

Two-Layered Protocol for a Large-Scale Group of Processes*

KOJIRO TAGUCHI AND MAKOTO TAKIZAWA

Department of Computers and Systems Engineering

Tokyo Denki University

Hatoyama, Hiki-gun, Saitama 350-0394, Japan

E-mail: {tagu,taki}@takilab.k.dendai.ac.jp

A group including a larger number of processes implies larger computation and communication overheads $O(n^2)$ required to manipulate and transmit messages for a number n of processes. In this paper, we discuss a group which is composed of subgroups of processes. Each subgroup has a gateway process which communicates with the other gateway processes. We propose a mechanism to causally deliver messages to processes in a group by using a vector of message sequence numbers whose size is the number of subgroups, not the number of processes. We assume that networks are less-reliable; i.e., that messages may be lost.

Keywords: distributed systems, group communications, causality, large-scale group, two-layered group

1. INTRODUCTION

Multiple processes cooperate to achieve some objectives in distributed applications like teleconferencing. A *group* means a set of multiple processes which cooperate on the basis of *peer-to-peer* cooperation of processes. A *group* does not imply client-server cooperation. In these applications, hundreds of processes cooperate are distributed on not only local area but also wide area networks. A *large-scale* group is a group which includes hundreds of processes. Each communication channel between processes may not support the same Quality of Service (QoS). A *wide-area* group is a group where the processes are distributed in wide-area networks like the Internet. Tachikawa and Takizawa [14, 15] discuss protocols for wide-area groups which adopt fully distributed control and destination retransmission, and examine an international experiment conducted by Japan, USA, and Europe.

A group communication protocol supports a group of $n (> 1)$ processes with causally/totally ordered delivery of messages [1, 2, 8, 9]. In order to support the ordered delivery of messages, a *vector clock* [2, 8] including n elements is used, assuming that the underlying networks are reliable; that is to say, each message is delivered to its destination without message loss and in a sending order. Here, the header length of messages is $O(n)$ for the number n of processes in the group. $O(n^2)$ computation and communication overheads are required to send and receive messages in a group of n processes. Even if a group of tens of processes can be realized by traditional group protocols, it is

Received May 15, 2002; accepted July 25, 2002.

Communicated by Biing-Feng Wang, Stephan Olariu and Gen-Huey Chen.

* A preliminary version of the paper was presented at the 2002 International Conference on Parallel and Distributed Systems, Chungli, Taiwan.

difficult, maybe impossible to support a group of hundreds of processes due to the large computation and communication overheads. In order to reduce the computation and communication overheads, *hierarchical* groups have been studied [5, 16]. Papers [3, 5] discuss how to multicast messages in tree routings but do not discuss ordered delivery of messages. Takamura and Takizawa [16] discuss how to support the causally ordered delivery of messages in a hierarchical group by using the vector clock, but the vector size is the total number of processes. In this paper, a group is composed of multiple disjointed subgroups, each of which includes processes in a local area. Subgroups are interconnected by the Internet. We discuss a *two-layered group (TG)* protocol for a large-scale, wide-area group of processes. Messages are causally ordered by using a type of vector clock whose size is the number of subgroups, which is smaller than the total number of processes. Furthermore, we assume that the underlying networks are less reliable; i.e., messages may be lost and may be delivered out of order. The TG protocol supports the causally ordered delivery of messages during detection and recovery from message loss.

In section 2, we present a system model. In section 3, we discuss the causally ordered delivery of messages in a two-layered group. In section 4, we discuss the two-layered group (TG) protocol. In section 5, we evaluate the TG protocol in terms of delay time compared with the traditional flat group.

2. SYSTEM MODEL

2.1 Groups

In a *group* of multiple processes, the processes cooperate in order to achieve some objectives in a distributed system. In one-to-one communication and multicast communication [3], each message is *reliably* delivered to one and more than one process. On the other hand, multiple processes first establish a *group* in group communication. Then, a process autonomously sends a message to multiple processes while receiving messages from multiple processes in the group. Lamport [7] defines a *happens before* relation among events occurring in multiple processes. A sending event of a message m *happens before* a receiving event of m . An event *happens before* another event if the events occur in this sequence in the same process. When the transitive *happens before* relation is used, a causally precedent relation among messages can be defined: a message m_1 *causally precedes* another message m_2 ($m_1 \rightarrow m_2$) iff a sending event of m_1 *happens before* a sending event of m_2 [2]. A process is required to deliver a message m_1 before another message m_2 if m_1 causally precedes m_2 ($m_1 \rightarrow m_2$).

Due to computation and communication overheads $O(n^2)$ for a number n of processes in a group, it is difficult to support a group with a large number of processes using a group communication service, such as causally ordered delivery of messages. According to practical experiments [11, 14], more than several tens of processes cannot cooperate in a group from the performance point of view. In one approach to reducing the overheads, a group G is decomposed into disjointed subgroups G_1, \dots, G_k . Each subgroup G_i is composed of processes and a *gateway* process p_{i0} . If process p_{is} in subgroup G_i sends a message m to destination processes in another subgroup G_j ($j \neq i$), then

process p_i first sends message m to gateway process p_{i0} in subgroup G_i . Then, gateway process p_{i0} forwards message m to gateway process p_{j0} of destination subgroup G_j . Gateway process p_{j0} delivers message m to the destination processes in the subgroup G_j . Group G is referred to as *two-layered* [Fig. 1]. A group is *flat* iff every pair of processes in the group directly exchange messages. For example, processes in a subgroup are interconnected in a local area network. A pair of gateway processes are interconnected on the Internet. In this paper, we discuss a group communication protocol for a two-layered group.

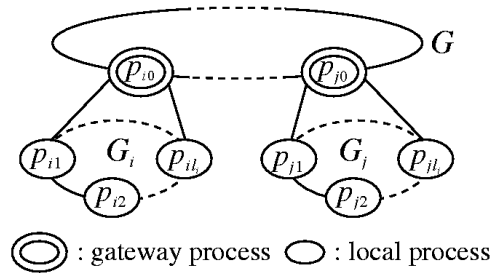


Fig. 1. Two-layered group.

Processes in each subgroup directly communicate with each other by using a basic communication service supported by underlying networks like local area networks. A pair of gateway processes exchange messages on the Internet. Each communication channel between processes can be realized by means of TCP [4] and UDP [12]. Most group protocols like ISIS [2] assume that the underlying network is reliable. We have designed a group protocol based on the assumption that networks are less reliable; i.e., messages may be lost and may be delivered out of a sending order.

2.2 Distributed Protocols

It is important to discuss which process coordinates communication among processes in a group. In a *centralized* coordination approach [5, 6], there is one controller in a group. Every process first sends a message to the controller, and then the controller delivers the message to all the destination processes in the group. The delivery order of messages is decided by the controller. Thus, the messages easily can be totally, i.e., causally ordered. In a *distributed* approach, there is no centralized controller. Every process directly sends messages to the destination processes and directly receives messages from every other process in a group. Each process makes a decision on the delivery order and automatic receipt of messages by itself, e.g., by using the vector clock [8]. ISIS [2] uses a *decentralized* approach, where every destination process sends a receipt confirmation to the sender of a message in a reliable underlying network. Takizawa *et al.* [10, 11, 15] use a *fully distributed* approach, where every destination process sends a receipt confirmation to not only the sender, but also all the other destinations in less-reliable networks.

2.3 Confirmation Vector

For a group G of n (> 1) processes p_1, \dots, p_n , vector V is in the form $\langle V_1, \dots, V_n \rangle$ [8]. Every process p_i has a vector $V = \langle V_1, \dots, V_n \rangle$, where each element V_j is initially 0 ($j = 1, \dots, n$). Each time process p_i sends message m , the i th element V_i is incremented by one. Then, message m carries vector $V(m.V)$ of the sender process p_i . Upon receipt of message m from another process, $V_k := \max(V_k, m.V_k)$ ($k = 1, \dots, n, k \neq i$) in a process. Here, for a pair of vectors $A = \langle A_1, \dots, A_n \rangle$ and $B = \langle B_1, \dots, B_n \rangle$, $A \leq B$ iff $A_j \leq B_j$ ($j = 1, \dots, n$). Message m_1 causally precedes message m_2 ($m_1 \rightarrow m_2$) iff $m_1.V \leq m_2.V$. Message m_1 is *causally concurrent* with message m_2 ($m_1 \parallel m_2$) iff neither $m_1 \rightarrow m_2$ nor $m_2 \rightarrow m_1$.

The confirmation vector $CV = \langle CV_1, \dots, CV_n \rangle$ of message sequence numbers is used to detect message loss in group protocols [10, 11]. A sequence number seq is incremented by one in process p_i each time process p_i sends a message. Message m carries a sequence number seq as $m.seq$. Process p_i manipulates variable CV_j , which shows a sequence number seq of a message which process p_i expects to receive next from process p_j ($j = 1, \dots, n$). Each message m carries the confirmation vector $m.CV$ ($= \langle m.CV_1, \dots, m.CV_n \rangle$). On receipt of message m from process p_j , process p_i *accepts* message m if $CV_j = m.seq$. Then, the j th element CV_j in the confirmation vector CV is incremented in process p_i , i.e., $CV_j := CV_j + 1$. The confirmations $m.CV_1, \dots, m.CV_n$ are stored in an *acknowledgment matrix* AK as $AK_{jk} := m.CV_k$ ($k = 1, \dots, n$). Message m received from process p_j is *pre-acknowledged* in process p_i if $m.seq < \min(AK_{1j}, \dots, AK_{nj})$; i.e., process p_i knows that every other process has accepted message m . If every message is destined for all the processes, then message m_1 causally precedes message m_2 ($m_1 \rightarrow m_2$) iff $m_1.CV < m_2.CV$ [11].

Next, we will discuss the case when a pre-acknowledged message is to be delivered in process p_i . Process p_i can deliver a pre-acknowledged message m if every message causally preceding message m is delivered and process p_i receives from every process a pre-acknowledged message causally preceded by message m . Here, message m is *acknowledged* in process p_i . Process p_i is sure that message m is pre-acknowledged in every process; i.e., every process knows that every other process accepts message m .

On receipt of message m from process p_j , if $CV_j < m.seq$, then process p_i finds a message gap; i.e., process p_i loses message m' from process p_j , where $CV_j \leq m'.seq < m.seq$. Next, suppose process p_k sends message m_1 to a pair of processes p_i and p_j , but that process p_i fails to receive message m_1 . After receiving message m_1 , process p_j sends message m_2 to process p_i , where $m_2.CV_k = m_1.seq + 1$. Process p_i receives message m_2 , where $CV_k < m_2.CV_k$, and finds that process p_i has not received message m_1 from process p_k . Thus, process p_i discovers the loss of message m from another process p_k upon receipt of message m' from process p_j if $CV_k \leq m.seq < m'.CV_k$ ($k \neq j$).

3. CAUSALLY ORDERED DELIVERY IN A TWO-LAYERED GROUP

A two-layered group (TG) G is composed of multiple subgroups G_1, \dots, G_k ($k > 1$). Each subgroup G_i includes processes p_{i1}, \dots, p_{il_i} ($l_i > 1$) and one gateway process p_{i0} . Processes and messages transmitted in a subgroup are referred to as *local*. A *main sub-*

group is composed of gateway processes p_{i0}, \dots, p_{k0} , where *global messages* are exchanged. If local message m is destined for a process in another subgroup, then m is an *outgoing* local message. An outgoing local message m sent in subgroup G_i is changed into global message M . Then, global message M is transmitted in a main subgroup. Global message M is changed into local message m_j in destination subgroup G_j . Here, a pair of local messages m and m_j are the *source* and *destination* local messages of a global message M , $sl(M)$ and $dl_j(M)$, respectively. A small letter m indicates a local message. A capital letter M indicates a global message for local message m . Let $dl_j(m)$ denote a destination local message of a source local message m in subgroup G_j . Let $sl(m)$ be a source local message of a destination local message m . Let $g(m)$ denote a global message of a local message m .

A notation " $M_1 \rightarrow_G M_2$ " (M_1 globally causally precedes M_2 in group G) shows that global message M_1 causally precedes global message M_2 among the gateway processes in a main subgroup of group G . A notation " $m_1 \rightarrow_i m_2$ " (m_1 locally causally precedes m_2) indicates that local message m_1 causally precedes local message m_2 in subgroup G_i . We define a causally precedent relation among local messages by using the globally and locally precedent relations \rightarrow_G and \rightarrow_i as follows:

Definition: Local message m_1 causally precedes local message m_2 ($m_1 \rightarrow m_2$) iff $sl(m_1) \rightarrow_i sl(m_2)$, $dl_i(m_1) \rightarrow_i sl(m_2)$ in some subgroup G_i , or $m_1 \rightarrow m_3 \rightarrow m_2$ for some local message m_3 . \square

The following property holds for the causally precedent relation \rightarrow and globally causally precedent relation \rightarrow_G :

Theorem 1 $g(m_1) \rightarrow_G g(m_2)$ if $m_1 \rightarrow m_2$.

Proof: This is straightforward from the definition. \square

Suppose group G includes a pair of subgroups G_i and G_j , whose gateway processes are p_{i0} and p_{j0} , respectively. Local process p_{is} in subgroup G_i sends local message m_1 to local process p_{jt} in subgroup G_j . Local process p_{jt} sends local message m_2 before receiving destination local message m_1' ($= dl_j(m_1)$) of m_1 and receives local message m_3 after receiving local message m_1' as shown in Fig. 2. Since gateway process p_{j0} sends global message M_2 to gateway process p_{i0} after receiving global message M_1 , M_1 globally causally precedes M_2 in the main subgroup G ($M_1 \rightarrow_G M_2$). However, m_1 is causally concurrent with m_2 ($m_1 \parallel m_2$). " $M_1 \rightarrow_G M_2$ " if " $m_1 \rightarrow m_2$ " from Theorem 1. However, " $m_1 \rightarrow m_2$ " does not necessarily hold even if $M_1 \rightarrow_G M_2$. We have to discuss a mechanism for ordering only a pair of global messages M_1 ($= g(m_1)$) and M_2 ($= g(m_2)$) such that " $m_1 \rightarrow m_2$ " holds. That is, unless $m_1 \rightarrow m_2$, a pair of global messages M_1 and M_2 can be delivered in any order.

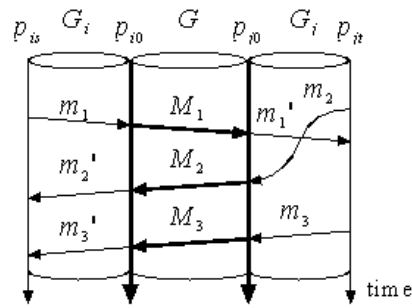


Fig. 2. Two-layered group (TG).

4. TG PROTOCOL

We will next discuss a group protocol for one type of two-layered group, a *broadcast two-layered group* (B-TG) G , where every process sends messages to all the processes. We assume that networks are less reliable; i.e., messages may be lost due to communication faults like congestion and unexpected delay. We will discuss a basic data transmission procedure to causally order messages in a broadcast two-layered group G . Group G is composed of subgroups G_1, \dots, G_k ($k > 0$). Each subgroup G_i is composed of gateway process p_{i0} and local processes p_{i1}, \dots, p_{il} ($l_i > 0$). Each local message m to be transmitted in subgroup G_i includes the following fields:

$m.seq$ = local sequence number;
 $m.sg$ = source subgroup G_i ;
 $m.sp$ = source process p_{ij} in $m.sg$;
 $m.rsq$ = local confirmation vector $\langle rsq_0, rsq_1, \dots, rsq_{l_i} \rangle$ ($l_i > 0$);
 $m.RSQ$ = global confirmation vector $[RSQ_1, \dots, RSQ_k]$ ($k > 0$);
 $m.data$ = data.

Each global message M includes the following fields:

$M.GSQ$ = global sequence number;
 $M.SG$ = sender subgroup;
 $M.SP$ = source process in $M.SG$;
 $M.RSQ$ = global confirmation vector $[RSQ_1, \dots, RSQ_k]$;
 $M.DATA$ = data.

Each local process p_{ij} in subgroup G_i manipulates the following variables to receive and send local messages:

seq = local sequence number;
 $rsq = \langle rsq_0, rsq_1, \dots, rsq_{l_i} \rangle$;
 $RSQ = [RSQ_1, \dots, RSQ_k]$;
 $ack = l_i \times l_i$ acknowledgment matrix.

Each gateway process p_{i0} manipulates the following variables to receive and send global messages:

GSQ = global sequence number;
 $ACK = k \times k$ acknowledgment matrix.

The sequence numbers GSQ and seq are initially 1, and each element in global confirmation vector RSQ and local confirmation vector rsq are also initially 1 in every process. First, local process p_{is} in subgroup G_i sends a source local message m as follows:

$m.sp := p_{is}$;
 $m.sg := G_i$;
 $m.seq := seq$;
 $seq := seq + 1$. $m.rsq_s := seq$. $m.rsq_u := rsq_u$ ($u = 0, 1, \dots, l_i, u \neq s$);
 $RSQ_i := RSQ_i + 1$. $m.RSQ := RSQ$.

Then, gateway process p_{i0} receives outgoing local message m from source local process p_{is} in subgroup G_i . Here, variables of local confirmation vector rsq and matrix ACK are manipulated in the gateway process p_{i0} as follows:

$rsq_s := rsq_s + 1$;
 $ack_{su} := m.rsq_u$ ($u = 0, 1, \dots, l_i$).

Gateway process p_{i0} sends all the gateway processes global message $M(= g(m))$, which is created from the outgoing local message m as follows:

$M.SG := m.sg$;
 $M.SP := m.sp$;
 $M.GSQ := GSQ$;
 $GSQ := GSQ + 1$;
 $M.RSQ_h := m.RSQ_h$ ($h = 1, \dots, k, h \neq i$);
 $M.RSQ_i := GSQ$;
 $M.DATA := m.data$.

Next, suppose gateway process p_{j0} in subgroup G_j receives global message M from subgroup G_i . Here, variables of global confirmation vector RSQ and matrix ACK are manipulated in gateway process p_{j0} as follows:

$RSQ_i := RSQ_i + 1$;
 $ACK_{ih} := M.RSQ_h$ ($h = 1, \dots, k$).

Gateway process p_{j0} sends all the processes in subgroup G_j destination local message $m_j(= dl(M))$, created from global message M as follows:

$m_j.sp := M.SP$;
 $m_j.sg := M.SG$;

$$\begin{aligned}
m_j.seq &:= seq. \quad seq := seq + 1; \\
m_j.rsq_0 &:= seq. \quad m_j.rsq_u := rsq_u \quad (u = 1, \dots, l_j); \\
m_j.RSQ &:= M.RSQ; \\
m_j.data &:= M.DATA.
\end{aligned}$$

Local process p_{jt} receives destination local message m_j from gateway process p_{j0} according to the following procedure:

$$\begin{aligned}
rsq_0 &:= rsq_0 + 1; \\
ack_{0u} &:= m_j.rsq_u \quad (u = 0, 1, \dots, l_j); \\
RSQ_h &:= \max(RSQ_h, m_j.RSQ_h) \quad (h = 1, \dots, k).
\end{aligned}$$

If local process p_{it} receives local message m from local process p_{is} in the same subgroup G_i , then local and global confirmation vectors rsq , and RSQ , and matrix ack are manipulated in local process p_{it} as follows:

$$\begin{aligned}
rsq_s &:= rsq_s + 1; \\
ack_{su} &:= m.rsq_u \quad (u = 0, 1, \dots, l_i); \\
RSQ_h &:= \max(RSQ_h, m.RSQ_h) \quad (h = 1, \dots, k).
\end{aligned}$$

Local messages received are causally ordered in a local process according to the following ordering rule:

Ordering rule 1 Local message m_1 locally *precedes* local message m_2 in subgroup G_i ($m_1 \Rightarrow_i m_2$) if $m_1.rsq < m_2.rsq$ and $m_1.RSQ < m_2.RSQ$. \square

Theorem 2 If local message m_1 causally precedes local message m_2 ($m_1 \rightarrow m_2$), then m_1 locally precedes m_2 in subgroup G_i ($m_1 \Rightarrow_i m_2$) according to ordering rule 1.

Proof: Suppose $m_1 \rightarrow m_2$ but $m_1 \not\Rightarrow_i m_2$. If $m_1 \rightarrow m_2$, then $g(m_1) \rightarrow_G g(m_2)$ according to Theorem 1. If $g(m_1) \rightarrow_G g(m_2)$, then $m_1 \Rightarrow_i m_2$. This contradicts the assumption. \square

Global messages received are causally ordered in a gateway process according to the following ordering rule:

Ordering rule 2 Global message M_1 globally *precedes* global message M_2 in main subgroup ($M_1 \Rightarrow_G M_2$) if $M_1.RSQ < M_2.RSQ$. \square

Theorem 3 If global message M_1 globally causally precedes global message M_2 in main subgroup ($M_1 \rightarrow_G M_2$), then $M_1 \Rightarrow_G M_2$ according to ordering rule 2.

Proof: Suppose $M_1 \rightarrow_G M_2$ but $M_1 \not\Rightarrow_G M_2$. If $M_1 \rightarrow_G M_2$, then $dl_i(m_1) \Rightarrow_i dl_i(m_2)$ according to ordering rule 1. If $dl_i(M_1) \Rightarrow_i dl_i(M_2)$, then $M_1 \Rightarrow_G M_2$ according to ordering rule 2. This contradicts the assumption. \square

Even if global message M_1 globally causally precedes global message M_2 in main subgroup G ($M_1 \rightarrow_G M_2$), the causally precedent relation “ $m_1 \rightarrow m_2$ ” does not necessarily

hold for local messages m_1 and m_2 of global messages M_1 and M_2 , respectively. Suppose gateway process p_{i0} receives outgoing local messages m_1 and m_2 from local processes p_{i1} and p_{i2} in subgroup G_i , respectively. Gateway process p_{i0} creates global messages M_1 and M_2 from outgoing local messages m_1 and m_2 , respectively. Gateway process p_{i0} sends global message M_1 before global message M_2 if local message m_1 causally precedes local message m_2 . Here, suppose a pair of local messages m_1 and m_2 are locally causally concurrent in subgroup G_i ($m_1 \parallel_i m_2$). In the TG protocol, each time gateway process p_{i0} sends global message M , $M.RSQ_i := GSQ$ and $GSQ := GSQ + 1$. If gateway process p_{i0} receives local message m_1 before local message m_2 , $M_1.RSQ < M_2.RSQ$; i.e., global message M_1 globally precedes global message M_2 ($M_1 \Rightarrow^G M_2$). Thus, for a pair of local messages m_1 and m_2 sent in the same subgroup G_i , global message $g(m_1)$ may globally precede global message $g(m_2)$ even if m_1 parallel m_2 .

Theorem 4 If local message m_1 causally precedes local message m_2 ($m_1 \rightarrow m_2$) and $m_1.sg \neq m_2.sg$, then global message $g(m_1)$ globally precedes global message $g(m_2)$ ($g(m_1) \Rightarrow^G g(m_2)$) in G according to the ordering rules.

Proof: This is straightforward from the definitions and Theorem 3. □

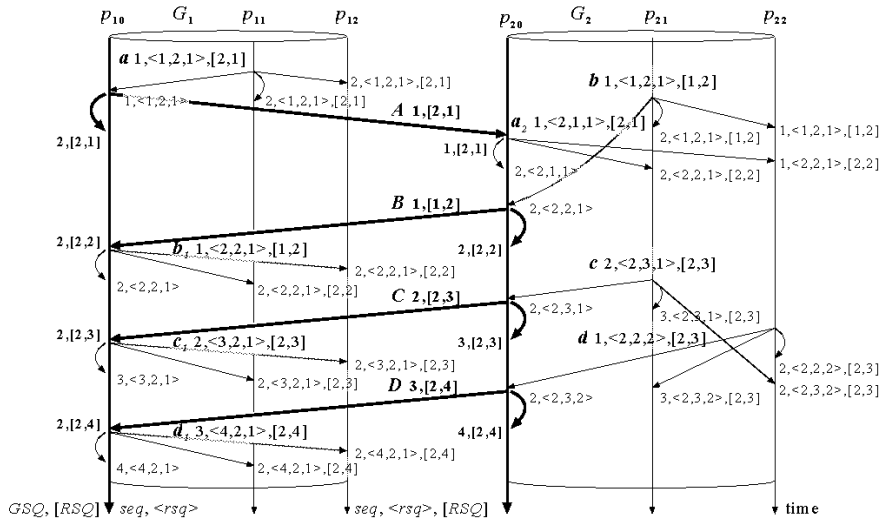


Fig. 3. Communication among subgroups G_1 and G_2 .

Example. Fig. 3 shows group G , composed of two subgroups G_1 and G_2 , where a pair of processes p_{10} and p_{20} are gateway processes. First, local process p_{11} in subgroup G_1 sends a source local message a to all the processes in subgroup G_1 . Here, $a.seq = 1$, $a.rsq = \langle 1, 2, 1 \rangle$, and $a.RSQ = [2, 1]$. Local message a is sent to gateway process p_{10} . Gateway process p_{10} creates global message A from local message a . Here, $A.GSQ = 1$ and $A.RSQ = [2, 1]$. Gateway process p_{10} sends global message A with $A.RSQ = [2, 1]$ to all the gateway processes in the main subgroup.

The global confirmation vector RSQ in gateway process p_{20} is changed to $[2, 1]$. Gateway process p_{20} sends a destination local message a_2 of global message A to all the processes in subgroup G_2 . On receipt of the destination local message a_2 , the global confirmation vector RSQ is changed to $[2, 2]$ in a pair of local processes p_{21} and p_{22} of subgroup G_2 .

Local process p_{21} sends a source local message b with $b.seq = 1$, $b.rsq = \langle 1, 2, 1 \rangle$, and $b.RSQ = [1, 2]$ before receiving the destination local message a_2 . The gateway process p_{20} sends global message B created from local message b after receiving global message A . According to the traditional definition, global message A causally precedes global message B ($A \rightarrow B$) since gateway process p_{20} sends global message B after receiving global message A . However, since local message b is sent before local message a_2 is received by local process p_{21} , a pair of global messages A and B must be causally concurrent. $A.RSQ = [2, 1]$, while $B.RSQ = [1, 2]$. A pair of local messages a and b_1 are causally concurrent (a parallel b_1). $a.rsq = \langle 1, 2, 1 \rangle$ and $a.RSQ = [2, 1]$ while $b.rsq = \langle 2, 2, 1 \rangle$ and $b.RSQ = [1, 2]$. According to the ordering rules, neither global messages A and B nor local messages a and b_1 are ordered. From Theorem 4, global message C globally precedes global message D ($C \rightarrow_G D$) even if a pair of local messages c and d are causally concurrent in subgroup G_2 because a pair of local messages c and d are sent in the same subgroup.

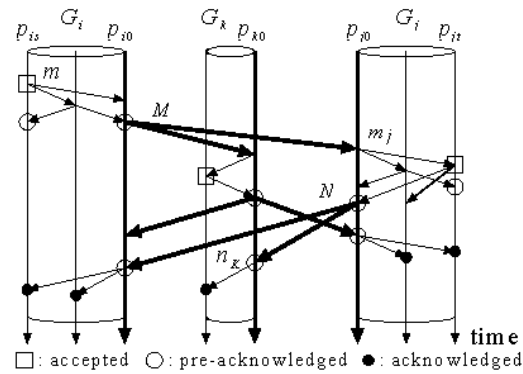


Fig. 4. Data transmission.

In each subgroup G_i , the vectors of message sequence numbers are used to causally order messages and detect message loss. First, local process p_{i1} sends message m in subgroup G_i [Fig. 4]. After receipt of local message m , another local process sends a message confirming receipt of local message m . Gateway process p_{i0} forwards global message $M (= g(m))$ to other gateway processes. On receipt of global message M , gateway process p_{j0} sends local message $m_j (= dl_j(M))$. On receipt of local message m_j , every local process p_{jt} sends a message confirming receipt of local message m_j . If local message m_j is pre-acknowledged in gateway process p_{j0} , gateway process p_{j0} sends global message N with confirming receipt of global message M . If global message M is pre-acknowledged in gateway process p_{k0} , gateway process p_{k0} sends local message n_k confirming receipt of local message m . On receipt of local message n_k , local message m

is pre-acknowledged in every process of subgroup G_k . In each local process, messages are ordered according to ordering rule 1 by the local confirmation vector rsq and the global confirmation vector RSQ as discussed in the preceding section. If a process loses a message m in a subgroup, one process which accepts that message m forwards it to a process which fails to receive m .

5. EVALUATION

The two-layered group (TG) protocol is evaluated here in terms of time needed to deliver a message. The delay time is the sum of the processing time in the gateway and local processes and the communication delay time. The following parameters are used to evaluate the protocols:

- n = number of processes in a group G ;
- k = number of subgroups;
- l_i = number of local processes in each subgroup G_i ;
- δ_F = delay time in a flat group;
- δ_T = delay time in a two-layered group.

In the TG protocol, the size of a global confirmation vector RSQ is k ($< n$), and the size of a local confirmation vector rsq is l_i ($< n$) in subgroup G_i . The overhead of each local process in subgroup G_i is on the order of $(l_i + k) l_i$, $O((l_i + k)l_i)$. The overhead for communication among gateway processes is $O(k^2)$ for k subgroups. The overhead of a gateway process in subgroup G_i is $O((l_i + k)l_i + k^2)$.

It takes three rounds to deliver messages in the two-layered group, while it takes one round in the flat group. The delay time δ_T in the two-layered group was compared with the delay time δ_F in the flat group. In the evaluation, the delay time means the duration from the time when a process created a message until the time when all the processes received and manipulated the message in a group.

In the evaluation, the processes of a group were realized in a computer. A communication process was realized to simulate one-to-one and broadcast types of communication. The processes communicated with other processes through the communication process. In a flat group, we considered a pair of processes [Fig. 5]. In a two-layered group (TG) composed of k subgroups G_1, \dots, G_k , we considered four processes [Fig. 6]. Here, we assumed that every subgroup included the same number l of local processes, and that $k = \sqrt{n}$. The minimum overhead of a gateway process was obtained for $k = \sqrt{n}$.

First, suppose a process sends a message to each destination process, i.e. a one-to-one network. Here, a process sends n messages that are delivered to n processes in a flat group. On the other hand, a local process sends l local messages in a source subgroup, then a gateway process sends k global messages, and then a gateway process sends l local messages in a destination subgroup, all which happens in a two-layered group. Fig. 7 shows the delay time for n processes in a group. The figure shows that the two-layered group (TG) protocol had a shorter delay time than the flat group. For example, the delay time was reduced to 80.5 [%] for $n = 25$, 14.3 [%] for $n = 400$, and 4.5 [%] for $n = 1600$.

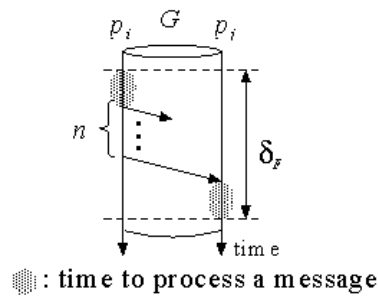


Fig. 5. Delay time in a flat group.

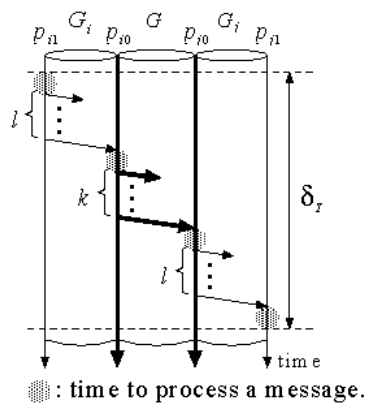


Fig. 6. Delay time in a two-layered group.

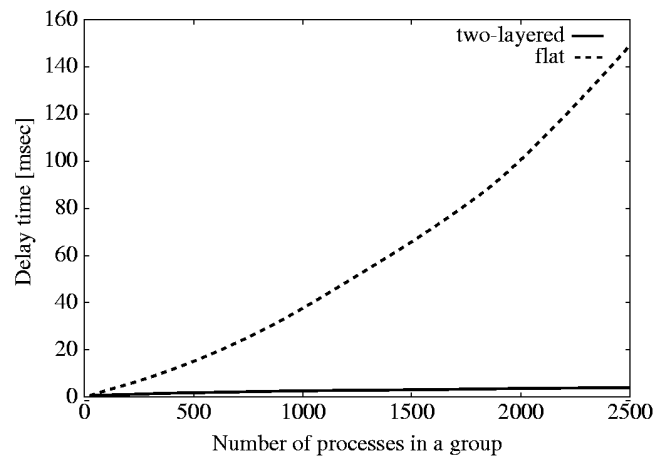


Fig. 7. Delay time in a one-to-one network.

Next, suppose a process broadcasts a message in each subgroup. That is, each local process delivers each message to all the local processes, including a gateway by means of one transmission. Fig. 8 shows the delay time for n processes in a group. If $n \geq 900$, the two-layered group had a shorter delay time than the flat group. For example, the delay time was reduced to 93.5 [%] for $n = 900$, 70.7 [%] for $n = 1225$, and 66.4 [%] for $n = 1600$.

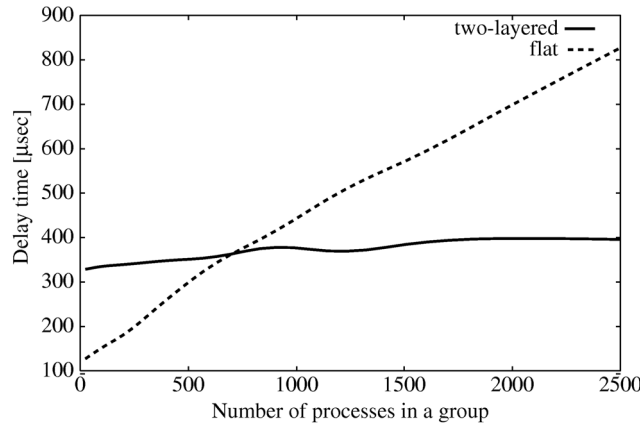


Fig. 8. Delay time in a broadcast network.

6. CONCLUDING REMARKS

We have discussed the two-layered group (*TG*) protocol for a large-scale group of processes. Here, processes in a local area network make up a subgroup. Subgroups are interconnected with a wide-area network. In the *TG* protocol, each message carries a confirmation vector, whose size is smaller than the total number n of processes. We have evaluated the *TG* protocol in terms of the delay time and compared it with the traditional flat group. We have shown that the *TG* protocol has a shorter processing time than the flat group.

REFERENCES

1. R. Baldoni, A. Mostefaoui, and M. Raynal, "Efficient causally ordered communications for multimedia real-time applications," in *Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing (HPDC-4)*, 1995, pp. 140-147.
2. K. Birman, "Lightweight causal and atomic group multicast," *ACM Transactions on Computer Systems*, 1991, pp. 272-290.
3. S. Deering, "Host groups: A multicast extension to the internet protocol," *RFC 966*, 1985, pp. 1-27.

4. Defense Communications Agency, *DDN Protocol Handbook*, Vol. 1-3, NIC 50004-50005, 1985.
5. M. Hofmann, T. Braun, and G. Carle, "Multicast communication in large scale networks," in *Proceedings of the 3rd IEEE Workshop on High Performance Communication Subsystems*, 1995, pp. 147-150.
6. M. F. Kaashoek and A. S. Tanenbaum, "An evaluation of the amoeba group communication system," in *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems (ICDCS-16)*, 1996, pp. 436-447.
7. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, Vol. 21, 1978, pp. 558-565.
8. F. Mattern, "Virtual time and global states of distributed systems," *Parallel and Distributed Algorithms*, 1989, pp. 215-226.
9. L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal, "Extended virtual synchrony," in *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems (ICDCS-14)*, 1994, pp. 56-65.
10. A. Nakamura and M. Takizawa, "Reliable broadcast protocol for selectively ordering PDUs," in *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems (ICDCS-11)*, 1991, pp. 239-246.
11. A. Nakamura and M. Takizawa, "Causally ordering broadcast protocol," in *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems (ICDCS-14)*, 1994, pp. 48-55.
12. J. Postel, "User datagram protocol," *RFC 768*, 1980, pp. 1-3.
13. L. Rodrigues and M. Raynal, "Atomic broadcast in asynchronous crash-recovery distributed systems," in *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS-20)*, 2000, pp. 288-295.
14. T. Tachikawa, H. Higaki, and M. Takizawa, "Group communication protocol for realtime applications," in *Proceedings of the 18th IEEE International Conference on Distributed Computing Systems (ICDCS-18)*, 1998, pp. 40-47.
15. T. Tachikawa, H. Higaki, and M. Takizawa, " Δ -causality and ϵ -delivery for wide-area group communications," *Computer Communications Journal*, Vol. 23, 2000, pp. 13-21.
16. M. Takizawa, M. Takamura, and A. Nakamura, "Group communication protocol for large group," in *Proceedings of the 18th IEEE Conference on Local Computer Networks (LCN)*, 1993, pp. 310-319.
17. Z. Xiao and K. Birman, "Providing efficient, robust error recovery through randomization," in *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems Workshops (ICDCS-21)*, 2001, pp. 31-36.



Kojiro Taguchi was born in 1979. He received his B.E. degree in Computers and Systems Engineering from Tokyo Denki University, Japan in 2001. He is now a graduate student of Tokyo Denki University. His research interests include group communication protocols and distributed systems.



Makoto Takizawa is a full professor in the Department of Computers and Systems Engineering, Tokyo Denki University, Japan. He is now a dean of the graduate school of Science and Engineering, Tokyo Denki University. He chaired the Information Division at the Research Institute for Technology, Tokyo Denki University from 1998 to 2002. He was a visiting professor at GMD-IPSI, Germany (1989-1990) and has been a regular visiting professor at Keele University, England since 1990. Takizawa is a fellow of Information Processing Society of Japan (IPSJ) and was a member of the executive board of IPSJ from 1998 to 2000. He chaired SIGDPS (distributed processing) of IPSJ from 1997 to 2000 and was an editor of the Journals of IPSJ (1994-1998). Takizawa received his BE and ME in applied physics, and DE in computer science from Tohoku University, Japan. In 1996, he won the best paper award at IEEE International Conference on Parallel and Distributed Systems (ICPADS). He was a general co-chair of IEEE ICDCS-2002 and a program co-chair of IEEE ICDCS-1998. He is a founder of ICOIN and AINA conferences. He was elected for 2003-2005 BoG member of IEEE Computer Society. He is a member of the IEEE and a member of the ACM and IPSJ. His research interests include distributed systems, group communication protocols, distributed objects, fault-tolerant systems, and information security.