マルチメディア通信と分散処理 58−14
（1992. 11. 20）

優先度付き放送通信プロトコルにおける連同期方法

中村 章人　滝沢 誠

東京電機大学理工学部経営工学科

グループウェアシステム等の分散型応用システムでは、複数の宛先に、テキスト、音声、動画等の複数種類のデータを転送するための、高信頼放送通信サービスが必要となる。このための方法として、各プロトコルデータ単位（PDU）に優先度を与え、優先度の高いものから順に宛先に届けることがある。本論文では、分散型応用システムを構成する全応用エンティティが、PDUを紛失なく、優先度に基づく同一の順序で受信する放送通信サービスを定義し、このためのプロトコルを提案する。優先度に基づいてPDUを配送するサービスでは、低優先度のPDUが最悪の場合、無限に待ち続けてしまう問題がある。本論文では、この問題を解決するために、連の概念を導入する。連は、優先度順に整列されたPDUの系列であり、応用エンティティは、連の系列としてPDUを優先度順に受信する。

Run Synchronization in the Priority-Based Broadcast Protocol

Akihito Nakamura　Makoto Takizawa

Department of Computers and Systems Engineering
Tokyo Denki University
Ishizaka, Hatoyama, Hiki-gun, Saitama 350-03
E-mail {naka, taki}@takilab.k.dendai.ac.jp

In distributed applications like groupware systems, it is required to provide reliable broadcast service, by which application entities send various kinds of data, e.g. text, voice, and video, to multiple destinations reliably and efficiently. In such kinds of applications, some protocol data units (PDUs) have to be delivered to the destinations earlier than another PDUs. One approach to providing such communication service is to give a priority to each PDU and to deliver the PDUs to the destinations in the priority-based order. In this paper, we discuss distributed broadcast protocols which provide priority-based receipt ordering of PDUs for entities by using a single channel system like Ethernet. In the priority-based ordering service, there is a *starvation* problem, i.e. lower-priority PDUs can be left waiting indefinitely in the receipt queue since higher-priority PDUs jump over the lower-priority PDUs. In this paper, we present a method by which even lower-priority PDUs are delivered to the application in some pre-defined time by partitioning the receipt sequence of PDUs into *runs*, where each run is priority-based ordered.

1 Introduction

In distributed applications like groupware systems [5], group communication among multiple entities is required in addition to one-to-one communication provided by OSI [9] and TCP/IP [4]. Reliable broadcast communication systems have been discussed in [1, 2, 6, 7, 10, 11, 15, 16, 18, 19, 21, 23, 24, 25, 26]. In these papers, it has been discussed how to provide atomicity of delivering PDUs and receipt ordering of them.

In the distributed applications, various kinds of data are broadcast to multiple sites. For example, control messages have to be delivered to the destinations earlier than another data. One approach to delivering more time-critical PDUs to the destinations earlier than less time-critical ones is to give priority to each PDU. There are two kinds of priority-based transmission schemes i.e. *controlled access* [12, 22] and *contention-based* [12, 14] protocols. In this paper, we discuss contention-based priority concepts among multiple entities in a group named a *cluster*. In the broadcast communication, it is important to consider in which order each entity in the cluster receives PDUs with priorities. In [20], two priority-based broadcast services are defined, i.e. *priority-based total ordering* (PriTO) and *priority-based semi-total ordering* services. In the PriTO, every entity receives all the PDUs not only in the same order but also the priority-based order. In the other service, PDUs received are ordered according to the priorities, where PDUs with the same priority may be ordered differently by different entities. One problem is *starvation*, i.e. lower-priority PDUs can be left waiting indefinitely in the receipt queue since higher-priority PDUs jump over lower-priority ones. In this paper, we present a PriTO protocol by which even lower-priority PDUs are delivered to the application entities in some pre-defined time by partitioning the receipt sequence of PDUs into *runs*, each of which is priority-based ordered. Each entity receives the same sequence of the runs.

In section 2, we define the basic concepts. In section 3, we discuss the priority-based ordering of PDUs. In section 4, we present a protocol which provides a PriTO service by using single channel network like the Ethernet. In section 5, we discuss how to resolve the starvation problem.

2 Basic Concepts

2.1 Cluster

A communication system is modeled to be composed of three layers, i.e. application, system, and network layers. Entities in the system layer provide some service for entities in the application layer by adding some value to the service provided by the network layer. A *cluster* C [23, 24] is defined to be a set of *service access points* (SAPs) $S_1, ..., S_n$ $(n \geq 2)$. Each S_i is supported by a system entity E_i, and each application entity A_i takes communication service through S_i to which A_i is attached $(i = 1, ..., n)$. Here, C is said to be *supported* by E_1,...,E_n, and be *composed* of A_1,...,A_n. The cluster is an extension of the conventional connection concept on two SAPs to n SAPs. In this paper, we assume that a cluster is established by multiple entities by using a protocol presented in [23, 24].

2.2 Correct receipt among multiple entities

There are three levels of correct receipt among multiple entities, i.e. *accepted*, *pre-acknowledged*, and *acknowledged* [23, 24]. Here, suppose that a cluster C is supported by n system entities E_1,...,E_n.

(1) When a PDU p arrives at E_i, p is *accepted* by E_i.

(2) When E_i knows that every entity in C has accepted p, p is *pre-acknowledged* by E_i.

(3) When E_i knows that every entity in C has pre-acknowledged p, p is *acknowledged* by E_i.

At (2), although E_i knows that every entity in C has accepted p, some E_j still may not know that another entity has accepted p. For example, E_j has not received the acknowledgment to p from some E_h yet. (3) represents the highest correct level.

Next, we would like to consider logical properties of cluster services. The service is modeled as a set of logs. A log L is a sequence of PDUs $< p_1 \ ... \ p_m \ ]$, where p_1 is the top and p_m is the last. Here, $top(L)$ and $last(L)$ denote the top and the last PDUs in L, respectively. In L, p_i precedes p_j (written as $p_i \rightarrow_L p_j$) if $i < j$. Here, let L_1 be a log $< q_1 \ ... \ q_n \ ]$. $L \ || \ L_1$ denotes a concatenation of L and L_1, i.e. $< p_1 \ ... \ p_m \ q_1 \ ... \ q_n \ ]$. Here, let $L|_i^j$ $(i$

$\leq j$) denote a subsequence $< p_i\ p_{i+1}\ \ldots\ p_{j-1}\ p_j\ ]$ of L. Each entity E_i has a sending log SL_i and a receipt log RL_i. SL_i and RL_i are sequences of PDUs sent and received by E_i, respectively. There are the following relations among the receipt and sending logs.

- RL_i is *order-preserved* iff for every entity E_j, $p \rightarrow_{RL_i} q$, if $p \rightarrow_{SL_j} q$.
- RL_i is *information-preserved* iff RL_i includes all the PDUs in SL_1,...,SL_n.
- RL_i and RL_j are *information-equivalent* iff RL_i and RL_j include the same PDUs.
- RL_i and RL_j are *order-equivalent* iff for every pair of PDUs p and q included in both RL_i and RL_j, $p \rightarrow_{RL_i} q$ iff $p \rightarrow_{RL_j} q$.

RL_i is *preserved* iff RL_i is order- and information-preserved. RL_i and RL_j are *equivalent* iff RL_i and RL_j are both order- and information-equivalent.

[**Definition**] A *one-channel* (1C) service is one where every receipt log is order-preserved and order-equivalent. □

The 1C service is abstraction of services provided by the Ethernet MAC [8] and radio networks. Although every entity receives PDUs in the same order preserving the sending order, each entity may fail to receive PDUs. In this paper, the 1C service is used as the underlying network layer.

[**Definition**] *Order-preserved* (OP) service is one where every receipt log is preserved. *Total ordering* (TO) service is an OP service where every receipt log is order-equivalent with each other. □

The protocols which provide the TO and OP services are presented in [18, 19, 23, 24, 25, 26].

3 Priority-Based Cluster Service

Each entity gives each PDU p a unique sequence number $p.SEQ$ and a priority $p.PRI$ (> 0). If p is broadcast after a PDU q, $p.SEQ > q.SEQ$. If p has higher priority than q, $p.PRI > q.PRI$. Let $p_{[r]}$ denote that p has priority r, i.e. $p.PRI = r$. The priority 0 is only used in the system. Here, a notation p^i is used to explicitly denote that p is broadcast by E_i. $p.SRC$ denotes the source entity of p.

[**Definition**] A log L is said to be *priority-based ordered* iff for every two PDUs p and q in L, (1) if $p.PRI > q.PRI$, then $p \rightarrow_L q$, and (2) if $p.PRI = q.PRI$, $p.SRC = q.SRC$, and $p.SEQ < q.SEQ$, then $p \rightarrow_L q$. □

[**Definition**] Two logs L_i and L_j are *priority-equivalent* iff L_i and L_j are both information-equivalent and priority-based ordered. □

[**Example**] Suppose that E_1 broadcasts PDUs a, b, and c, E_2 broadcasts p and q, and E_3 broadcasts x, y, and z. The sending and receipt logs of each entity are shown in Figure 1. RL_1, RL_2, and RL_3 are priority-based equivalent because they include the same PDUs, and are priority-based ordered. It is noted that x and y are received in the sending order because they are broadcast by E_3 and have the same priority 1. □

$$
\begin{aligned}
SL_1 &= < a_{[1]}\ b_{[2]}\ c_{[3]}\ ] \\
SL_2 &= < p_{[2]}\ q_{[1]}\ ] \\
SL_3 &= < x_{[1]}\ y_{[2]}\ z_{[1]}\ ] \\
RL_1 &= < c_{[3]}\ b_{[2]}\ y_{[2]}\ p_{[2]}\ a_{[1]}\ x_{[1]}\ q_{[1]}\ z_{[1]}\ ] \\
RL_2 &= < c_{[3]}\ y_{[2]}\ b_{[2]}\ p_{[2]}\ x_{[1]}\ q_{[1]}\ z_{[1]}\ a_{[1]}\ ] \\
RL_3 &= < c_{[3]}\ p_{[2]}\ b_{[2]}\ y_{[2]}\ q_{[1]}\ x_{[1]}\ z_{[1]}\ a_{[1]}\ ]
\end{aligned}
$$

Figure 1: Priority-based equivalent

[**Definition**] Let R be a subsequence of a log L. R is said to be a *run* in L (written as $R \sqsubseteq L$) if R is priority-based ordered. □

[**Definition**] Let R_1 and R_2 be subsequences $L|_{i_1}^{j_1}$ and $L|_{i_2}^{j_2}$ of L, respectively. If $i_2 = j_1 + 1$, R_2 is said to be *connected* to R_1. □

[**Example**] Let us consider a log L as shown in Figure 2. R_1, R_2, and R_3 are runs of L. R_4 is not a run of L because $e.PRI(=1) < f.PRI(=2)$, i.e. it is not priority-based ordered. R_3 is connected to R_2. □

$$
\begin{aligned}
L &= < a_{[3]}\ b_{[2]}\ c_{[2]}\ d_{[1]}\ e_{[1]}\ f_{[2]}\ g_{[1]}\ h_{[1]}\ ] \\
R_1 &= L|_1^3 = < a_{[3]}\ b_{[2]}\ c_{[2]}\ ] && \sqsubseteq L \\
R_2 &= L|_2^5 = < b_{[2]}\ c_{[2]}\ d_{[1]}\ e_{[1]}\ ] && \sqsubseteq L \\
R_3 &= L|_6^8 = < f_{[2]}\ g_{[1]}\ h_{[1]}\ ] && \sqsubseteq L \\
R_4 &= L|_5^8 = < e_{[1]}\ f_{[2]}\ g_{[1]}\ h_{[1]}\ ] && \not\sqsubseteq L
\end{aligned}
$$

Figure 2: Runs

[**Definition**] A log L is said to be *run-partitioned* to R_1,...,R_k if $L = (R_1 \parallel \ldots \parallel R_k)$. □

[**Definition**] A run-partition $(R_1 \parallel \ldots \parallel R_k)$ of L is said to be *maximum* in L if (1) $k = 1$, or (2)

$k > 1$, for every pair of connected runs R_i and R_{i+1}, $p.PRI > q.PRI$ where p is the last of R_i and q is the top of R_{i+1} ($i = 1, ..., k-1$). □

It is noted that there exists only one maximum run-partition for every log L although there may be more than one run-partition of L.

[Example] Figure 3 shows run-partitions of logs L_1 and L_2. The run-partition (S_{11} || S_{12} || S_{13}) is not maximum in L_1. However, (R_{11} || R_{12}) is maximum. (R_{21} || R_{22}) is maximum but (S_{21} || S_{22} || S_{23}) is not in L_2. □

$$L_1 = < \overbrace{\underbrace{a_{[3]}\ b_{[2]}}_{S_{11}}\ \underbrace{c_{[2]}\ d_{[1]}\ e_{[1]}}_{S_{12}}}^{R_{11}}\ \overbrace{\underbrace{f_{[2]}\ g_{[1]}\ h_{[1]}}_{S_{13}}}^{R_{12}}]$$

$$L_2 = < \overbrace{\underbrace{a_{[3]}\ c_{[2]}}_{S_{21}}\ \underbrace{b_{[2]}\ e_{[1]}\ d_{[1]}}_{S_{22}}}^{R_{21}}\ \overbrace{\underbrace{f_{[2]}\ h_{[1]}\ g_{[1]}}_{S_{23}}}^{R_{22}}]$$

Figure 3: Run-partition

[Definition] Let (R_{11} || ... || R_{1h}) and (R_{21} || ... || R_{2k}) be maximum run-partitions of logs L_1 and L_2, respectively. L_1 and L_2 are said to be *run-equivalent* iff (1) $h = k$ ($= m$), and (2) R_{1i} and R_{2i} are priority-based equivalent for $i = 1, ..., m$. □

[Example] Let us consider two logs L_1 and L_2 as shown in Figure 3. The run-partitions (R_{11} || R_{12}) of L_1 and (R_{21} || R_{22}) of L_2 are run-equivalent because R_{11} and R_{21}, and R_{12} and R_{22} are priority-based equivalent, respectively. □

There are two kinds of broadcast communication service on the priority.

[Definition] The cluster service of a cluster C is said to be a *priority-based ordering* (PriO) service iff every receipt log in C is information-preserved and is run-equivalent with each other. The PriO service of C is said to be a *priority-based total ordering* (PriTO) service iff every receipt log in C is order-equivalent with each other. □

[Example] Examples of PriO and PriTO services are shown in (1) and (2) of Figure 4, respectively. Here, a cluster C is supported by three entities E_1, E_2, and E_3. In (1) and (2), every entity receives all the PDUs broadcast in C, i.e. a, b, c, d, e, and f. Hence, RL_1, RL_2, and RL_3 are information-preserved. A run-partition (R_{1i} || R_{2i}) of RL_i is maximum, where R_{1i} includes b, c, d, and e, and R_{2i} includes a and f ($i = 1, 2, 3$). R_{11}, R_{12}, and R_{13} are priority-equivalent, and so are R_{21}, R_{22}, and R_{23}. Hence, every receipt log is run-equivalent in (1) and (2). In the PriO service (1), PDUs which have the same priority may be received in any order, e.g. c and e, and a and f are received in different orders. On the other hand, all the entities receive the same PDUs in the same order in the PriTo service (2). □

$$SL_1 = < a_{[2]}\ b_{[3]}]$$
$$SL_2 = < c_{[2]}\ d_{[1]}]$$
$$SL_3 = < e_{[2]}\ f_{[2]}]$$

$$RL_1 = < \overbrace{b_{[3]}\ c_{[2]}\ e_{[2]}\ d_{[1]}}^{R_{11}}\ \overbrace{a_{[2]}\ f_{[2]}}^{R_{12}}]$$
$$RL_2 = < \overbrace{b_{[3]}\ e_{[2]}\ c_{[2]}\ d_{[1]}}^{R_{21}}\ \overbrace{a_{[2]}\ f_{[2]}}^{R_{22}}]$$
$$RL_3 = < \overbrace{b_{[3]}\ e_{[2]}\ c_{[2]}\ d_{[1]}}^{R_{31}}\ \overbrace{f_{[2]}\ a_{[2]}}^{R_{32}}]$$

(1) PriO service

$$RL_1 = < \overbrace{b_{[3]}\ c_{[2]}\ e_{[2]}\ d_{[1]}}^{R_{11}}\ \overbrace{a_{[2]}\ f_{[2]}}^{R_{12}}]$$
$$RL_2 = < \overbrace{b_{[3]}\ c_{[2]}\ e_{[2]}\ d_{[1]}}^{R_{21}}\ \overbrace{a_{[2]}\ f_{[2]}}^{R_{22}}]$$
$$RL_3 = < \overbrace{b_{[3]}\ c_{[2]}\ e_{[2]}\ d_{[1]}}^{R_{31}}\ \overbrace{a_{[2]}\ f_{[2]}}^{R_{32}}]$$

(2) PriTO service

Figure 4: PriO and PriTO

4 Priority-Based Total Ordering (PriTO) Protocol

We present a protocol which provides the PriTO service for the application entities by using the 1C service.

4.1 Transmission and receipt

Each entity E_i has the following variables ($j, k = 1, ..., n$).

- SEQ = sequence number of PDU which E_i would transmit next.
- REQ_j = sequence number of PDU which E_i expects to receive next from E_j.
- AL_{jk} = sequence number of PDU which E_i knows that E_k expects to receive next from E_k.

- $minAL_j$ = minimum of $AL_{j1}, ..., AL_{jn}$.
- PAL_{jk} = sequence number of PDU which E_i knows that E_k expects to pre-acknowledge next from E_j.
- $minPAL_j$ = minimum of $PAL_{j1}, ..., PAL_{jn}$.
- PEQ_j = sequence number of PDU which E_j expects to pre-acknowledge next from E_j.

Each PDU p from E_i has the following control information for each E_j, in addition to $p.SRC$, $p.SEQ$, and $p.PRI$.

- $p.ACK_j$ = sequence number of a PDU which E_i expects to receive next from E_j.

Each entity E_i broadcasts a PDU p according to the following *transmission* procedure. Here, *enqueue*(L, p) denotes an operation to put p in the tail of a log L. *broadcast*(p) is an operation to broadcast p by using the 1C service.

[Transmission procedure]
$p.SEQ := SEQ$; $SEQ := SEQ + 1$;
$p.ACK_j := REQ_j$ $(j = 1, ..., n)$;
enqueue(SL_i, p); *broadcast*(p); □

When a higher-priority PDU p is received, p has to jump over lower-priority PDUs received in the receipt log. Here, a *priority-based insert* operation $\lhd$ is introduced.

[Priority-based insert] Let L be a priority-based ordered log $< p_1 \ ... \ p_m >$, and p be a PDU. A priority-based insert $L \lhd p$ is defined to be a priority-based ordered log $< p_1 \ ... \ p_{i-1} \ p \ p_i \ ... \ p_m >$ where $p.PRI > p_i.PRI$. □

For example, $< a_{[4]} \ b_{[3]} \ c_{[1]} > \lhd d_{[2]}$ is $< a_{[4]} \ b_{[3]} \ d_{[2]} \ c_{[1]} >$, and $< a_{[4]} \ b_{[3]} \ c_{[1]} > \lhd e_{[3]}$ is $< a_{[4]} \ b_{[3]} \ e_{[3]} \ c_{[1]} >$.

Each entity E_i accepts a PDU p from E_j according to the following *accept* procedure. When p is received, a *pseudo-PDU* p^* is created for p. p^* is a PDU which is the same as p except that p^* has no data. p^* is given a priority 0. E_i has two logs RRL_i and PRL_i for receiving PDUs. They are the subsequences of RL_i, and RRL_i is connected to PRL_i.

[Accept procedure]
if ($p^j.SEQ = REQ_j$) {
  $RRL_i \lhd p^{j*}_{[0]}$; $(PRL_i \ || \ RRL_i) \lhd p^j$;
  $REQ_j := p^j.SEQ + 1$;
  $AL_{hj} := p^j.ACK_h$ $(h = 1, ..., n)$;
} □

p is priority-based inserted to a concatenation of PRL_i and RRL_i. Hence, PDUs in $PRL_i \ || \ RRL_i$ are priority-based ordered. The pseudo-PDU $p^*_{[0]}$ is inserted to the tail of RRL_i. This means that the order of the pseudo-PDUs in the receipt log shows the receipt order of the PDUs.

PDUs in the receipt log are pre-acknowledged according to the following procedure.

[Pre-acknowledgment (PACK) procedure]
while ((p^j = *top*(RRL_i) is not a pseudo-PDU)
  or ((p^j is a pseudo-PDU)
  and ($p^j.SEQ < minAL_j$))) {
  p^j := *dequeue*(RRL_i); *enqueue*(PRL_i, p^j);
  if (p^j is a pseudo-PDU) {
    $PEQ_j := p^j.SEQ + 1$;
    $PAL_{hj} := p.ACK_h$ $(h = 1, ..., n)$;
  }
} □

PDUs are forwarded to the application entity in the priority-based order by enqueuing the PDUs into a log ARL_i according to the following procedure. Here, *delete*(L, p) denotes a procedure to remove a PDU p in a log L. The application entity receives the PDUs from ARL_i in the priority-based order.

[Acknowledgment (ACK) procedure]
NotEnd := TRUE;
while (*NotEnd*) {
  if (p^j = *top*(PRL_i) is not a pseudo-PDU) {
    if ($p^j.SEQ < minPAL_j$) {
      p^j := *dequeue*(PRL_i);
      enqueue(ARL_i, p^j);
      delete(PRL_i, p^{j*});
    }
  } else *NotEnd* := FALSE;
} □

4.2 Failure

In our system, the 1C service is used as the underlying service. In this service, some entity E_i may fail to receive some PDU. E_i detects the PDU loss by using the sequence number of PDUs. If E_i detects that it fails to receive a PDU, all

the entities agree on which PDU they fail to receive by broadcasting the information on *REQ*. In [23, 25, 26], Then, every entity rejects all the PDUs following the lost PDU in the receipt log. The PDUs rejected are rebroadcast. That is, *go-back-n* protocol [27] is used.

5 Starvation-Free PriTO Protocol

In the PriTO protocol, PDUs are forwarded to the application entities in the priority-based order. One problem is that lower-priority PDUs can be left waiting indefinitely in the receipt log even if they are acknowledged. That is, a PDU p has to be left waiting in the receipt log until higher-priority PDUs which have jumped over p are acknowledged. In order to resolve the starvation problem, we introduce a *run synchronization* protocol to end the current run and to start a new run. By using this protocol, PDUs left waiting in the receipt log for the prefixed time are forced to be delivered to the application entity.

Each entity E_i has a variable $TOSEQ_h$ (h = 1,...,n) which denotes a maximum sequence number of timed out PDU from E_h. Initially, each $TOSEQ_h = NIL$. When each PDU p is acknowledged, a timer starts for p.

[Run synchronization protocol]

(1) The timer for a PDU p^h is expired in E_i. E_i stops the PACK and ACK procedures while E_i accepts PDUs. $TOSEQ_h := p^h.SEQ$. E_i broadcasts a *Run-Sync* PDU s where $s.TOSEQ_j = TOSEQ_j$ and $s.PEQ_j = PEQ_j$ (j = 1,...,n). s carries information on which PDUs are timed out and until which PDUs from each entity are pre-acknowledged in $s.TOSEQ_j$ and $s.PEQ_j$, respectively.

(2) Suppose that E_j receives the Run-Sync s from E_i. E_j stops the PACK and ACK procedures while PDUs are accepted. Then, $TOSEQ_h := s.TOSEQ_h$ if $TOSEQ_h = NIL$ or $TOSEQ_h < s.TOSEQ_h$ (h = 1,...,n). If E_j finds that the timer for q^k is expired, and $TOSEQ_k < q^k.SEQ$, then $TOSEQ_k := q^k.SEQ$ (k = 1,...,n). E_j broadcasts a *Run-Sync-Pack* PDU sp where $sp.TOSEQ_h = TOSEQ_h$ and $sp.PEQ_h = PEQ_h$ (h = 1,...,n).

(3) If each entity E_j receives Run-Sync or Run-Sync-Pack PDUs from all the entities in the cluster, E_j broadcasts a *Run-Sync-Ack* PDU sa where $sa.TOSEQ_h = TOSEQ_h$ and $sa.PEQ_h = PEQ_h$ (h = 1,...,n).

(4) Suppose that every E_j receives the Run-Sync-Ack PDUs from all the entities. Here, all the entities have the same $TOSEQ_h$ and PEQ_h (h = 1,...,n). First, the PACK procedure is executed and all the pseudo-PDUs which are pre-acknowledged are moved from RRL_j to PRL_j. Next, the acknowledged PDUs which precede the PDU timed out lastly, and the timed out PDUs are moved from PRL_j to ARL_j in the priority-based order. Then, the PACK and ACK procedures are restarted. □

After the steps from (1) to (3) in the run synchronization protocol, every entity agrees on which PDUs are pre-acknowledged and are timed out. Here, since E_i stops the PACK and the ACK procedures, AL is not changed even if E_i receives PDUs during the protocol execution. Further, at (4), PDUs which are acknowledged and are timed out are moved to the application entity in the priority-based order. Here, the current run is forwarded to the application entity. As a consequence, every application entity receives the same PDUs in the same priority-based order.

[Example] Figure 5 shows an example of the run synchronization. There are three entities E_1, E_2, and E_3.

(1) First, E_1, E_2, and E_3 receive the PDUs as shown in (1) of Figure 5. For example, E_1 accepts PDUs a, b, c, d, e, f, g, h, ... in this order which is denoted by the sequence of the pseudo-PDUs, and a, b, c, d, e, f are already pre-acknowledged in E_1. a and d are forwarded to the application entities in E_2 and E_3, but not in E_1 yet. Suppose that the time out occurs for a in E_1 and b in E_3. E_1 and E_3 broadcast Run-Sync PDUs.

(2) E_1, E_2, and E_3 broadcast Run-Sync-Pack and Run-Sync-Ack PDUs. Every entity agrees that b and PDUs preceding b are timed out by checking $TOSEQ$. PDUs preceding h are pre-acknowledged by checking PEQ, and all the PDUs preceding e are acknowledged.

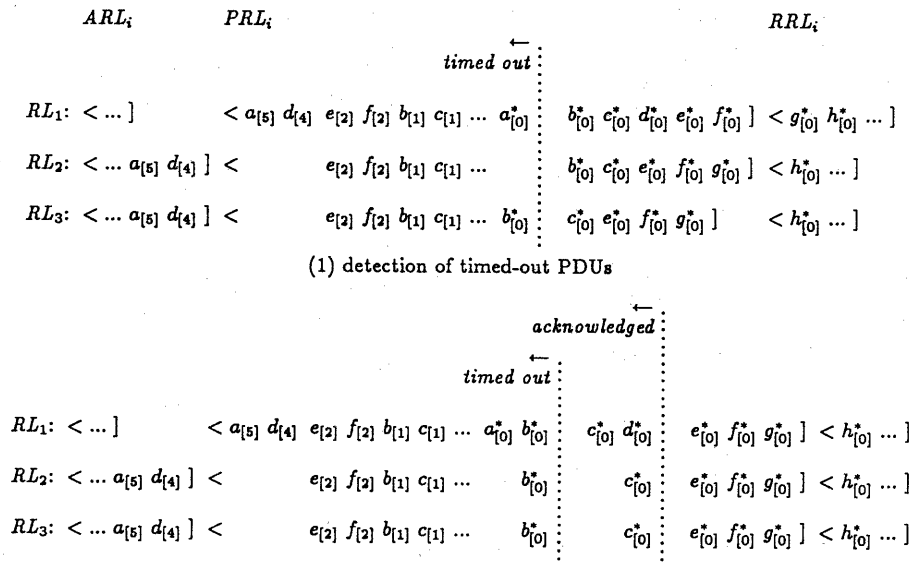

Figure 5: Example of run synchronization

(3) Then, a, d, and b are forwarded to the application entity in E_1. e and f are not forwarded because they are not acknowledged. c is not forwarded because it is not timed out although it is acknowledged. Since a and d are passed already, only b is passed to the application entities in E_2 and E_3. E_1, E_2, and E_3 have the same priority-equivalent runs, i.e. $< a_{[5]}\ d_{[4]}\ b_{[1]}]$. □

6 Concluding Remarks

In this paper, we have discussed a broadcast protocol which provides priority-based receipt ordering of PDUs by using the 1C service. Furthermore, the receipt sequence of PDUs is partitioned into runs. PDUs in each run are ordered according to the priority of the PDUs. If there exist some PDUs which are acknowledged already but are left waiting in the receipt log for a long time, they are forced to be forwarded to the application entities. Here, a run which includes the PDUs is created. By this scheme, every entity receives the same sequence of the same runs while a starvation problem is resolved. By the protocol, applications where various kinds of data are broadcast in a group of entities can be easily realized. It is implemented already in Sun workstations interconnected by the Ethernet which is used as the 1C service.

References

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

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

[3] Chanson, S., Neufeld, G., and Liang, L., "A Bibliography on Multicast and Group Com-

munications," *ACM SIGOPS OS Review*, Vol.23, No.4, Oct. 1989.

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

[5] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," *Comm. ACM*, Vol.34, No.1, 1991, pp.38-58.

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

[7] Garcia-Molina, H. and Spauster, A., "Message Ordering in a Multicast Environment," *Proc. of the 9th IEEE ICDCS*, 1989, pp.354-361.

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

[9] International Standards Organization, "OSI – Connection Oriented Transport Protocol Specification," ISO 8073, 1986.

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

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

[12] Kurose, J. S., Schwartz, M., and Yemini, Y., "Multiple-Access Protocols and Time-Constrained Communication," *ACM Computing Surveys*, Vol.16, No.1, 1984, pp.43-70.

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

[14] Liu, M. and Papantoni-Kazakos, P., "A Random Access Algorithm for Data Networks Carrying High Priority Traffic," *Proc. of the 9th IEEE INFOCOM*, 1990, pp.1087-1094.

[15] Luan, S. W. and Gligor, V. D., "A Fault-Tolerant Protocol for Atomic Broadcast," *IEEE Trans. Parallel and Distributed Systems*, Vol.1, No.3, 1990, pp.271-285.

[16] Melliar-Smith, P. M., Moser, L. E., and Agrawala, V., "Broadcast Protocols for Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, Vol.1, No.1, 1990, pp.17-25.

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

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

[19] Nakamura, A. and Takizawa, M., "Design of Reliable Broadcast Communication Protocol for Selectively Partially Ordered PDUs," *Proc. of the IEEE COMPSAC'91*, 1991, pp.673-679.

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

[21] Schneider, F. B., Gries, D., and Schlichting, R. D., "Fault-Tolerant Broadcasts," *Science of Computer Programming*, Vol.4, No.1, pp.1-15, 1984.

[22] Sharrock, S. M. and Du, D. H. C., "Efficient CSMA/CD-Based Protocols for Multiple Priority Classes," *IEEE Trans. Computers*, Vol.38, No.7, 1989, pp.943-954.

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

[24] Takizawa, M., "Design of Highly Reliable Broadcast Communication Protocol," *Proc. of IEEE COMPSAC'87*, 1987, pp.731-740.

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

[26] Takizawa, M. and Nakamura, A., "Reliable Broadcast Communication," *Proc. of IPSJ InfoJapan*, 1990, pp.325-332.

[27] Tanenbaum, A. S., "Computer Networks (2nd ed.)," *Prentice-Hall*, 1989.