S_i \) manipulates variables \( SQN, LSN_1, \ldots, LSN_n, SQN \) denotes \( sgn \) of the message which \( S_i \) expects to send next. \( LSN_j \) shows \( lsn_j \) of the message which \( S_i \) expects to send next to \( S_j \).

3.2 Continuous acceptance

\( S_i \) manipulates a variable \( LRN_j \) which denotes \( lsn_j \) of the message which \( S_i \) expects to accept next from \( S_j \) \((j = 1, \ldots, n)\). Suppose that \( S_j \) sends a message \( m \) to \( S_i \). On receipt of \( m \), \( S_i \) simply accepts \( m \) if \( S_j \in m.dst \) and enqueues \( m \) into the receive queue \( RRQ_i \) for \( S_j \). Messages from \( S_j \) are stored in \( RRQ_i \) in the sending order. \( S_i \) continuously accepts \( m \) if \( m.lsn_j = LRN_j \). If \( m.lsn_j \neq LRN_j \), \( S_i \) finds that \( S_j \) does not accept a message \( m' \) from \( S_j \) where \( m.lsn_j > m'.lsn_j > LRN_j \), \( S_i \) requires \( S_j \) to send \( m' \) again. Then, \( S_j \) sends \( m' \) to \( S_i \). On receipt of \( m' \), \( S_i \) stores \( m' \) in \( RRQ_i \) so that the messages in \( RRQ_i \) are ordered in \( lsn_j \).

Suppose that \( S_i \) continuously accepts \( m \) from \( S_j \). \( S_i \) sends the acceptance confirmation of \( m \) to \( S_j \). The acceptance confirmation of \( m \) is carried back by messages which \( S_i \) sends to \( S_j \). \( LRN_j \), i.e., \( sgn \) of the message which \( S_j \) expects to simply accept next from \( S_i \) is stored in \( m.ack_j \) \((h = 1, \ldots, n)\). On receipt of \( m \) from \( S_j \), \( S_i \) knows that \( S_j \) has continuously accepted messages from \( S_i \) whose \( sgn < m.ack_h \).

\( S_i \) manipulates an \( n \times n \) matrix \( AL \), \( m.ack_h \) is stored in \( AL_j \) \((h = 1, \ldots, n)\) and \( m.sgn \) is in \( AL_j \) if \( m \) from \( S_j \) is accepted by \( S_i \). \( AL_j \) denotes \( sgn \) of the message from \( S_i \) which \( S_j \) knows expects to continuously accept next.

In the SCO protocol, a message \( m \) sent by \( S_j \) is sent to only the destinations in \( G \). If \( S_j \) sends no message to \( S_i \), \( S_i \) cannot know which messages \( S_j \) has accepted. \( S_i \) sends at least one message to every process every predetermined time units, i.e., control message, if \( S_i \) has no data to \( S_j \), \( S_i \) has variables \( ACC_1, \ldots, ACC_n \) to denote to which process \( S_i \) has sent the acceptance confirmation. Here, if \( ACC_j = on \), \( S_i \) has not yet sent \( S_j \) the acceptance confirmation of the message which \( S_j \) has accepted from \( S_i \). If \( S_i \) sends some message to \( S_j \), \( ACC_j := off \). If \( ACC_j \) is still on after predetermined time units, \( S_i \) sends \( S_j \) a control message with \( ack_1, \ldots, ack_n \), and then \( ACC_h := on \) for \( h = 1, \ldots, n \). If \( S_i \) itself is the destination of \( m \), \( m \) is enqued into \( RRQ_i \), and is not sent to \( S_i \).

\( S_i \) accepts and sends a message \( m \) by the following procedures.

[A]cceptance On receipt of \( m \) from \( S_j \),

\[
\begin{align*}
AL_h & := m.ack_h \quad (h = 1, \ldots, n); \\
LRN_j & := LRN_j + 1; \\
\text{for } h = 1, \ldots, n, ACC_h & := on \text{ if } S_h \in m.dst; \\
\text{if } m \text{ is enqued into } RRQ_j; \\
\text{else retransmission}; \\
\end{align*}
\]

[Trans]mission

\[
\begin{align*}
m.dst & := \text{destinations of } m; \\
m.src & := S_i; \\
m.ack_j & := LRN_j \quad (j = 1, \ldots, n); \\
m.sgn & := SQN; \\
SQN & := SQN + 1; \\
\text{for } (j = 1, \ldots, n) \{ \\
\text{if } m.lsn_j = LRN_j; \\
\text{if } S_j \in m.dst; \\
\text{LSN}_j & := \text{LSN}_j + 1; \\
\text{send } m \text{ to the destinations}; \\
\end{align*}
\]

3.3 Causal acceptance

Next, we would like to consider how \( S_i \) can causally accept the messages. Let \( m \) and \( m' \) be messages sent to \( S_j \) from \( S_i \) and \( S_h \), respectively. If every message is sent to all the processes in \( G \), more exactly speaking, if \( m'.src \in m.dst \), the following condition [18] holds.

[Causality condition] If \( m'.src \in m.dst, m < m' \) iff

(1) \( S_j = S_h \), \( m.sgn < m'.sgn \),

(2) otherwise, \( m.sgn < m'.ack_j \).

In Figure 2, \( m_{2}.sgn \nless \less m_{4}.ack_1 \) since \( m_{1}.sgn \leq m_{2}.sgn \) and \( m_{4}.ack_1 = m_{1}.schn + 1 \). Thus, the causality condition does not hold unless \( P_{3} \in m.dst \). To order messages in \( \prec \), each message \( m \) sent by \( S_j \) carries the causal sequence numbers \( csn_1, \ldots, csn_N \), and \( S_i \) has variables \( CSN_1, \ldots, CSN_n \). On continuous acceptance of \( m \) from \( S_j \), \( S_i \) manipulates \( CSN_1, \ldots, CSN_n \), as follows.

[Causality rule]

(1) \( CSN_j := m.sgn + 1 \) if \( S_j = m.src \).

(2) \( CSN_j := \text{max}(CSN_h, m.csn_h) \) (for \( h = 1, \ldots, N, h \neq j \)).

CSN_j denotes \( sgn \) of the message from \( S_i \) which \( S_j \) expects to receive next. Here, \( CSN_h \geq AL_h \) \((h = 1, \ldots, n)\). Thus, the causality number is derived from the total sequence numbers while the vector clock [13] is derived from the local clock. When \( S_i \) sends a message \( m, m.csn_h := CSN_h \) \((h = 1, \ldots, n) \) in the transmission procedure.

The messages accepted can be causally ordered in the receive queue by the following theorem.

[Theorem] For every pair of messages \( m \) and \( m' \), \( m < m' \) iff

(1) \( m.sgn < m'.sgn \) if \( m.src = m'.src \),
Table 1: Acceptance levels

<table>
<thead>
<tr>
<th>acceptance level</th>
<th>ISIS (CBCAST)</th>
<th>CO (broadcast)</th>
<th>SCO (point-to-point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>network service</td>
<td>network service</td>
<td>network service</td>
</tr>
<tr>
<td>causal</td>
<td>network service</td>
<td>sequence number</td>
<td>sequence number</td>
</tr>
<tr>
<td>reliable</td>
<td>decentralized</td>
<td>distributed</td>
<td>distributed</td>
</tr>
</tbody>
</table>

(2) \( m_{\text{sn}} < m_{1}.c_{\text{snj}} \) for \( S_j = m_{\text{src}} \) otherwise.

[Proof] If \( m \) and \( m' \) are sent by the same process, it is clear. Suppose that \( m \) is sent by \( S_j \) and \( m' \) is sent by \( S_k \). First, suppose that \( m < m' \). From the causality rule, if \( S_k \) sends \( m'' \) after receiving \( m, m_{\text{sn}} < m''.c_{\text{snj}} \). Thus, \( m''.c_{\text{snj}} < m'.c_{\text{snj}} \). Hence, \( m_{\text{sn}} < m'.c_{\text{snj}} \).

Next, suppose that \( m_{\text{sn}} < m'.c_{\text{snj}} \). By the causality rule, there is a message \( m'' \) such that \( m < m'' \) and \( m'' < m' \). Hence, \( m < m' \). \( \Box \)

If \( S_j \) simply accepts a message \( m' \) from \( S_i \) where \( m'.l_{\text{sn}} > L_{\text{sn}} \), \( S_i \) finds that \( S_j \) has not continuously accepted a message \( m \) where \( L_{\text{sn}} < m'.l_{\text{sn}} \), \( m' \) is enqueued into \( RRQ_j \), and \( m \) is transmitted again. The messages in \( RRQ_j \) are ordered in \( m_1, \ldots, m_h \) where \( m_{\text{sn}} < m_{\text{sn}} \) for \( h < k \).

Here, let \( PRRQ_j \) be a maximally continuous prefix \( \langle m_1, \ldots, m_h \rangle \) of \( RRQ_j \) \((h \leq l) \) where \( m_h \) is continuously accepted for every \( k < h \) and \( m_{h+1} \) is not if \( h < l \). That, \( S_i \) does not accept messages sent after \( m_h \) before \( m_{h+1} \) by \( S_j \).

\( S_i \) moves messages continuously accepted in \( RRQ_1, \ldots, RRQ_n \) to a causality queue \( CRQ \) by the following procedure. In \( CRQ \) messages are causally ordered according to Theorem 2.

[Causally ordering procedure]

while \( (PRRQ_j \neq \emptyset \) for \( j = 1, \ldots, n \) \}{

1. \( S_j \) finds the top message \( m \) of some \( PRRQ_j \) where \( m < m' \) for the top \( m' \) of every other \( PRRQ_j \).

2. If \( m \) is found, \( m \) is dequeued from \( RRQ_j \) and enqueued into \( CRQ \). If there is the top \( m' \) of some \( PRRQ_j \) such that \( m || m' \), \( m' \) is also moved into \( CRQ \). \( \Box \)

If some \( PRRQ_j \) is empty, i.e. no message continuously accepted in \( RRQ_j \), \( S_i \) moves no message to \( CRQ \). \( S_i \) waits to causally accept \( m \) from \( S_j \) until \( S_i \) continuously accepts a message \( m' \) such that \( m < m' \) from every process.

[Example] Let us consider a group \( G = \langle S_1, S_2, S_3, S_4 \rangle \) as shown in Figure 6. Here, \( s_k(s_1, s_2, s_3, s_4) \) shows a message where \( s_{\text{sn}} = k \) and \( c_{\text{sn}} = s_i \) \((i = 1, \ldots, 4) \). Suppose that initially \( SQN = 1 \) in every process. \( S_k \) continuously accepts messages \( c_1 \) from \( S_3, a_1 \) from \( S_1 \), and \( c_2 \) from \( S_3, S_4 \) sends \( d_1 \) to \( S_3 \) and \( S_1 \). At (1) of Figure 6, \( S_3 \) has four receipt queues \( RRQ_1 = \langle a_1 \rangle, RRQ_2 = \langle b_2 \rangle, RRQ_3 = \langle c_1, c_2 \rangle \), and \( RRQ_4 = \langle d_1 \rangle \). Since \( PRRQ_2 \) is empty, no message is removed from any receipt queue. Then, \( S_4 \) continuously accepts \( a_2 \) from \( S_1 \), and \( b_2 \) from \( S_2 \). At (2), \( RRQ_1 = \langle a_1, a_2 \rangle, RRQ_2 = \langle b_2 \rangle, RRQ_3 = \langle c_1, c_2 \rangle \), and \( RRQ_4 = \langle d_1 \rangle \). The top messages \( a_1, b_2, c_1, \) and \( d_1 \) in the receipt queues are compared on \( csn \). Here, \( PRRQ_4 = RRQ_4 \) since there is no message loss \((i = 1, \ldots, 4) \). \( a_1 \) and \( c_1 \) are removed from \( RRQ_1 \) and \( RRQ_3 \), respectively, and are enqueued into \( CRQ \) since \( a_1 || c_1, a_1 < b_2, \) and \( a_1 < d_1 \). Here, \( RRQ_1 = \langle a_2 \rangle, RRQ_2 = \langle b_2 \rangle, RRQ_3 = \langle c_2 \rangle \), and \( RRQ_4 = \langle d_1 \rangle \). The tops \( a_2, b_2, c_2, \) and \( d_1 \) in the receipt queues are compared again. Since \( c_2 || d_1, c_2 < a_2, \) and \( c_2 < b_2, c_2 \) and \( d_1 \) are moved into the causality queue \( CRQ \). Here, \( CRQ = \langle a_1, a_2, c_2, d_1 \rangle \) where \( a_1 \leq c_2 \) and \( c_2 < a_1 \). Here, \( CRQ \) might be \( \langle a_1, c_1, c_2, d_1 \rangle, \langle b_1, c_1, d_1, d_2 \rangle \), or \( \langle c_2, d_1, c_2 \rangle \) since \( a_1 || c_1, c_2 || d_1 \).

![Figure 6: Example](image-url)
accepted. □

**Proposition 3** For every message $m$ in $CRQ$, $S_i$ causally accepts $m$.

**Proof** It is clear from Theorem 1. □

Each time a message $m$ from $S_j$ is moved to $CRQ$, $AL_{ij} := m.sqn$. All the messages from every $S_j$ whose $sqn < AL_{ij}$ are causally accepted by $S_i$.

### 3.4 Full acceptance

$S_i$ does not yet know whether messages in $CRQ$ are reliably accepted by the destinations. Let $m$ be a message accepted by $S_i$ from $S_j$ and $\text{min}_i\{AL_j\} = m.sqn$. All the messages from every $S_j$ whose $sqn < \text{min}_i\{AL_j\}$ are causally accepted. Hence, if $m.sqn < \text{min}_i\{AL_j\}$, $m$ is causally accepted. $m$ is fully accepted since $m$ can be delivered to the application.

**Theorem 4** If $S_i$ reliably accepts $m$, $m$ is eventually reliably accepted by every destination of $m$.

**Proof** Suppose that $S_i$ reliably accepts $m$ but $S_j$ does not. $S_j$ loses a message $m'$ confirming $m$ from some destination $S_k$ of $m$. By the time out mechanism of the acceptance procedure, $S_h$ retransmits $m'$ to $S_k$. Then, $S_j$ reliably accepts $m$. □

**Theorem 5** For every message $m$ in $ARQ$, $S_i$ fully accepts $m$.

**Proof** From Proposition 1 and Theorem 4, the messages in $ARQ$ are causally reliably accepted. □

$S_i$ has one acknowledgment queue $ARQ$ in which messages causally reliably accepted, i.e. fully accepted, are stored. While the top message of $CRQ$, where $m.dst = S_j$, satisfies $m.sqn < \text{min}_i\{AL_j\}$, $m$ is dequeued from $CRQ$ and enqueued into $ARQ$.

**Full acceptance**

while ( $m.sqn < \text{min}_i\{AL_j\}$ )

while ( $m.sqn < \text{min}_i\{AL_j\}$ )

$S_i$ sends the acceptance confirmation to the coordinator if they succeed in simply accepting $m$. The coordinator sends the acceptance confirmation of $m$ if all the destinations receive $m$, otherwise sends the failure to them. Hence, simply accepts 3r messages are transmitted and it takes three rounds for each message to be reliably accepted as shown in Figure 4(1).

In the distributed protocol, after accepting a message $m$ from $S_i$, each destination $S_j$ of $m$ sends the acceptance confirmation to $S_j$ and all the destinations as shown in Figure 4(2). $r^2$ messages are transmitted and it takes two rounds for each message to be reliably accepted. In order to reduce the number of messages, the confirmation of $m$ is carried back by the messages sent by $S_i$. $S_i$ does not send the confirmation of $m$ as soon as simply accepting $m$, i.e. deferred confirmation. $S_i$ sends the confirmation to $S_j$ if $S_j$ does not send messages to $S_i$ for predetermined time units. The same modification can be adopted to the centralized protocol. The distributed and non-distributed protocols with piggyback and deferred confirmation are named modified distributed and non-distributed.
ones, respectively.

Four control schemes, non-distributed \((C_0)\), modified non-distributed \((C_1)\), distributed \((D_0)\), and modified distributed \((D_1)\) ones are compared in terms of number of messages and delay to fully accept a message. \(D_1\) means the SCO protocol. Figures 7 and 8 show the ratios of the number of messages and the delay of \(C_1\), \(D_0\), and \(D_1\) to \(C_0\) where \(n=10\). Here, we assume that every process \(S_i\) sends a \((\geq 1)\) messages every one time unit. We also assume that \(S_i\) sends the control messages only to the processes to which \(S_i\) does not send messages in \(k\) \((\geq 1)\) time units. One round is \(t\) time units. Figures 7 and 8 show the ratios for \(k\), where \(a=1\), \(t=4\), and \(r=5\), i.e. each process sends one \((a)\) data message every time unit and sends control messages if there are processes to which \(S_i\) has sent no data message for \(k\) time units, and it takes four \((t)\) time units to deliver a message to the destination. Figure 7 shows that the longer \(k\) is, the lower number of control messages are transmitted but the longer delay it takes. However, \(D_1\) does not require much longer delay than \(D_0\) and the delay of \(D_1\) is much smaller than \(C_0\) and \(C_1\). \(C_1\) has the minimum number of messages but the longest response time, about two times longer than \(D_1\).

Figures 9 and 10 show the ratios of the number of messages and the delay for the number \(r\) of the destinations where \(a=1\), \(k=4\), and \(t=4\). \(D_1\) has lower number of messages than \(C_0\) and \(D_0\). For \(r=3\) to 8, \(C_1\) has lower number of messages than \(D_1\) but the difference between \(C_1\) and \(D_1\) is smaller than 20% of \(D_1\). The delay of \(D_1\) is 50% smaller than \(C_0\) and \(C_1\) and does not get much greater than \(D_0\) while the deferred confirmation is used in \(D_1\).

In summary, \(C_1\) has the lowest number of messages but the longest delay. \(D_0\) has the shortest delay but the largest number of messages. \(D_1\), i.e. SCO supports the second lowest number of messages and the second shortest delay, but the difference from the best one is small, i.e. smaller than 10%. Hence, \(D_1\) can support the better feature than the others in terms of number of messages and delay.

5 Concluding Remarks

This paper has discussed the intra-group communication (SCO) protocol which supports the causally ordered and selective delivery of messages to the destinations in the group. The SCO protocol uses the high-speed one-to-one network where the system processes may not receive messages due to buffer overrun and congestion. Messages are sent to only destinations in the group while they are sent to all the processes in [20]. The SCO protocol is based on the distributed control, where each process sends messages to the sender and destinations carrying the acceptance confirmation of the messages which the process has accepted. In order to reduce the number of messages, the SCO protocol adopts the deferred confirmation and the piggy back. We have shown that lower number of messages are transmitted and it takes shorter delay in the SCO protocol than the non-distributed ones.

References

Figure 10: Delay (n = 10)


