マルチメディア通信と分散処理 70−19
（1995. 5. 26）

分散型プロトコルを用いた選択的グループ通信

立川 敬行　滝沢 誠

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

分散型応用システムでは，複数のプロセス間でのグループ通信が必要となる．グループ通信では，グループ内の全プロセスがメッセージを受信するという原子性と，各プロセスがどのような順序でメッセージを受信するかという順序性を提供する必要がある．このような事象間の順序関係を，因果関係という．本論文では，グループ内の全プロセスに対して，因果関係順に，メッセージを受信させるグループ通信プロトコルについて論じる．本プロトコルは，バッファオーバランによりメッセージの紛失が起こり得る高速通信網を利用する．また，指揮プロセスの存在しない完全分散型の制御方式に基づいている．受信メッセージの因果関係による順序付けは，メッセージの通番を用いて行うために，メッセージの紛失を検出し，復旧できる．

Distributed Protocol for Selective Intra-group Communication

Takayuki Tachikawa and Makoto Takizawa

Tokyo Denki University

In distributed applications, a group of application processes is established and the processes in the group communicate with each other. i.e. intra-group communication. Here, messages have to be atomically delivered to all the destinations and be causally ordered. 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 selective and causally ordered (SCO) delivery of messages. The SCO protocol is based on the fully distributed control scheme.

1 Introduction

Distributed applications require group communications among multiple application processes. One kind of group communication [6–8] is *intra-group* communication where a group of processes is established and the processes communicate with one another in the group. In the *selective* group communication [8], each process can send messages to any subset of the group at any time. [5] discuses a *selective sending-order preserving* (*SOP*) protocol where each process can receive messages destined to the process in the sending order. [8] discusses a *selective totally ordering* (*STO*) protocol where every two common destination processes of every two messages receive the messages in the same order by using the broadcast network. ISIS [1] supports *multicast* where processes can send messages to pre-defined groups of processes where every two common processes in every two groups can receive the messages sent to both of the groups in the same order.

In this paper, we would like to discuss an intra-group communication protocol supporting the selective and causally ordered (*SCO*) delivery of messages. While [6–8] use the broadcast network and [1] use the reliable one-to-one network, the SCO protocol uses the high-speed one-to-one network. In the high-speed network, messages may be lost due to the buffer overrun and congestion. According to advances of VLSI technologies, each process can be considered to be reliable. Therefore, we can assume that the processes are reliable but messages may be lost.

The sender of each message p [1, 10] or *sequencer* [2] decides on the atomic and ordered delivery of p to all the processes in the non-distributed approaches. The SCO protocol adopts the distributed control. Here, each process has to send other processes the acceptance confirmation of messages received. That is, more messages are transmitted in the distributed control than the non-distributed one. In order to decrease messages, the *piggy back* and *deferred confirmation* are adopted.

In section 2, 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 Selective causal order

A communication system is composed of *application*, *system*, and *network* layers. The network layer provides the system layer with high-speed data transmission service. System process

E_i sends messages to other processes by using the network layer. In the high-speed network, E_i may fail to receive messages due to the buffer overrun and congestion. A group $\mathcal{G}$ [?] of application processes A_1, ..., A_n is supported by E_1, ..., E_n, written as $\mathcal{G} = \langle E_1, \ldots, E_n \rangle$ $(n \geq 2)$.

Processes P_1, ..., P_n at each layer use service provided by the underlying layer. We model the service of the underlying layer as a set of *logs*. A log L is a sequence of messages, denoted as $\langle m_1 \ldots m_k \rangle$. m_t *precedes* m_u in L $(m_t \rightsquigarrow_L m_u)$ iff $t < u$. P_i has a sending log SL_i and receipt log RL_i, which are sequences of messages sent and received by P_i, respectively $(i = 1, ..., n)$. $s_i[m]$ and $r_i[m]$ denote the sending and receipt events of message m by process P_i, respectively.

The causal precedence relation "$\prec$" [1] among the messages is defined based [3].

[Definition] For messages m and m' sent by P_i and P_j, respectively, m *causally precedes* m' $(m \prec m')$ iff $s_i[m] \rightarrow s_j[m']$. □

m and m' are *causally concurrent* $(m \parallel m')$ if neither $m \prec m'$ nor $m' \prec m$. "$\prec$" is transitive. $m \preceq m'$ iff $m \prec m'$ or $m \parallel m'$.

m *precedes* m' in SL_i $(m \rightsquigarrow_{SL_i} m')$ if $s_i[m] \rightarrow s_i[m']$. Here, m *locally precedes* m' in P_i. m *precedes* m' in RL_i $(m \rightsquigarrow_{RL_i} m')$ if $r_i[m] \rightarrow r_i[m']$.

[Definition]

(1) RL_i is *selectively information-preserved* iff RL_i includes all the messages in SL_1, ..., SL_n which are destined to P_i.

(2) RL_i is *local-order-preserved* iff for every pair of messages m and m' sent by P_j in RL_i, $m \rightsquigarrow_{RL_i} m'$ if $m \rightsquigarrow_{SL_j} m'$.

(3) RL_i is *causally preserved* iff for every pair of m and m' in RL_i, $m \rightsquigarrow_{RL_i} m'$ if $m \prec m'$. □

Figure 1 shows the data transmission among four processes E_1, E_2, E_3, and E_4. $t \prec p \prec r \prec q$ because E_1 sends p after t, E_2 sends r after receiving p, and E_3 sends q after receiving r. Since E_4 receives q after p, $RL_4 = \langle p\ q \rangle$ is causally preserved. That is, $r_4[p] \rightarrow r_4[q]$ since $p \prec q$.

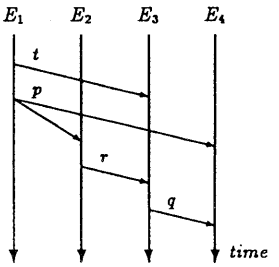


Figure 1: Causally preserved receipt

[Definition] *Selective causally ordering* (*SCO*) service is one where every RL_i is selectively information-preserved and causally preserved. □

The high-speed one-to-one network is local-order-preserved but not information-preserved, i.e. messages may be lost.

2.2 Acceptance levels

There are levels on how system process E_i accepts message p from E_j.

- E_i *simply* accepts p iff E_i takes p on receipt of p if p is destined to E_i.
- E_i *continuously* accepts p iff E_i simply accepts p and all the messages locally preceding p in E_j.
- E_i *causally* accepts p iff E_i simply accepts p and all the messages causally preceding p.

Unless E_i simply accepts p destined to E_i, E_i *loses* p. If E_i loses q locally preceding p in E_j, E_i can accepts simply but not continuously p.

[Theorem 1] For every process E_h and message p, if there is message q $(\preceq p)$ which E_i continuously accepts from E_h, E_i causally accepts p. □

- E_i *atomically* accepts p iff E_i knows that every destination of p simply accepts p.

A message q sent by each E_h includes the confirmation of p which E_h has simply accepted before sending q. Here, q is referred to as *confirm* p. If E_i simply accepts messages confirming p from every destination of p, E_i atomically accepts p.

- E_i *continuously atomically* accepts p iff E_i atomically accepts p and all the messages locally preceding p in E_j.
- E_i *causally atomically* accepts p iff E_i atomically accepts p and all the messages causally preceding p.

E_i can pass them to the application in the causal precedence order. p is *fully* accepted by E_i if E_i could pass p to the application. In Figure 2 , $\alpha \rightarrow \beta$ shows that α implies β.

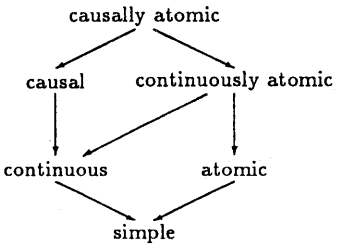


Figure 2: Implication of acceptance

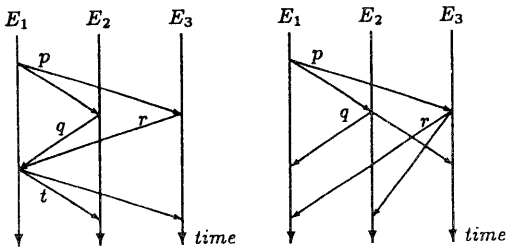
E_i has to causally accept p in the presence of loss of messages causally preceding p. If E_i loses q locally preceding p in E_j, E_i knows the loss of q when E_i receives messages locally following p in E_j. By E_j's retransmitting q to E_i, E_i can continuously accept p.

2.3 Control schemes

There are three kinds of schemes, i.e. *centralized*, *decentralized*, and *distributed* ones on how

to coordinate the cooperation among the system processes E_1, ..., E_n. In the non-distributed protocols [1, 2, 10], one controller or a sender of message p plays a role of the controller. They are based on the *two-phase commitment* protocol. In Figure 3(1), E_1 plays a role of controller and sends message p to E_2 and E_3. Totally $3d$ messages are transmitted and it takes three rounds for number d of destinations.

In the distributed protocol, every E_i makes decision on the atomic and ordered delivery of message by the cooperation with other processes. If E_i simply accepts q from E_j, E_i knows that E_j has simply accepted every message p such that $r_j[p] \rightarrow s_j[q]$. If E_i simply accepts the message confirming p from all the destinations of p, E_i knows that E_i atomically accepts p. Even if some E_k does not receive messages confirming p, E_k can ask another if p is atomically accepted. Figure 3(2) shows the distributed control. Here, totally d^2 messages are transmitted and it takes two rounds.



(1) Non-distributed control (2) Distributed control

Figure 3: Atomic delivery

In order to decrease messages in the distributed control without increasing the delay time, we adopt the following strategies:

(1) the acceptance confirmation is carried back by the message, and

(2) each E_i does not send the acceptance confirmation as soon as E_i receives messages.

Here, messages with and without data are *data* and *control* ones, respectively. After accepting data message p from E_j, E_i sends data q with the confirmation of p to E_j and the destinations if there is data to send. If there is no data to E_k (E_j or the destination of p), E_k does not receive the confirmation of p. While no additional control message is transmitted to E_k, it takes longer to atomically accept p. E_i sends the control messages to the processes which E_i has not sent the data messages for some time units.

3 Data Transmission

We present the data transmission procedure of the SCO protocol for a group $\mathcal{G} = \langle E_1, ..., E_n \rangle$ by using the high-speed one-to-one network.

3.1 Transmission

Messages are sent to only the destinations in $\mathcal{G}$ since the one-to-one network is used. E_i sends message p with *total* sequence number sqn and *local* sequence numbers lsn_1, ..., lsn_n. Each time E_i sends message, sqn is incremented by one. Each time E_i sends message to E_j, lsn_j is incremented by one. If the message is not destined to E_j, lsn_j is not incremented ($j = 1, ..., n$). p has field dst denoting the destinations in $\mathcal{G}$.

E_i has variables SQN, LSN_1, ..., LSN_n. SQN denotes sqn of message which E_i expects to send next. LSN_j shows lsn_j of message which E_i expects to send next to E_j. E_i constructs message p by the following procedure.

[Transmission]

$p.dst$:= destinations of p; $p.src := E_i$;
$p.sqn := SQN$; $SQN := SQN + 1$;
for ($j = 1, ..., n$) { $p.lsn_j := LSN_j$;
if $E_j \in p.dst$, $LSN_j := LSQ_j + 1$; } □

3.2 Continuous acceptance

E_i has variable LRN_j which denotes lsn_j of message which E_i expects to receive next from E_j ($j = 1, ..., n$). Suppose that E_j sends message p to E_i. On receipt of p, E_i simply accepts p if $E_i \in p.dst$ and enqueues p into a receipt queue RRQ_j for E_j. Messages from E_j are stored in RRQ_j in the sending order. E_i continuously accepts p if $p.lsn_i = LRN_j$. If $p.lsn_i \neq LRN_j$, E_i finds that E_i does not receive message q from E_j where $p.lsn_i > q.lsn_i \geq LRN_j$. E_i requires E_j to send again. On receipt of q from E_j, E_i stores q in RRQ_j.

Suppose that E_i continuously accepts p from E_j. E_i sends the acceptance confirmation of p to E_j. The confirmation of p is carried back by message which E_i sends to E_j. sqn of message which E_j expects to simply accept next from E_h is stored in $p.ack_h$ ($h = 1, ..., n$). On receipt of p from E_j, E_i knows that E_j has continuously accepted messages from E_h whose $sqn < p.ack_h$.

E_i has $n \times n$ matrix AL. $p.ack_h$ is stored in AL_{jh} ($h = 1, ..., n$) and $p.sqn$ is in AL_{ij} if p from E_j is accepted by E_i. AL_{jh} denotes sqn of message from E_h which E_j expects to continuously accept next. Hence, when E_i sends p, $p.ack_j := AL_{ij}$ ($j = 1, ..., n$) in the transmission procedure.

Unless E_j sends message to E_i, E_i cannot know which messages E_j has accepted. E_i sends at least one message to every process every some time units. E_i has variables ACC_1, ..., ACC_n to denote to which process E_i has to send the confirmation. Here, if $ACC_j = on$, E_i has not yet sent E_j the acceptance confirmation of message which E_i had accepted from E_j. If ACC_h is still on after some time units, E_i sends E_h control message with ack_1, ..., ack_n, and the$ACC_h := on$ for $h = 1, ..., n$.

On receipt of p from E_j, E_i accepts p by the following acceptance procedure.

[Acceptance procedure]
if $p.lsn_i = LRN_j$, {
$AL_{jh} := p.ack_h$ $(h = 1, \ldots, n)$;
$LRN_j := LRN_j + 1$;
for $h = 1, \ldots, n$, $ACC_h := on$ if $E_h \in p.dst$;
p is enqueued into RRQ_j; } □

3.3 Causal acceptance

Let p and q be messages sent to E_i from E_j and E_h, respectively. If every message is sent to all the processes in $\mathcal{G}$, more exactly speaking, if $q.src \in p.dst$, the following condition [7] holds.

[Causality condition] If $q.src \in p.dst$, $p \prec q$ iff
(1) if $E_j = E_h$, $p.sqn < q.sqn$,
(2) otherwise, $p.sqn < q.ack_j$. □

In Figure 1, $p.sqn \not< q.ack_j$ since $t.sqn < p.sqn$ and $q.ack_j = t.sqn + 1$. Thus, the causality condition does not hold unless $E_h \in p.dst$. A message p sent by E_j carries the *causal sequence numbers* $csn_1, \ldots, csq_n$, and E_i has variables $CSN_1, \ldots, CSN_n$. On continuous acceptance of p from E_j, E_i updates $CSN_1, \ldots, CSN_n$ as follows.

[Causality rule]
(1) $CSN_j := p.sqn + 1$ if $E_j = p.src$.
(2) $CSN_h := max(CSN_h, p.csn_h)$ (for $h = 1, \ldots, n$, $h \neq j$). □

Here, $CSN_h \geq AL_{jh}$ $(h = 1, \ldots, n)$. Thus, the causality number is derived from the total sequence numbers. When E_i sends message q, $q.csn_h := CSN_h$ $(h = 1, \ldots, n)$.

[Theorem 2] For every pair of messages p and q, $p \prec q$ iff
(1) $p.sqn < q.sqn$ if $p.src = q.src$,
(2) $p.sqn < q.csn_j$ for $E_j = p.src$ otherwise.

If E_i simply accepts message q from E_j where $q.lsn_i > LRN_j$, E_i finds that E_i has not continuously accepted message p where $LRN_j \leq p.lsn_i < q.lsn_i$. q is enqueued into RRQ_j and p is transmitted again. For $RRQ_j = [p_1, \ldots, p_m]$, let $PRRQ_j$ be a *maximally continuous prefix* $[p_1, \ldots, p_h]$ of RRQ_j $(m \leq h)$ where p_k is continuously accepted for every $k \leq h$ and p_{h+1} is not if $h < m$.

E_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 1.

[Causally ordering procedure]
while ($PRRQ_j \neq \phi$ for every j=1, ..., n) {
(1) E_i finds the top p of some $PRRQ_j$ where $p \prec q$ for the top q of every other $PRRQ_h$.
(2) p is moved from RRQ_j to CRQ. If there is the top q of some RRQ_h such that $p \parallel q$, q is also moved into CRQ. } □

[Example] Let us consider a group $\mathcal{G} = \{E_1, E_2, E_3, E_4\}$ as shown in Figure 4. Here, $m_k(l_1, l_2, l_3, l_4)$ shows message where $sqn = k$ and $csn_i = k_i$ $(i$=1, ..., 4). Suppose that initially $SQN = 1$ in every process. E_4 continuously accepts c_1 from E_3, a_1 from E_1, and c_2 from E_3. E_4 sends d_1 to E_3 and E_4. At (1) of Figure 4, E_4 has $RRQ_1 = \langle a_1 \rangle$, $RRQ_2 = \langle \phi \rangle$, $RRQ_3 = \langle c_1\ c_2 \rangle$, and $RRQ_4 = \langle d_1 \rangle$. Since $PRRQ_2 = \phi$, no message is removed from any receipt queue. E_4 continuously accepts a_2 from E_1, and b_2 from E_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 tops a_1, b_2, c_1, and d_1 of the receipt queues are compared on csn. Here, $PRRQ_i = RRQ_i$ $(i = 1, \ldots, 4)$. a_1 and c_1 are removed from RRQ_1 and RRQ_3, respectively, and are enqueued into CRQ since $a_1 \parallel c_1$, $a_1 \prec b_2$, and $a_1 \prec d_1$. Here, $RRQ_1 = \langle a_2 \rangle$, $RRQ_2 = \langle b_2 \rangle$, $RRQ_3 = \langle c_2 \rangle$, and $RRQ_4 = \langle d_1 \rangle$. Since $c_2 \parallel d_1$, $c_2 \prec a_2$, and $c_2 \prec b_2$, c_2 and d_1 are moved into CRQ. Here, $CRQ = \langle c_1\ a_1\ c_2\ d_1 \rangle$ where $c_1 \preceq a_1 \prec c_2 \preceq d_1$. Here, CRQ might be $\langle a_1\ c_1\ c_2\ d_1 \rangle$, $\langle a_1\ c_1\ d_1\ c_2 \rangle$, or $\langle c_1\ a_1\ d_1\ c_2 \rangle$ since $a_1 \parallel c_1$ and $c_2 \parallel d_1$.

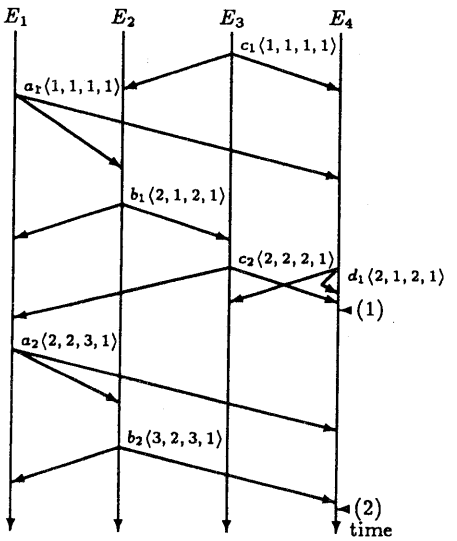


Figure 4: Example

Here, suppose that E_4 loses a_1. E_4 finds the loss of a_1 on acceptance of a_2. $PRRQ_1$ is empty while $RRQ_1 = \langle a_2 \rangle$. Hence, no message in RRQs is moved to CRQ. On receipt of a_1, E_4 obtains the same receipt queues as (2). Then, the messages are causally accepted. □

[Proposition 3] For every message p in CRQ, E_i causally accepts p. □

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

3.4 Full acceptance

Let p be message accepted by E_i from E_j and $minAL_j(p)$ be $min(\{AL_{hj} \mid E_h \in p.dst\})$. If $p.sqn < minAL_j(p)$, p is causally atomically accepted.

[Theorem 4] If E_i atomically accepts p, p is eventually atomically accepted by every destination.

[Theorem 5] For every message p in ARQ, E_i

fully accepts p. □

E_i has one *acknowledgment* queue ARQ in which messages causally atomically accepted, i.e. fully accepted are stored. While the top p of CRQ, where $p.dst = E_j$, satisfies $p.sqn < minAL_j(p)$, p is dequeued from CRQ and enqueued into ARQ.

[Full acceptance]

while ($p.sqn < minAL_j(p)$ for the top p of CRQ, where p is sent by E_j) {
    p is dequeued from CRQ and
    enqueued into ARQ. } □

3.5 Flow control

E_i includes the number of available buffers in the field buf of p, i.e. $p.buf := BUF_i$. On acceptance of q from E_j, E_i knows how many available buffers E_j has and $BUF_j := q.buf$. Let $minBF(p)$ denote $min\{\ BUF_h \mid E_h \in p.dst\ \}$. Each E_i can send message p only while the following flow condition is satisfied. Here, W is the maximum window size and H is constant (> 1).

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

4 Evaluation

Table 1 shows how to realize the acceptance levels in CBCAST [1], CO [7], and SCO protocols. In CBCAST, the network layer supports the continuous acceptance i.e. reliable. While messages may be lost in CO and SCO. In order to continuously accept messages, the sequence numbers of messages are used. While CBCAST uses the vector clock [4], CO uses the vector of sequence numbers assuming that the messages are sent to all the processes in the group. SCO uses the vector clock derived from the sequence numbers of the messages. CBCAST and AMp are decentralized. CO and SCO are distributed ones.

We assume that each process sends messages randomly to d ($\leq n$) processes in a group $\mathcal{G} = \langle\ E_1,\ ...,\ E_n\ \rangle$. In the non-distributed, one coordinator C sends message p to the destinations in $\mathcal{G}$ and the destinations send back the reply to C if they succeed in accepting p. C sends the confirmation of p if all the destinations receive p, otherwise sends the failure to them. Hence, $N(d) = 3d$ messages are transmitted and it takes three rounds for each message to be atomically accepted as shown in Figure 3(1).

In the distributed control, after accepting message p from E_j, each destination E_i of p sends the confirmation to E_j and all the destinations as shown in Fig 3(2). $D(d) = d^2$ messages are transmitted and it takes two rounds E_i sends the confirmation to E_k if E_i does not send message to E_k for some time units, i.e. *deferred confirmation*. The distributed and non-distributed controls with piggy back and deferred confirmation are named modified distributed and centralized controls, respectively.

The non-distributed (C_0), modified non-distributed (C_1), distributed (D_0), and modified distributed (D_1) schemes are compared in terms of number of messages and delay time to fully accept message. D_1 means the SCO protocol. Figures 5 and 6 show the ratios of the number of messages and delay time of C_1, D_0, and D_1 to C_0 where n=10. Here, we assume that every E_i sends a (≥ 1) messages every one time unit. We also assume that E_i sends the control messages to only the processes to which E_i does not send messages in k (≥ 1) time units. One round is r time units. In Figures 5 and 6, a=1, k=4, r=4, and d=5. Figure 5 shows that the longer k is, the less control messages are transmitted and the longer delay time it takes. However, D_1 does not require much longer delay time than D_0 and the delay time of D_1 is much smaller than C_0 and C_1. C_1 has the minimum number of messages but the largest response time, about two times longer than D_1. Figures 7 and 8 show the ratios of the number of messages and delay time for d. D_1 has less messages than C_0 and D_0. For d = 3 to 8, C_1 has less messages than D_1 but the difference between C_1 and D_1 is smaller than 20% of D_1. The delay time 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 least messages but the largest delay time. D_0 has the shortest delay time but the largest number of messages. D_1, i.e. SCO supports the second least messages and the second smallest delay time, but the difference from the best one is small, i.e. smaller than 20%. Hence, D_1 can support the better feature than the others in terms of number of messages and delay time.

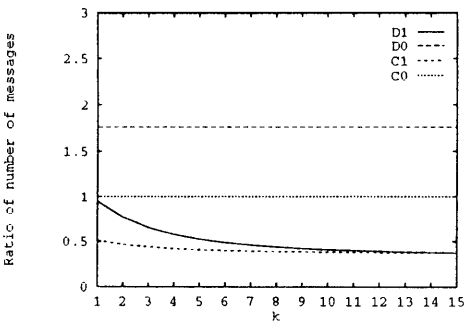


Figure 5: Number of messages (n = 10)

5 Concluding Remarks

This paper has discussed the *distributed* intra-group communication (SCO) protocol supporting the causally ordered and selective delivery of messages to the destinations in the group. The

Table 1: Acceptance levels

acceptance level	ISIS (CBCAST)	CO (broadcast)	SCO (point-to-point)
simple	network service	network service	network service
continuous	network service	sequence number	sequence number
causal	vector clock	sequence number	vector clock
atomic	decentralized	distributed	distributed

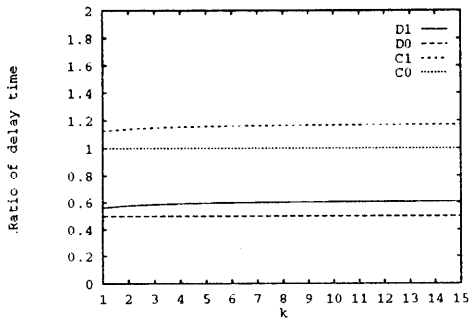


Figure 6: Delay time ($n = 10$)

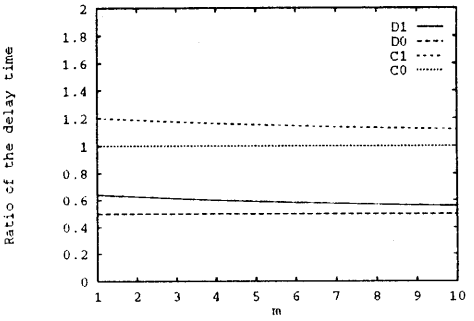


Figure 8: Delay time ($n = 10$)

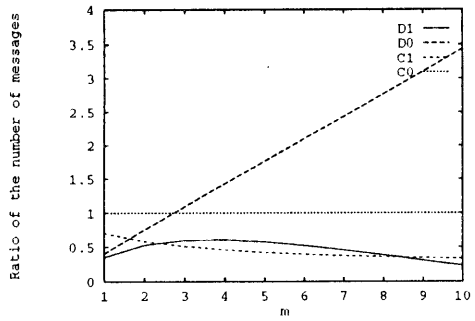


Figure 7: Number of messages ($n = 10$)

SCO protocol uses the high-speed one-to-one network. Messages are sent to only destinations in the group . In order to reduce the number of messages, the SCO protocol adopts the deferred confirmation and piggy back. We have shown that less messages are transmitted and it takes less delay time in the SCO protocol than the non-distributed control.

References

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

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

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

[4] Mattern, F., "Virtual Time and Global States of Distributed Systems," *Parallel and Distributed Algorithms, Cosnard, North-Holland*, 1989, pp.215-226.

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

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

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

[8] Tachikawa, T. and Takizawa, M., "Selective Total Ordering Broadcast Protocol," *Proc. of IEEE ICNP94*, 1994, pp.212-219.

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

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