Flexible Group Communication Protocol

Tomoya Enokido, Takao Komiya, Roziali Ghopur, and Makoto Takizawa
Department of Computers and Systems Engineering
Tokyo Denki University
E-mail {eno, komi, rozali, taki}@takilab.k.dendai.ac.jp

In a distributed application, a group of multiple processes are required to be cooperating by exchanging messages. A group protocol supports a group of multiple processes with the causally, possibly totally ordered delivery of messages. The group protocol is required to support enough QoS and types of service for applications in change of QoS supported by the underlying network and QoS requirements. A flexible protocol is composed of a collection of functions like retransmission and confirmation. There are multiple ways to realize each function. The flexible group protocol dynamically takes a type of module for each protocol function which is the most suitable for applications in change of network QoS and QoS requirement.

やわらかいグループ通信プロトコル

榎戸 智也　小宮 貴雄　ウプル ロズアリ　滝沢 誠
東京電機大学理工学部

分散システムでは、グループ内の複数のプロセスがメッセージの送受信によって協調動作を行う。グループ通信プロトコルは、グループ内の複数のプロセスに対してメッセージの送信や紛失の検出、紛失したメッセージの再送信などの様々なサービスを提供する。また、ネットワークのサービス品質 (QoS) は輻輳や機器の障害により動的に変化する。そのため、1つのグループ通信プロトコルの提供するサービスの方式では、ネットワークの QoS の変化に対して、アプリケーションの要求する QoS を常に満足することはできない。本論文では、やわらかいグループプロトコルを提案する。やわらかいグループ通信プロトコルは、メッセージの送信や紛失の検出、紛失したメッセージの再送信などの機能モジュールから構成されている。ネットワークの QoS の変化に対して、アプリケーションの要求する QoS を満足するように最適な機能モジュールを動的に選択することにより、やわらかさをアプリケーションに提供する。

1 Introduction

In group communications, multiple processes first establish a *group* and then messages are exchanged among processes. In the group, a process sends a message to multiple processes while receiving messages from multiple processes. Messages are required to be causally delivered to processes in the group [1]. A group communication protocol is realized by following functions:

1. Multicast of messages.
2. Receipt confirmation of message receipt.
3. Detection of message loss.
4. Retransmission of a messages lost.
5. Ordering of messages received.

There are various ways to realize these functions. In addition, implementation of group protocol depends on what types and quality of communication service the underlying network supports for processes. Messages sent by a process may be lost and unexpectedly delayed due to congestions and faults in the network. Thus, quality of service (QoS) like bandwidth and message loss ratio is changed. Furthermore, there are various types of networks like personal area networks [7], local area networks, and wide area networks which support different levels of QoS. For example, if the underlying network supports reliable one-to-one communications, there is no need to realize no mechanism to detect message loss and recover from the message loss. If broadcast networks like Ethernet and radio network [9] are used, a message can be sent to all the processes by one transmission. Applications require the system to support a group of processes with various types of service. For example, every message is required to be totally ordered in a group of replicas of database service, i.e. every process can receive messages in a same order. Some messages are allowed to be lost in some kinds of multimedia applications. The higher level of communication function is supported, the larger computation and communication overheads are implied. Hence, the system has to take only necessary and sufficient types of functions to support service required by application by taking usage of services supported by the underlying network. In this paper, we discuss a *flexible* group protocol which can dynamically support types and quality of service required by applications even if QoS supported by the underlying network is changed. Thus, the flexibility of the protocol is realized by dynamically selecting

types of functions in change of service supported by the underlying network.

In section 2, we present what types of service an underlying network supports. In section 3, we discuss group communication service. In section 4, we discuss what types of functions to be selected to design a group communication protocol. In section 5, we discuss how to support flexiblity by changing retransmission schemes.

2 Underlying Networks

A group of multiple processes p_1, ..., p_n ($n \geq 2$) are cooperating through exchanging messages by using underlying network service. Networks are characterized in terms of the number of destinations of each message, level of reliability, and types of ordered delivery of messages. First, there are two types of networks, *one-to-one* and *broadcast* networks, with respect to processes how many processes each message can be sent to. In the one-to-one network, a message is sent to one process by one transmission. TCP/IP [8] supports one-to-one communication service. On the other hand, a message is sent to all the processes by one transmission in the broadcast network. Ethernet and radio network [9] are broadcast types of networks.

Secondly, networks support different levels of reliability. In the *reliable* one-to-one network, messages are delivered to the destinations with neither message loss nor duplication in the sending order. A TCP connection [8] supports reliable one-to-one communication service as long as the connection exists. In a reliable broadcast network, every process receives all messages in a same order. In *less-reliable* one-to-one networks, messages may be lost. In less-reliable broadcast networks, some process does not receive a message broadcast. The Ethernet supports less reliable broadcast communication because some frames are lost due to contention.

Lastly, we discuss in what order a destination process receives messages. Suppose a process sends messages to another process. In a *sending order preserving* network, every destination process receives a message in the sending order. This is supported by a TCP connection between a pair of processes. Now, suppose multiple processes send messages to multiple processes. A process receives messages from multiple processes. In a *totally ordering* network, every process receives the messages in the same order. For example, Ethernet and radio networks support the totally ordering service while it is less-reliable. If each pair of processes communicate with one another by using a TCP connection, each process reliabily receives messages in the sending order from each process. However, a pair of processes may not receive messages from different processes in the same order.

The underlying network is modeled to be a collection of bidirectional logical communication channels, each of which exists between a pair of processes in a group. Here, notations $\langle p_i, p_j \rangle$ and $C_{i,j}$ show a channel between a pair of processes p_i and p_j. If each channel is realized by a TCP [8] connection, the network supports processes with reliable, sending order preserved, one-to-one communication service. Each channel $\langle p_i, p_j \rangle$ supports some quality of service (QoS), delay time [msec], message loss ratio [%] and bandwidth [bps]. Let Q_{ij} show QoS supported by a channel $\langle p_i, p_j \rangle$. QoS supported by each channel is changed due to congestions in the network. The change of the network is modeled to be change of QoS supported by the network.

3 Group Communication Service

A group of multiple processes p_1, ..., $p_n (n>1)$ are exchanging messages in the network. There is no centralized controller. Let $s_i(m)$ and $r_i(m)$ denote sending and receipt events of a message m in a process p_i. A message m_1 *causally precedes* another message m_2 ($m_1 \rightarrow m_2$) if and only if (iff) $s_i(m)$ *happens before* $r_j(m)$ [1,2], m_1 is *causally concurrent* with m_2 ($m_1 \parallel m_2$) if neither $m_1 \rightarrow m_2$ nor $m_2 \rightarrow m_1$. For example, suppose there are three processes p_1, p_2, and p_3 in a group G [Fig. 1]. A process p_1 sends a message m_1 to a pair of processes p_2 and p_3. The process p_2 sends a message m_2 to p_3 after receiving a message m_1. Here, m_1 causally precedes m_2 ($m_1 \rightarrow m_2$). Due to communication delay, m_1 may arrive at the process p_3 after m_2. The process p_3 is required to deliver m_1 before m_2 because $m_1 \rightarrow m_2$. A pair of messages m_1 and m_2 are *causally delivered* iff $m_1 \rightarrow m_2$ and m_1 is delivered before m_2 in every common destination of m_1 and m_2. Some messages are causally concurrent. In the *totally ordered* delivery, all the messages are delivered in every common destination of the messages in the same order. That is a pair of messages m_1 and m_2 are *totally* delivered iff m_1 and m_2 are causally delivered if $m_1 \rightarrow m_2$ or $m_2 \rightarrow m_1$, and m_1 and m_2 are delivered in a same order in every common destination of m_1 and m_2 if m_1 and m_2 are causally concurrent.

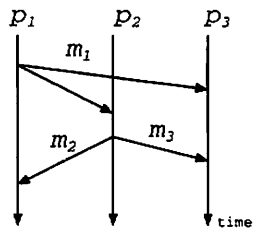

Fig. 1: Causally ordered delivery.

Messages received are ordered by every process in the distributed approach. In order to causally deliver messages, each process p_i manipulates a vector clock $V = \langle V_1, ..., V_n \rangle$ [3]. Initially, every element in the vector (V) is zero. Each time a process p_i sends a message, $V_i := V_i + 1$. Then a message m carries the vector clock $m.V$ ($=V$). On receipt of a message m, $V_j := \max(V_j, m.V_j)$ ($j = 1, ..., n$,

$j \neq i$) in a process p_i. A message m_1 causally precedes another message m_2 ($m_1 \rightarrow m_2$) iff $m_1.V < m_2.V$. Thus, the process can causally deliver messages by using the vector clock under an assumption that the underlying network is reliable. Message gaps cannot be detected by using the vector clock.

Nakamura and Takizawa [4-6] discuss a vector of sequence numbers to detect message loss and causally order messages. Each message m sent by a process p_i is assigned a sequence number $m.seq$. The sequence number seq is incremented by one each time p_i sends a message. The process p_i manipulates variables rsq_1, ..., rsq_n to correctly receive messages. Each variable rsq_j shows a sequence number seq of message which p_i expects to receive next from another process p_j (j=1, ..., n). A message m sent by p_i carries the receipt confirmation $m.rsq_j (= rsq_j)$(j=1, ..., n). Suppose a process p_i receives a message m from another process p_j. If $rsq_j = m.seq$, the process p_i accepts the message m. Otherwise, there is some message m' from p_j where $rsq_j \leq m'.seq < m.seq$, i.e. p_j fails to receive m'. If p_i accepts a message m from a process p_j, the receipt confirmation information carried by m is stored in a matrix Ack, $Ack[j,k] := m.rsq_k$ (k=1, ..., n). A message m_1 *causally precedes* another message m_2 ($m_1 \rightarrow m_2$) iff $m_1.rsq < m_2.rsq$ [6]. Suppose a process p_i accepts a message m from another process p_j. After receiving messages, the messages received are tested by using Ack if the process p_i accepts them. A message m received from a process p_j is referred to as *re-acknowledged* by a process p_i if $m.seq < min(Ack[1,j], ..., Ack[n,j])$. Here, the process p_i is sure that m is received by every process. Here, there still might be another process where m is not re-acknowledged. A message m from a process p_j is referred to as *acknowledged* iff m is re-acknowledged and there is one re-acknowledged message m_k from every process p_k where $m \rightarrow m_k$. That is, the process p_i is sure that m is re-acknowledged in every process, i.e. every process knows that every other process surely receives the message m. Here, the process p_i can deliver the message m.

4 Functions of Group Protocol

4.1 Control

There are following types of control schemes [Fig. 2]:

1. Centralized control.
2. Distributed control.

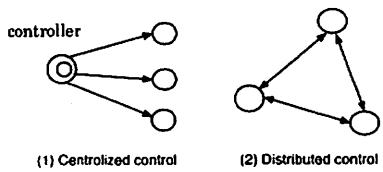

Fig. 2: Control scheme.

In the centralized control, there is one centralized controller in a group. A process first sends a message to the controller and then the controller forwards the message to the destination processes. Each destination process sends receipt confirmation to the controller if the process successfully receives the message. Then the controller sends receipt confirmation of the message to the sender if the controller receives the confirmation message from all the destination processes. Most distributed systems like current teleconference systems take this approach. It takes at least two rounds to deliver messages since every message is forwarded by the controller. On the other hand, there is no centralized controller in the distributed control scheme. Each process makes a decision on correct receipt and delivery order of messages received by itself. The vector clock [3] can be used to causally order messages in each process.

4.2 Transmission

There are following schemes to transmit a message m to multiple processes [Fig. 3]:

1. Direct transmission.
2. Indirect transmission.

In the direct transmission, each process directly sends a message to each destination, and directly receives messages from other processes [Fig. 3 (1)]. Thus, a message can be delivered to every destination by one round.

In the indirect transmission, messages are first sent to some process. The process forwards the message to another process and finally delivers the message to the destination processes. The tree routing [6] is an example of the indirect transmission. In the centralized control, the indirect transmission is adopted. It takes more than one round to deliver a message in the indirect one. The direct scheme is preferable in real-time communication because of shorter delay time. In the direct transmission, each process is required to make a decision on correct receipt and ordered delivery of messages by itself, i.e. distributed control.

4.3 Confirmation

There are following schemes to confirm the message receipt [Fig. 4]:

1. Centralized confirmation.
2. Decentralized confirmation.
3. Distributed confirmation.

In the centralized confirmation, every process sends receipt confirmation message to some process, e.g. centralized controller. After receiving confirmation messages from all the destination processes, the process sends a receipt confirmation to the sender process. In the decentralized one, a sender process plays

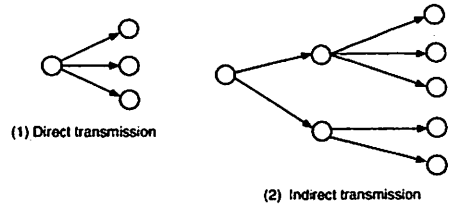

Fig. 3: Transmission.

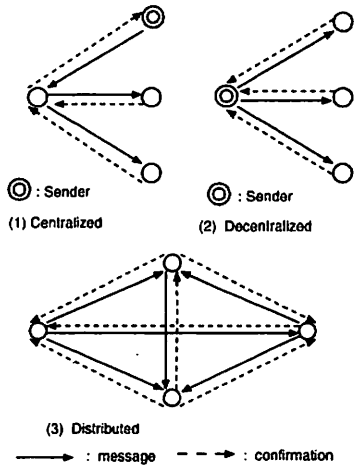

Fig. 4: Confirmation schemes.

a role of the centralized controller. That is, each destination process sends a receipt confirmation to the sender.

In the distributed confirmation scheme, each destination process sends a receipt confirmation to not only the sender process but also all the other destination processes. Since a group includes n processes, a sender process p_i sends $(n-1)$ instances of a message in the one-to-one network and one message instance in the broadcast network. Then, each destination process sends $(n-1)$ confirmation messages in the one-to-one network and one confirmation message in the broadcast network. Hence, totally $(n-1)^2$ and $(n-1)$ messages are transmitted in the one-to-one and broadcast networks, respectively. Thus, in the one-to-one and broadcast networks, communication overheads are $O(n^2)$ and $O(n)$ for number n of processes, respectively. In order to reduce the number of messages transmitted in the network, confirmation information of message receipt is *carried back* by other messages. In addition, a process does not send a confirmation message as soon as the process receives a message if the process does not have data to send. If the process receives some number of messages or it takes some time since the process has most recently received a message, the process sends the confirmation of every message which is not confirmed yet. Thus, the delayed confirmation strategy is adopted to reduce the number of messages transmitted.

Suppose a process p_i sends a message to processes p_1, ..., p_n. In the centralized confirmation, every destination process p_j sends a confirmation message to one controller process p_k if p_j is succeeded in receiving the message m. If the controller process p_k receives confirmation messages from all the destination processes, the process p_k sends a confirmation message to the sender p_i. In the decentralized confirmation, each destination process p_j sends a confirmation message back to a sender process p_i of message m. Each process p_i does not send only confirmation message each time the process receives a message. If there is no data to be sent, the process p_i sends a confirmation of the message to each destination process of the message after p_i receives some number of messages. Tachikawa and Takizawa show that the overhead of distributed way can be reduce to $O(n)$ by using these ways.

4.4 Detection of message loss

Messages are lost due to buffer overrun, unexpected delay, and congestion in the network. Message loss can be detected by checking sequence numbers as presented in the preceding section. On receipt of a message m from another process p_j, a process p_i accepts m if $rsq_j = m.seq$. Then, rsq_j is incremented by one. Otherwise, p_i finds there is some message m' from p_j where $rsq_j \leq m'.seq < m.seq$. Now suppose that a process p_i sends a message m. The message m carries a sequence number $m.seq$ and receipt confirmation $m.rsq$ $(=\langle m.rsq_1, ..., m.rsq_n \rangle)$. Here, suppose a process p_1 sends a message m_1 to a pair of processes p_2 and p_3. Here, p_2 fails to receive m_1 although p_3 accepts m_1. The process p_3 sends a message m_2 to p_2 after receiving m_1. Here, $m_2.rsq_1 \geq m_1.seq$. Now, p_2 receives m_2. Here, $rsq_1 < m_2.rsq$ in p_2 since p_2 expects to receive m_1 from p_1 i.e. $rsq_1 = m_1.seq_2$. Thus a process p_i can find loss of message m_1 from a process p_j on receipt of a message m_2 from another process p_k if $rsq_j < m_2.rsq_j (j \neq k)$.

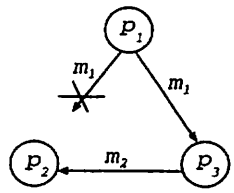

Fig. 5: Detection of message loss.

If a process selectively sends messages to a subset of processes, not necessarily all the processes, additional subsequence numbers ssq_1, ..., ssq_n are used. Here, ssq_j is incremented by one each time a process sends a message destined to a process p_j (j=1, ..., n). Each process manipulates variables $rssq_1$, ..., $rssq_n$ on receipt of a message m from a process p_j, a process p_i accepts m if $m.ssq = rssq_j$. Then $rssq_j := rssq_j + 1$. If $m.ssq_j > rssq_j$, p_i fails to receive a message m' from p_j when $rssq_j < m'ssq_j < m.ssq_j$. A sender process can detect message loss of destination process by timeout mechanism.

4.5 Retransmission

If a process p_i fails to receive a message m, m is required to be retransmitted. There are following retransmission schemes with respect to which process retransmits a message m [Fig. 6]:

1. Sender retransmission.
2. Destination retransmission.

Suppose a process p_j sends a message m to processes and one destination process p_i fails to receive

m. The first way is that the sender process p_j retransmits the message m to p_i. This is the *sender retransmission*, which is adopted by most protocols. Another way is the *destination retransmission*. Here, one or more than one destination process which has safely received the message m forwards m to the process p_i. In the distributed confirmation, not only a sender process but also every destination process receives receipt confirmation of a message m from every other destination process. Hence, each process can know if every other destination process safely receives a message m. If a destination process p_k finds that another destination process p_i has not received the message m, p_k forwards m to p_i. Here, if multiple destination process forward the message, the network traffic is increased. One process has to be selected. One way is that destination process nearest to p_j forwards the message.

5 Flexible Protocol

5.1 Architecture

The classes of the protocol functions for transmission, confirmation, retransmission, detection of message loss, and the control schemes are stored in a protocol module base (PMB). The flexible group protocol module (FGPM) takes one way for each type of group communication functions from PMB, which can support an application with necessary and sufficient QoS, given QoS supported by an underlying network. The FGPM is distributed in every process. The FGPM monitors QoS supported by the underlying network. The network QoS information monitored is stored in a QoS base (QB). If QoS is detected to be changed, the FGPM reconstructs a group protocol module (GPM) by selecting protocol function in the protocol base.

5.2 Retransmission

5.2.1 Cost model

First, we discuss which retransmission scheme the group protocol takes, i.e. sender and destination retransmission ones. Suppose a process p_s sends a message m to processes in a group G and then a pair of processes p_t and p_u receive the message m while another process p_v fails to receive m. We use following parameters:

1. d_{ij} = delay time of channel C_j between a pair of processes p_i and p_j [msec].
2. f_{ij} = probability that a message is lost in a channel C_j.

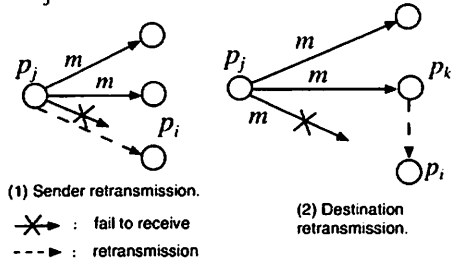

Fig. 6: Retransmission scheme.

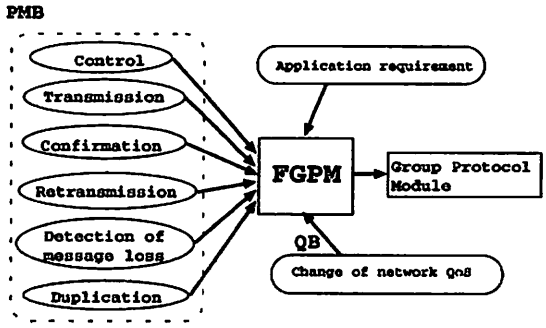

Fig. 7: Flexible group protocol.

3. b_{ij} = bandwidth of the channel C_j [bps].
4. $|m|$ = size of message m [bit] .

First, let us consider the sender retransmission. The sender p_s detects that a destination process p_v has not received the message m. It takes $(2d_{ij} + |m|/ b_{ij})$ [msec] to detect message loss after p_s sends the message m. Then p_s retransmits m to p_v. Here, the message m may be lost again. The expected time ST_{sv} and expected number SN_{sv} of message to be transmitted to deliver a message m to a destination p_v are given as follows:

1. $ST_{sv} = (2d_{sv} + |m|/ b_{sv})/(1 - f_{sv})$.
2. $SN_{sv} = 1/(1 - f_{sv})$.

In the destination retransmission, some destination process forwards the message m to the process p_v [Fig. 8]. Suppose the process p_t forwards the message m to p_v. The expected time DT_{sv} and expected number DN_{sv} of messages to deliver m to p_v are given as follows:

1. $DT_{sv} = (d_{st} + |m|/ b_{st}) + (d_{tv} + |m|/ b_{tv})/(1 - f_{tv})$.
2. $DN_{sv} = (2 - f_{tv}) /(1 - f_{tv})$.

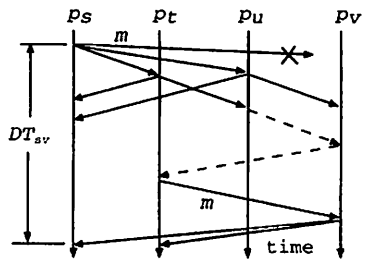

Fig. 8: Destination retransmission.

If $ST_{sv} > DT_{sv}$, the destination process p_t forwards the message m to the process p_v.

Each process p_t monitors delay time d_{tu}, bandwidth b_{tu}, and loss probability f_{tu} for each process p_u. For example, the process p_t obtains these QoS informations by periodically sending *ping* messages to all the processes in the group. The process p_t maintains the quality of service (QoS) information in a variable Q where $Q_{tu} = \langle b_{tu}, d_{tu}, f_{tu} \rangle$ for $u = 1, ..., n$. If the process p_t receives QoS information from another process p_s, $Q_{su} = \langle b_{su}, d_{su}, f_{su} \rangle$ for $u = 1, ..., n$. Each process p_t checks the retransmission condition by using the QoS information Q.

5.2.2 Change of retransmission scheme

In change of QoS, each process p_t in the group G changes the type of retransmission function. Suppose a sender process p_s sends a message m and all the processes take the sender retransmission scheme. As shown in Fig. 8, a process p_v fails to receive the message m. According to the change of QoS supported by the underlying network, the sender p_s makes a decision to change the retransmission scheme with the destination one, say a process p_t forwards the message m to p_v. However, the process p_t still takes the sender retransmission. Here, no process forwards the message m to p_v. In order to prevent this silent situation, we take a following protocol:

1. A process p_t sends a message m if p_t is a sender of m. If p_t is a destination process of m, p_t sends a receipt confirmation of m to the sender and all the other destination processes in the group G.
2. The process p_t detects that p_v has not received the message m.
3. The process p_t selects a retransmission scheme based on the QoS information Q.
4. Suppose the process p_t takes the destination retransmission scheme and p_t is a process to forward the message m to a process p_v.
   - p_t forwards m to p_v.
   - p_t sends a message Retxt to the sender p_s and a destination process, say p_v, which is next best process to forward m.
5. If the process p_i is not a process to forward the message m, p_t waits for receipt confirmation if m from p_v.
6. Suppose process p_t is a sender of a message m. If p_t takes a sender retransmission scheme, p_t retransmits m to p_v. If p_t takes a destination retransmission scheme, p_t waits for Retxt message from a destination. If p_t does not receive $ReTxT$, p_t retransmits m to p_v.

Next, suppose all the processes take the destination retransmission scheme in a group G. Here, a process p_s sends a message m and a process p_u fails to receive a message m. Suppose a process p_t is a process to forward m. Here, suppose QoS supported by the network is changed and p_t decides to take the sender retransmission scheme. However, the sender p_s still takes the destination retransmission scheme. Here, no process forwards the message m to the process p_u. In order to overcome the difficulty, p_t still forwards the message m to the process p_u.

1. Sender process p_s sends a message m to all the destination processes in the group G.
2. All destination processes in G send receipt confirmation not only to sender process p_s but also to other destination processes in group.
3. Process p_u fails to receive the message m.
4. A destination process p_t detects that p_u has not received the message m.
5. The process p_t forwards m to the process p_u at the same time, p_t sends $ReTxT$ message to the sender process p_s.
6. On receipt of the $ReTxT$ message, the sender p_s retransmits the message m to the process p_u.

[**Theorem**] At least one process forwards a message m to a process which fails to receive the message m. □

6 Concluding Remarks

In this paper, we made clear what types of functions to be realized in group communication protocol. We are now discussing how to select other functions so as to satisfy application requirements in change of network.

References

[1] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. on Computer Systems*, 5(1):47–76, 1987.

[2] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *CACM*, 21(7):558–565, 1978.

[3] F. Mattern. Virtual Time and Global States of Distributed Systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

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

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

[6] A. Nakamura and M. Takizawa. Causally Ordering Broadcast Protocol. *Proc. of IEEE ICDCS-14*, pages 48–55, 1994.

[7] R. Prasad. Basic Concept of Personal Area Networks. *WWRF, Kick off Meeting*, 2000.

[8] M. Rey. Transmission Control Protocol. *RFC793*, 1981.

[9] X. Zhao, C. Castelluccia, and M. Baker. Flexible Network Support for Mobile Hosts. MONET, 6(2), 2001.