マルチメディア通信と分散処理 49－7
（1991. 3. 7）

Selective Broadcast Service for Distributed Systems

Akihito NAKAMURA and Makoto TAKIZAWA

Dept. of Information and Systems Engineering
Tokyo Denki University
e-mail naka@takilab.k.dendai.ac.jp
e-mail taki@takilab.k.dendai.ac.jp

This paper discusses how to provide reliable broadcast communication for multiple entities in distributed systems by using unreliable broadcast communication services. In real distributed systems, each process in some application group rather sends every message to only the subset than all the processes in the group, and each process receives only messages destined to it from some process in the same order as they were sent. In this paper, we discuss how to design a protocol which provides such a selective broadcast service for the application processes in the group by using unreliable broadcast service in the presence of message loss.

分散型システムのための 選択的放送通信サービス

中村 章人　滝沢 誠
東京電機大学理工学部経営工学科

本論文では、低信頼放送通信サービスを用いて、分散型システム上の複数の応用プロセスに対し、高信頼な放送通信サービスを提供する問題について述べる。分散型システムにおいて、各応用グループ内のプロセスが送信する各メッセージは、送信順に受信される必要があるが、必ずしもそれが全プロセスに届けられることを必要としない。本論文では、分散型システム上の各応用プロセスに対して、メッセージの紛失が発生する低信頼な放送通信サービスを用いて、このような選択的放送通信サービスを提供するためのプロトコルの設計について述べる。

1. INTRODUCTION

Current communication networks provide reliable connection oriented protocols between two peer entities like *OSI* [OSI] protocols and *TCP/IP* [DOD]. The cooperation of a collection of more than two entities is required to realize distributed systems, e.g. distributed database systems. In these applications, processes in different sites send and receive messages by using the underlying communication system. In particular, processes need to send messages to all the processes in cooperation.

Local area networks (*LAN*s) and radio networks provide broadcast communication at the media access control (*MAC*) layer [IEEE]. However, they do not provide reliable broadcast communication among entities, e.g. some process in a station may fail to receive frames due to the lack of the buffer.

Reliable broadcast protocols have been studied in many literatures [CHAN84, SCHN84, TAKI87a,b89,90a,b, GARC88,89, NAKA88,89, KAAS89]. In these protocols, every protocol data unit (*PDU*), i.e. massage, is broadcast to all the entities in some group. In real distributed applications, although a collection of entities composes a group, each entity rather sends each *PDU* p to only a subset of the entities which are the destination of p than all the entities in the group. Also, each entity receives the *PDU*s destined to it from some entity in the same order as it sends. We name such a service a broadcast service for selectively partially ordering *PDU*s (*SPO* service). A simple mechanism for selective broadcast is studied in [WALL82]. It uses spanning trees for routing *PDU*s to their destinations and is based on one-to-one communication service. In this paper, we discuss how to design a protocol which provides the *SPO* service for the entities in the group by using unreliable broadcast communication where only lost *PDU* occurs as the failure and using distributed control.

In section 2, we give the definitions of correct receipt concepts among multiple entities. In section 3, we model unreliable and reliable broadcast communication services. In section 4, we present a data transmission procedure of the *SPO* protocol. Finally, we discuss the correctness and performance of the *SPO* protocol in section 5.

2. CORRECT RECEIPT CONCEPT AMONG MULTIPLE ENTITIES

A communication system M is composed of n (≥ 2) entities $\{E_1, \ldots, E_n\}$. Each entity E_k is a finite automaton, which is defined to be an initial state and a sequence of events and states ($k = 1, \ldots, n$). There are two kinds of events, i.e. receipt and sending events. Let $s_k[p]$ and $r_k[p]$ denote sending and receipt events of a *PDU* p in E_k, respectively. Here, let EE be a set of events in M. We define partial ordering relations $\rightarrow_k$ and $\rightarrow \subseteq EE^2$.

[Definition] For every pair of events e_1 and e_2 in E_k, $e_1 \rightarrow_k e_2$ *iff* e_1 occurs before e_2 (for $k = 1, \ldots, n$). $e_1 \rightarrow e_2$ *iff* (1) for some entity E_k, $e_1 \rightarrow_k e_2$, or (2) for some entities E_k and E_j (not necessarily different), there exists some *PDU* p such that $e_1 = s_k[p]$ and $e_2 = r_j[p]$.□

Let $\rightarrow_k^*$ and $\rightarrow^*$ be transitive closures of $\rightarrow_k$ and $\rightarrow$, respectively. $\rightarrow^*$ is a *happened-before* relation [LAMP78].

A. Accept

A *cluster* C is defined to be a set of n entities $E_1, \ldots, E_n$ [TAKI87a, b]. For every *PDU* p, let $p.DST$ be a set of the destination entities of p. $p.DST$ is a subset of C.

[Definition] A *PDU* p from E_j is said to be *accepted* in E_k *iff* for every *PDU* q from E_j, if $s_j[q] \rightarrow^* s_j[p]$, then $r_k[q] \rightarrow^* r_k[p]$.□

This means that E_k receives every *PDU* q sent by E_j before p.

B. Pre-Acknowledgment

We assume that every *PDU* from E_k carries the acknowledgments for *PDU*s which E_k has received already. This scheme is a most straightforward way to efficiently implement reliable communication protocols.

[Definition] A *PDU* p from E_j is said to be *partially pre-acknowledged* for E_h in E_k (written as $s_j[p] \Rightarrow_p^{h*} r_k[q]$) *iff* $s_j[p] \rightarrow r_h[p] \rightarrow^* r_k[q]$. A *PDU* p from E_j is said to be *pre-acknowledged* in E_k (written as $s_j[p] \Rightarrow_p^* r_k[q]$) *iff* for each $E_h \in p.DST$, $s_j[p] \rightarrow r_h[p] \rightarrow^* r_k[q]$ ($s_j[p] \Rightarrow_p^{h*} r_k[q]$).□

When p is pre-acknowledged in E_k, E_k knows that every entity in $p.DST$ has received p already.

C. Acknowledgment

Even if p is pre-acknowledged in E_k, E_k cannot consider that p is correctly received by all the entities in C. Because p might not be pre-acknowledged in some entity E_j, i.e. E_j considers that some entity E_h has not received p, e.g. failed to receive some reply from E_h.

[Definition] A *PDU* p from E_j is said to be *acknowledged* in E_k (written as $s_j[p] \Rightarrow_A^* r_k[q]$) *iff* for each $E_h \in p.DST$, $s_j[p] \Rightarrow_P^* r_h[g] \rightarrow^* r_k[q]$.□

When a *PDU* p is acknowledged in E_k, E_k considers that p is correctly received by all the destination entities. E_k knows that p is pre-acknowledged by every entity in $p.DST$.

3. SERVICE MODEL

We model the communication service for multiple entities. The entities in the cluster C send and receive *PDU*s by using the underlying communication service. The service which every entity uses is modeled as a set of logs [TAKI89, 90b].

3.1 Log

A *log* L is defined to be a sequence $(S, \rightarrow_L)$, i.e. a set S is totally ordered with respect to the ordering relation $\rightarrow_L \subseteq S^2$. Let $top(L)$ and $last(L)$ be elements a and b such that for every element c in L, $a \rightarrow_L c$ and $c \rightarrow_L b$, respectively. Elements in L are numbered from $top(L)$ to $last(L)$ as $1, 2, \ldots, m$, where m is the cardinality of S. Let $L[i]$ be the i-th element in L and i be the index of the element. L^i is inductively defined, i.e. $L^1 = L[1]$ and $L^i = L^{i-1} \mid L[i]$ $(i > 1)$ where $\mid$ is a concatenation of sequences. Also, $L[i] \rightarrow_L L[j]$ *iff* $i < j$. $L[i] \Rightarrow_L L[i+1]$ for $i = 1, \ldots, m-1$. We also write L consisting m elements as $< a_1 \ldots a_m]$ where $a_i = L[i]$ for $i = 1, \ldots, m$, and $a_1 = top(L)$ and $a_m = last(L)$.

For each entity E_k, there are two logs, i.e. a sending log SL_k and receipt log RL_k. SL_k is a log $(SP_k, \rightarrow_{SL_k})$ where SP_k is a set of *PDU*s which E_k has broadcast and $\rightarrow_{SL_k} \subseteq SP_k^2$ where for every p and q in SP_k, if $s_k[p] \rightarrow_k s_k[q]$, then $p \rightarrow_{SL_k} q$. That is, SL_k denotes a sequence of *PDU*s which E_k has broadcast.

The receipt log RL_k is a log $(RP_k, \rightarrow_{RL_k})$, i.e. a sequence of *PDU*s which E_k has received. For every p and q in RL_k, if $r_k[p] \rightarrow_k r_k[q]$, then $p \rightarrow_{RL_k} q$. Let RL_{kj} be a sublog $(RP_{kj}, \rightarrow_{RL_{kj}})$ of RL_k, where RP_{kj} is a set of *PDU*s which E_k has received from E_j and $\rightarrow_{RL_{kj}}$ is a restriction of $\rightarrow_{RL_k}$ to RP_{kj}.

3.2 Reliable Service

We define what is the reliable broadcast service for multiple entities.

[Definition] Two receipt logs RL_j and RL_k are said to be *order-equivalent iff* for every pair of *PDU*s p and q in both RL_j and RL_k such that both of E_j and E_k are in $p.DST \cap q.DST$, $p \rightarrow_{RL_j} q$ *iff* $p \rightarrow_{RL_k} q$. RL_j and RL_k are said to be *content-equivalent iff* $RP_j = RP_k$.□

In the order-equivalent case, two entities E_j and E_k receive *PDU*s in the same order. But they may fail to receive some *PDU*. In the content-equivalent case, they receive same *PDU*s, but the receipt sequences may be different.

[Definition] RL_k is said to be *order-preserved iff* for every entity E_j and for every *PDU* p and q in SL_j, if p and q in RL_k and $p \rightarrow_{SL_j} q$, then $p \rightarrow_{RL_k} q$. A receipt log RL_k is said to be *content-preserved iff* $RP_k = SP_1 \cup \ldots \cup SP_n$. RL_k is said to be *selectively content-preserved iff* $RP_k = SP_{1k} \cup \ldots \cup SP_{nk}$.□

If RL_k is order-preserved, E_k receives *PDU*s from each entity E_j in the same order as E_j sent. If RL_k is content-preserved, E_k receives all the *PDU*s which were sent by $E_1, \ldots, E_n$. If RL_k is selectively content-preserved, E_k receives all and only the *PDU*s destined to E_k.

[Definition] RL_k is said to be *correct iff* RL_k is order-preserved and content-preserved. RL_k is said to be *selectively correct iff* RL_k is order-preserved and selectively content-preserved. A communication service S is said to be *reliable iff* every receipt log in S is correct or selectively correct.□

[Definition] A communication service S is said to be a *multi-channel* (*MC*) service *iff* every receipt log in S is order-preserved.□

MC service is an abstraction of the service provided by systems where computers are connected by multiple channels, e.g. multiple *Ethernets*. Here, every entity can receive *PDU*s from each entity in the sending order but may fail to receive some of them. In this paper, we try to provide reliable broadcast service by using the *MC* service.

3.3 Selective Broadcast Communication (SBC) Service

Now, we define what is a *selective broadcast communication* (*SBC*) *service*. The *SBC* service is a kind of the reliable broadcast service, where each *PDU* is sent to only the destinations (not all the entities) in the cluster.

[Definition] A communication service S is said to be a *selective broadcast communication* (*SBC*) *service iff* every receipt log in S is selectively content-preserved.□

There are two kinds of *SBC* services according to the receipt ordering of *PDU*s.

[Definition] An *SBC* service S is said to be one for *selectively partially ordering PDU*s (*SPO*) *iff* every receipt log in S is selectively correct. S is said to be one for *selectively totally ordering PDU*s (*STO*) *iff* every receipt log in S is selectively correct and order-equivalent with each other.□

[Example 3.1] As an example of the *SPO* service, let us consider sending and receipt logs of three entities E_1, E_2, and E_3 as shown in Fig.1. Here, for each *PDU* p, $p_{j \ldots k}$ means that $p.DST = \{E_j, \ldots, E_k\}$. For example, a_{23} is a *PDU* whose destinations are E_2 and E_3. In the *SPO* service, every entity receives all and only the *PDU*s which are destined to it in the sending order. For example, E_1 sends *PDU*s c, d, and g to E_1, a, d, e, f, and g to E_2, and a, b, c, and g to E_3. Every entity receives all the *PDU*s from each entity which destined to it, i.e. $RP_1 = \{c, d, g, x, y, p, q\}$, $RP_2 = \{a, d, e, f, g, x, y, p\}$, and $RP_3 = \{a, b, c, g, y, z, p\}$. Also, each entity receives the *PDU*s in the sending order, e.g. $RL_{11} = <c\ d\ g\ ]$, $RL_{21} = <a\ d\ e\ f\ g\ ]$, and $RL_{31} = <a\ b\ c\ g\ ]$ in E_1, E_2, and E_3, respectively, for *PDU*s sent by E_1.□

E_1	$RL_1: <c\ x\ p\ y\ d\ g\ q\ ]$	$SL_1: <a_{23}\ b_3\ c_{13}\ d_{12}\ e_2\ f_2\ g_{123}\ ]$
E_2	$RL_2: <a\ d\ x\ e\ y\ p\ f\ g\ ]$	$SL_2: <x_{12}\ y_{123}\ z_3\ ]$
E_3	$RL_3: <a\ y\ b\ c\ p\ z\ g\ ]$	$SL_3: <p_{123}\ q_1\ ]$

Fig.1 An Example of the *SPO* Service

4. SPO PROTOCOL ON THE MC SERVICE

In this section, we discuss how to provide the *SPO* service (service for selectively partially ordering *PDUs*) by using the multi-channel (*MC*) service. Suppose that a cluster C includes n (≥ 2) entities $E_1, \ldots, E_n$.

4.1 Variables

A notation p^k is used to denote explicitly that a *PDU* p is sent by E_k. p^k has the following structure ($j = 1, \ldots, n$).

$p^k : <SRC;\ DST;\ TSEQ;\ <PSEQ_1 \ldots PSEQ_n>;\ <ACK_1 \ldots ACK_n>;\ BUF;\ DATA>$

$p^k.SRC = E_k$, i.e. an entity which sends p^k.
$p^k.DST$ = the set of destination entities of p^k.
$p^k.TSEQ$ = the total sequence number of p^k.
$p^k.PSEQ_j$ = the partial sequence number for E_j.
$p^k.ACK_j$ = the total sequence number of a *PDU* which expects to receive next from E_j.
$p^k.BUF$ = the number of buffers available in E_k.
$p^k.DATA$ = the data to be broadcast.

Every *PDU* p^k has *DST* field which informs receivers of whether they has to accept p^k or not. When E_j receives p^k, if $E_j \in p^k.DST$, E_j have to accept p^k. Otherwise, E_j can discard p^k. Each p^k has two kinds of sequence numbers, i.e. *total* and *partial* sequence numbers. Each p^k has a unique *total* sequence number $p^k.TSEQ$ which denotes the position in the total sequence of *PDU*s broadcast by E_k. Also, p^k has a unique *partial* sequence number $p^k.PSEQ_j$ for each entity E_j which denotes the position of the sequence of *PDU*s broadcast by E_k and destined to E_j ($j = 1, \ldots, n$). $p^k.ACK_j$ informs every entity in the cluster that E_k has received every *PDU* q^j from E_j where $q^j.TSEQ < p^k.ACK_j$. For the purpose of flow control, each *PDU* p^k carries the number $p^k.BUF$ of buffers in E_k.

Each E_k maintains the following variables ($h, j = 1, \ldots, n$).

$TSEQ$ = the total sequence number of a *PDU* which E_k expects to broadcast next.
$PSEQ_j$ = the partial sequence number of a *PDU* which E_k expects to send to E_j next.

$TREQ_j$ = the total sequence number of a *PDU* which E_k expects to receive next from E_j.
$PREQ_j$ = the partial sequence number of a *PDU* which E_k expects to receive next from E_j.
AL_{hj} = the total sequence number of a *PDU* which E_k knows E_j expects to receive next from E_h.
PAL_{hj} = the total sequence number of a *PDU* which E_k knows that E_j expects to pre-acknowledge from E_h.
F_j = the number of buffers in E_j which E_k knows of.

Let $minAL_j$ denote the minimum among $AL_{j1}, \ldots, AL_{jn}$. This means that all the entities have already received every *PDU* g^j where $g^j.TSEQ < minAL_j$. Let ISS_j be an initial total sequence number of E_j. Initially, $TSEQ = PSEQ_j = ISS_j$ and $TREQ_j = PREQ_j = AL_{jh} = ISS_j$ $(h, j = 1, \ldots, n)$ in E_k. We suppose that every entity in the cluster knows ISS_j and initial buffer size IBF_j for every E_j when the cluster is established by the cluster establishment procedure [TAKI87a, b]. Each E_k has n variables $F_1, \ldots, F_n$, where F_j denotes the number of buffers in E_j which E_k knows of, and initially $F_j = IBF_j$ $(j = 1, \ldots, n)$. Let *minF* denote the minimum among $F_1, \ldots, F_n$.

4.2 Accept and Transmission

Here, each entity E_k has n receipt sublogs $RL_{k1}, \ldots, RL_{kn}$, where each RL_{kj} keeps track of *PDUs* from E_j $(j = 1, \ldots, n)$.

A. Accept

When E_k receives p^j (from E_j), if p^j satisfies the following accept condition, it is accepted by the accept action.

[Accept Condition for p^j] (1) (1-1) $p^j.TSEQ = TREQ_j$ or (1-2) $p^j.PSEQ_k = PREQ_j$, and (2) $p^j.ACK_h \le TREQ_h$ $(h = 1, \ldots, n)$.□

[Accept Action for p^j](1) $TREQ_j := p^j.TSEQ$, (2) $AL_{hj} := p^j.ACK_h$ $(h = 1, \ldots, n)$, and (3) If $E_k \in p^j.DST$, then $PREQ_j := p^j.PSEQ_k + 1$, and E_k enqueues p^j into RL_{kj} and marks it "*accepted*". Otherwise, E_k discards p^j.□

If E_k does not fail to receive *PDUs*, the condition (1-1) is always satisfied. Even if E_k fails to receive a *PDU*, say p^j, if $p^j.DST$ does not include E_k, the loss of p^j does not matter E_k. The condition (1-2) is one to check it.

Let RPL_{kj} be a sublog of RL_{kj} which is composed of accepted *PDUs*. RPL_{kj} is a postfix of RL_{kj}.

B. Transmission

If the flow condition holds, E_k broadcasts a *PDU* p^k. Here, W and H are constants. W gives the window size.

[Flow Condition] $minAL_k \le TSEQ < minAL_k + min(W, minF/(H*n^2))$.□

[Transmission Action of p^k] (1) $p^k.TSEQ := TSEQ$, $TSEQ := TSEQ + 1$. (2) $p^k.PSEQ_j := PSEQ_j$ $(j = 1, \ldots, n)$, and for each E_j, if E_j is a destination of p^k, then $PSEQ_j := PSEQ_j + 1$, and $p^k.DST := p^k.DST \cup \{E_j\}$. (3) $p^k.ACK_h := TREQ_h$ $(h = 1, \ldots, n)$. (4) E_k enqueues p^k into SL_k and broadcasts p^k.□

As long as p^k is stored in SL_k, E_k can rebroadcast p^k if necessary.

4.3 Pre-Acknowledgment

The problem is how each entity E_k decides the correct receipt of p^j based on received *PDUs* in the distributed control scheme. Here, the following notations are introduced.

$AL_j(p^j) = \{ AL_{jh} \mid E_h \in p^j.DST \}$.
$minAL_j(p^j)$ = the minimum number in $AL_j(p^j)$.

$minAL_j(p^j)$ means that every entity in $p^j.DST$ has received a *PDU* whose *TSEQ* is less than it. Hence, if the following condition holds for p^j which has been accepted already, E_k can know that every destination entity of p^j has accepted p^j. That is, p^j is pre-acknowledged in E_k. If p^j satisfies the *PACK* condition, E_k performs the following *PACK* action.

[Pre-acknowledgment (*PACK*) Condition for p^j] $p^j.TSEQ < minAL_j(p^j)$. □

[Pre-acknowledgment (*PACK*) Action] For every $j = 1, \ldots, n$, while $p^j = top(RPL_{kj})$ satisfies the *PACK* condition, { p^j is marked "*pre-acknowledged*". $PAL_{hj} := p^j.ACK_h$ $(h = 1, \ldots, n)$ }.□

[Lemma 4.1] If p^j received by E_k satisfies the *PACK* condition, p^j is pre-acknowledged in E_k.
[Proof] The *PACK* condition means that for every $E_h \in p^j.DST$, $p^j.TSEQ < AL_{jh}$. That is, for every $E_h \in p^j.DST$, there exists a *PDU* q^h such that $s_j[p^j] \Rightarrow_{p^h} r_k[q^h]$. Hence, p^j is pre-acknowledged.□

Let PPL_{kj} be a subsequence of RL_{kj} which is composed of pre-acknowledged *PDUs*. PPL_{kj} is an infix of RL_{kj}.

4.4 Acknowledgment

Next, we consider how to acknowledge *PDU*s. Here, the following notations are introduced.

$PAL_j(p^j) = \{ PAL_{jh} \mid E_h \in p^j.DST \}$.
$minPAL_j(p^j)$ = the minimum number in $PAL_j(p^j)$.

[Acknowledgment (*ACK*) Condition for p^j] $p^j.TSEQ < minPAL_j(p^j)$.□

[Acknowledgment (*ACK*) Action] For every $j = 1, \ldots, n$, while $p^j = top(PPR_{kj})$ and p^j satisfies the *ACK* condition, { p^j is marked "*acknowledged*" }.□

[Lemma 4.2] If p^j satisfies the *ACK* condition, p^j is acknowledged in E_k.
[Proof] The receipt of q^h which partially pre-acknowledges p^j for E_h means that E_h has received p^j. The pre-acknowledgment of q^h means that every entity in $p^j.DST$ knows that E_h received p^j. Hence, if every *PDU* which pre-acknowledges p^j is received, E_k knows that every entity in $p^j.DST$ has known that every $E_h \in p^j.DST$ had received p^j. Each PAL_{jh} means that a *PDU* which partially pre-acknowledges p^j for E_h is pre-acknowledged in E_k. Hence, $minPAL_j(p^j)$ means that a *PDU* which partially pre-acknowledges p^j for every E_h in $p^j.DST$ is pre-acknowledged in E_k. Therefore, p^j is acknowledged in E_k.□

That is, every *PDU* which satisfies the *ACK* condition in PPL_{kj} is acknowledged. Let APL_{kj} be a prefix of PPL_{kj} which is composed of acknowledged *PDU*s.

4.5 Failures

When the *MC* service is used, *PDU*s may be lost. Lost *PDU*s can be detected by checking the following *FP* condition each time when E_k receives some *PDU*.

[Failure Point (*FP*) Condition] [Fig.2] (1) On receipt of p^j, if $PREQ_j < p^j.PSEQ_k$, then E_k has not received g^j such that $PREQ_j \le g^j.PSEQ_k < p^j.PSEQ_k$ ($j = 1, \ldots, n$). (2) On receipt of q^h, for some j ($\neq h$), if $TREQ_j < q^h.ACK_j$, then E_k has not received g^j such that $TREQ_j \le g^j.TSEQ < q^h.ACK_j$ ($h = 1, \ldots, n$).□

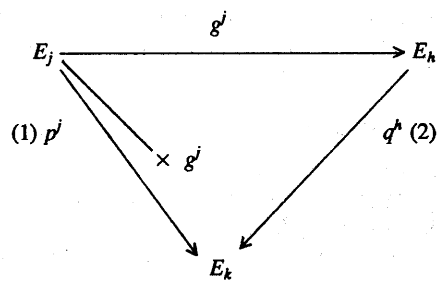

Fig.2 Detection of Lost *PDU*s

On receipt of *PDU*, if a lost *g* is found by the *FP* condition, the following lost *PDU* action is performed.

[Lost *PDU* Action] (1) If the *FP* condition (1) holds, E_k invokes the *RETRANS* procedure to require the entities which broadcast *g* to rebroadcast *PDU*s, which is presented later.
(2) If the *FP* condition (2) holds, E_k starts a timer for E_j. If q^h satisfies the accept condition, q^h is accepted.
(2-1) If the timer expires, E_k invokes the *RETRANS* procedure.
(2-2) If E_k receives r^j from E_j, the timer is stopped and E_k checks the accept condition for r^j. If satisfied, E_k accepts r^j. If not, E_k invokes the *RETRANS* procedure.□

When the *FP* condition (1) holds, E_k has failed to receive some *PDU* and has to receive every *PDU* g^j such that $PREQ_j \le g^j.PSEQ_k < p^j.PSEQ_k$. On the other hand, if (2) holds, E_k detects some lost *PDU* g^j but does not know whether g^j is destined to E_k, i.e. $E_k \in g^j.DST$, or not. E_k has to receive only *PDU*s g^j such that $E_k \in g^j.DST$. If E_k invokes the *RETRANS* procedure as soon as the *FP* condition (2) holds, it may be meaningless for E_j to rebroadcast g^j, because $g^j.DST$ may not include E_k. Hence, E_k waits on some *PDU* from E_j for a while. Suppose that E_k receives a *PDU* r^j. If $r^j.PSEQ_k = PREQ_j$, E_k does not need to receive g^j. If $PREQ_j < r^j.PSEQ_k$, E_k should have received g^j.

[Retransmission (*RETRANS*) Procedure] (1) E_k broadcasts a *RETRANS PDU rt* such that $rt.ACK_h = TREQ_h$ ($h = 1, \ldots, n$). (2) If E_j receives the *RETRANS rt* from E_k, E_j rebroadcasts *PDU* g^j such that $g^j.TSEQ \geq rt.ACK_j$ and $E_k \in g^j.DST$.□

Since entities rebroadcast *PDU*s, some duplicate processing is required.

[Duplicate *PDU* Condition for p^j] $p^j.TSEQ < TREQ_j$, or $p^j.PSEQ_k < PREQ_j$.□

[Duplicate *PDU* Action for p^j] p^j is neglected.□

5. EVALUATION

5.1 Correctness

Here, we prove that the *SPO* protocol provides the *SPO* service.

[Theorem 5.1] The *SPO* protocol provides an *SPO* service for the cluster on the *multi-channel* (*MC*) service.
[Proof] If there is no failure, it is clear from Lemma 4.1 and 4.2. Suppose that E_k fails to receive some *PDU* p^j. It is found by the *FP* condition. (1) If p^j is not received by E_k, it is not pre-acknowledged in every entity E_h in $p^j.DST$ since E_k never informs E_h of the receipt of p^j. (2) In a case that E_k fails to receive q^i which pre-acknowledges p^j, it is detected by the *FP* condition. If p^j is not pre-acknowledged in E_k, p^j is not acknowledged in any entity in $p^j.DST$.□

By this theorem, the *SPO* service can be provided for the upper layer by the *SPO* protocol on the underlying communication system like a system which is composed of the multiple *Ethernets* or multiple radio channels.

5.2 Performance

Let n be the number of entities in the cluster and m be the average number of destinations of *PDU*s broadcast in the cluster ($m \leq n$). For every entity E_i, let d_i be a mean time between transmission of *PDU*s. That is, E_i broadcasts *PDU*s every d_i time units on the average. Let t_i be a mean time between arrival of *PDU*s. Since the *Ethernet* is used as the underlying service, t_i is a constant t. t is $1/(1/d_1 + \ldots + 1/d_n)$. If every d_i is a constant d, t is d/n where n is a number of entities in C. Suppose that every d_i is the same d. Let r be an average propagation delay time from one entity to the other entity.

First, we assume that the underlying communication service has infinite capacity, i.e. every entity can broadcast *PDU*s any time without waiting. It takes a received *PDU* $(r + d/2)$ time units on the average to be pre-acknowledged. During the time, an entity receives $(r + d/2)/t = (r + d/2)n/d$ *PDU*s. It gives a number of *PDU*s in the queues, i.e. *RPL* and *PPL*. If r is independent of n, the queue length is $O(n)$. In the other case, the underlying service has a limited capacity. Especially, if the network is heavily loaded, the delay time r is proportional to n. In this case, the queue length is $O(n^2)$.

As compared with the *TO* and *PO* protocols [TAKI87a,b, NAKA89, TAKI89,90a,b], the performance advantage of *SPO* protocol appears when a failure occurs. We measure the number of *PDU*s discarded and retransmitted when a *PDU* is lost. Let N_{TO}, N_{PO}, and N_{SPO} be the number of *PDU*s discarded in the *TO*, *PO*, and *SPO*, respectively, when the lost *PDU* p is detected. In the *SPO*, if p does not include E_k in its destination, E_k does not need to receive it, i.e. $N_{SPO} = 0$. In the *TO* and *PO*, the destination entities of each *PDU* are all the entities in the cluster. In the *TO*, *PDU*s which come from more than one entity are discarded since every *PDU* received by E_k is stored in a single receipt log. In the *PO*, discarded *PDU*s due to a lost *PDU* are always from an entity since *PDU*s broadcast by different entities are stored in different receipt sublogs. In the *PO*, the number of *PDU*s discarded and retransmitted is $N_{PO} = N_{TO}/n$. In both the *TO* and *PO*, lost *PDU*s are eventually rebroadcast, but it depends on the destination in *SPO*. Even if the lost *PDU* includes E_k in its destination, $N_{SPO} = N_{TO}/n * m/n$. That is, $N_{SPO} \leq N_{PO} \leq N_{TO}$.

6. CONCLUDING REMARKS

In this paper, we have discussed a design of data transmission procedure which provides one class of reliable broadcast communication service, i.e. a broadcast (*SPO*) service for selectively partially ordering *PDU*s, by using unreliable broadcast *MC* services. In the *SPO* service, each *PDU* is destined to not all the entities, but only the destinations. The protocol is based on distributed control and the cluster concept. A cluster is a set of multiple entities. The *SPO* protocol provides the partial ordering of received *PDU*s which are destined to the entity on the *MC* service. Also, we have shown the correctness and the performance of the *SPO* protocol on the *MC* service.

REFERENCES

[CHAN84] Chang, J. M. and Maxemchuk, N. F., "Reliable Broadcast Protocols," *ACM Trans. on Computer Systems*, Vol.2, No.3, pp.251-273, 1984.

[DOD] Defense Communications Agency, "DDN Protocol Handbook," Vol.1 - 3, NIC 50004-50005, 1985.

[GARC88] Garcia-Molina, H. and Kogan, B., "An Implementation of Reliable Broadcast Using an Unreliable Multicast Facility," *Proc. of the 7th IEEE Symposium on Reliable Distributed Systems*, pp.428-437, 1988.

[GARC89] Garcia-Molina, H. and Spauster, A., "Message Ordering in a Multicast Environment," *Proc. of the 9th IEEE Inter. Conf. on Distributed Computing Systems*, pp.354-361, 1989.

[IEEE] "IEEE Project 802 Local Network Standards-Draft," 1982.

[KAAS89] Kaashoek, M. F., Tanenbaum, A. S., Hummel, S. F., and Bal, H. E., "An Efficient Reliable Broadcast Protocol," *ACM Operating System Review*, Vol.23, No.4, pp.5-19, 1989.

[LAMP78] Lamport, R., "Time, Clocks, and the Ordering of Events in Distributed Systems," *Communications of the ACM*, Vol.21, No.7, pp.558-565, 1978.

[METC76] Metcalfe, R. M., "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, Vol.19, No.7, pp.395-404, 1976.

[NAKA88] Nakamura, A. and Takizawa, M.,"Totally Ordering Broadcast Protocol on Multi-Channel System," *IPSJ DPS*, 39-1, pp.1-8, 1988.

[NAKA89] Nakamura, A. and Takizawa, M., "Totally Ordering (TO) and Partially Ordering (PO) Broadcast Protocol," *Proc. of the 4th Joint Workshop on Computer Communication (JWCC)*, pp.35-43, 1989.

[OSI] "Data Processing - Open Systems Interconnection - Basic Reference Model," ISO 7498, 1987.

[SCHN84] Schneider, F. B. and et al., "Fault-Tolerant Broadcasts," *Science of Computer Programming*, Vol.4, pp.1-15, 1984.

[TAKI87a] Takizawa, M., "Cluster Control Protocol for Highly Reliable Broadcast Communication," *Proc. of the IFIP Conf. on Distributed Processing*, pp.431-445, 1987.

[TAKI87b] Takizawa, M., "Design of Highly Reliable Broadcast Communication Protocol," *Proc. of IEEE COMPSAC87*, pp.731-740, 1987.

[TAKI89] Takizawa, M. and Nakamura, A., "Totally Ordering Broadcast (TO) Protocol on the Ethernet," *Proc. of the IEEE Pacific RIM Conf. on Communications, Computers and Signal Processing*, pp.16-21, 1989.

[TAKI90a] Takizawa, M. and Nakamura, A., "Partially Ordering Broadcast (PO) Protocol," *Proc. of the 9th IEEE Conf. on Computer Communications (Infocom)*, pp.357-364, 1990.

[TAKI90b] Takizawa, M. and Nakamura, A., "Reliable Broadcast Communication," *Proc. of IPSJ Inter. Conf. on Information Technology (InfoJapan)*, pp.325-332, 1990.

[WALL82] Wall, D. W., "Selective Broadcast in Packet-Switched Networks," *Proc. of the 6th Berkely Workshop on Distributed Data Management and Computer Networks*, pp.239-258, 1982.