Flexible General Purpose Communication Primitives for Distributed Systems

Roberto Baldoni, Roberto Beraldi
Dipartimento di Informatica e Sistemistica
Universitá di Roma "La Sapienza"
Via Salaria 113, Roma, Italy
baldoni,beraldi@dis.uniroma1.it

Ravi Prakash
Department of Computer Science
University of Rochester
Rochester, New York 14627, U.S.A
prakash@cs.rochester.edu

Abstract

This paper presents the slotted-FIFO communication mode that supports communication primitives for the entire spectrum of reliability and ordering requirements of distributed applications: FIFO as well as non-FIFO, and reliable as well as unreliable communication. Hence, the slotted-FIFO communication mode is suitable for multimedia applications, as well as non real-time distributed applications. As FIFO ordering is not required for all messages, message buffering requirements are considerably reduced. Also, message latencies are lower. We quantify such advantages by means of a simulation study. A low overhead protocol implementing slotted-FIFO communication is also presented. The protocol incurs a small resequencing cost.

1. Introduction

A major focus of recent research in data communication over computer networks has been to ensure a reliable and, if possible, ordered flow of data. The definition of communication modes, such us FIFO and causal ordering [8, 9], and flow control using windowing and positive/negative acknowledgments [14] have been motivated by such research. At the conceptual level, communication subsystems have been usually modeled as being reliable with unpredictable, yet finite, message transmission time. Such a communication model is relevant for database applications that rely on ordered and reliable flow of data. Until recently, database applications were dominant among all applications executing in a networked environment.

However, satisfying the reliability and ordering requirements for all messages may entail *extensive buffering*, and in some instances *long latencies* in message delivery [8, 9]. Moreover, many applications require communication with relaxed reliability and/or ordering constraints [12, 15]. For example, multimedia applications have to handle audio and video streams of data that are played back in real-time at the destination site. These streams can tolerate some loss of information as long as these losses do not cause a degradation in the desired *quality of service* (QoS) for the application [13, 16]. New communication paradigms should thus be suitably defined. One such communication paradigm is Δ-causal ordering [6, 7, 15] that, assuming application messages have a lifetime, delivers as many messages as possible in their lifetime in such a way that these deliveries are causally ordered.

A good example of the requirements of multimedia communication applications is the MPEG video compression standard [11]. MPEG compressed video consists of three kinds of frames. The *Intraframe* frames have no dependencies and can be decoded independently of other frames. A *Predicted* frame can be decoded only if the previous frame is available. A *Bidirectional* frame requires the closest Intraframe or Predicted frame before and after it for decoding. Thus, MPEG induces dependencies between some frames. It is obvious that the loss of some frames in a sequence may lead to an inability to decode all the subsequent frames which depend on it, while the loss of another frame may not affect the decoding of any frame. This leads to communication with different priorities, ordering and reliability constraints for different data units within the same data stream.

A trivial approach to handle such a data flow could be to send data on two separate channels. The first frames of distinct sequences are sent along a channel supporting reliable and FIFO delivery. Succeeding frames of the sequence (that are dependent on the first frame) can be sent along a channel that can lose and/or reorder them. This solution raises issues pertaining to re-synchronization and maintaining dependencies among frames at the destination site of the two distinct data flows along different channels.

This paper makes three contributions: (i) we introduce a communication mode called *slotted-FIFO* ordering that allows messages with different ordering and reliability constraints to be interleaved on the same channel while maintaining certain order dependencies among messages, (ii) propose a protocol based on sequence numbers to implement *slotted-FIFO* communication, and (iii) evaluate the proposed protocol's performance through simulation experiments. One would expect that combining message ordering and reliability would incur high overheads. However, the simulation results indicate otherwise.

The *slotted-FIFO* communication mode offers a set of synchronization primitives for sending messages, namely, FR*–send*, $\bar{F}R$*–send*, $F\bar{R}$*–send*, and $\bar{F}\bar{R}$*–send*, to be read as *FIFO and Reliable* send, *non-FIFO and Reliable* send, *FIFO and non-Reliable* send, and *non-FIFO and non-Reliable* send, respectively. For a message m, sent by process p to process q along the channel $c_{p,q}$ using an $FR - send$ primitive, the following properties are ensured at process q: 1) a message sent before m along $c_{p,q}$, if delivered to q, is delivered before m, 2) a message sent after m along $c_{p,q}$, if delivered to q, is delivered after m. If the message is sent using a reliable primitive, the delivery eventually occurs. The case where all messages are sent using the reliable primitive $FR - send$ corresponds to the *two way flush* primitive proposed by Ahuja [1, 3] to increase concurrency on reliable channel compared to FIFO channels. The interested reader can refer to [5] for a qualitative comparison between slotted-FIFO communication mode, flush primitives [1] and hierarchical channels [4].

Two successive message send events, corresponding to messages m and m', using the $FR - send$ primitive define a *slot* S along the channel $c_{p,q}$. A message m_1 sent using the $\bar{F}R - send$ primitive, executed inside S, ensures that m_1 is delivered after m and before m'. A pair of message m_1 and m_2 sent using $F\bar{R} - send$ primitives within S ensure that m_1 and m_2, if delivered to q, are delivered after m, before m', and in their sending order. A message m_1 sent using the $\bar{F}\bar{R} - send$ primitive ensures that m_1, if delivered to q, is delivered after m and before m'. If the loss of some messages will not adversely affect the quality of the service, such messages can be sent in an unreliable fashion using the $F\bar{R}$ or $\bar{F}\bar{R}$ primitives.

Approaches similar to slotted-FIFO channels have been proposed in the literature. Specifically, in the definition of new error control schemes for interprocess communication that provide variable degrees of error recovery according to application's requirements [12] and to implement the transport layer of a group communication system to support multimedia streams of data [10].

Compared to the reliable FIFO channels, we get more concurrency and two basic advantages: (i) substantial reduction in the required buffer space at the receiver process and (ii) short message latencies. Indeed, the destination process does not have to buffer the reliable messages that overtake the unreliable messages to ensure the delivery of the latter and a message that cannot be lost does not have to await the delivery of unreliable messages. These advantages are quantified by a simulation study, showing that the average buffer requirements and the message latency can be reduced up to one tenth.

The remainder of this paper is organized as follows: Section 2 contains a discussion of some applications for which the slotted-FIFO channels might be useful. The system model is described in Section 3. Section 4 presents the slotted-FIFO communication mode and Section 5 shows an implementation based on sequence numbers. Section 6 reports the simulation results. Finally conclusions are presented.

2. Applications of Slotted-FIFO Channels

Slotted-FIFO channels provide varying degrees of reliability and ordering for message communication. Such flexibility is desirable for a variety of applications [12, 15]. For example:

Video telephony: In video telephony the audio and video data streams may be sent along different channels; typically high bandwidth channels for video signals and low bandwidth channels for audio signals. It is to be noted that demands are being placed on the audio and video channels by other applications executing concurrently in the network. Hence, the probability distribution of the propagation times of the two different data streams between any source-destination pair may be different, depending on the loads on the audio and video channels in the network [16]. Ideally, the corresponding audio packet and video frame should be played simultaneously at the destination. However, such synchronization for every audio packet - video frame pair would require very little variability between the latencies of the audio and video channels, and/or extensive buffering at the receiver. As such requirements are expensive to meet, the following strategy can be employed. Periodically, corresponding audio packets and video frames are sent using the FR primitive, and their delivery to the receiver process is synchronized. The time interval between successive synchronization points corresponds to a slot. During a slot the audio and video signals can be sent using the $F\bar{R}$ and $\bar{F}\bar{R}$ primitives. This is because an occasional loss of a small number of audio packets and video frames is beyond human perception. The slot duration should be determined based on the characteristics of the audio and video channels so that the audio and video streams do not get significantly out of synch during a slot.

<u>**MPEG video transmission:**</u> MPEG compressed video consists of three kinds of frames [11]. The *Intraframe* frames have no dependencies and can be decoded independently of other frames. A *Predicted* frame can be decoded only if the previous frame is available. A *Bidirectional* frame requires the closest Intraframe or Predicted frame before and after it for decoding. As the loss of an *Intraframe* frame renders all the dependent *Predicted* frames that follow it useless, the *Intraframe* frames should be sent using the FR primitive. As loss of a few of the following *Predicted* frames leads to a marginal degradation in the quality of service, such frames can be sent using $F\bar{R}$ primitive. As the decoding of *Predicted* frames is dependent on the decoding of the preceding *Intraframe* and *Predicted* frames, employing the $\bar{F}R$ and $\bar{F}\bar{R}$ primitives may lead to non-FIFO delivery of the *Predicted* frames. Such non-FIFO delivery may not only lead to a degradation in the quality of service, but also unpredictable logical errors in the decoding of the *Predicted* frames.

<u>**Sliding window protocol:**</u> Enforcing FIFO order among the acknowledgments in a sliding window protocol is both expensive and unnecessary. An acknowledgment for a later packet implicitly acknowledges the reception of earlier packets at the receiver. Hence, to minimize the cost of acknowledgments they can be sent using the $\bar{F}\bar{R}$ primitive. If loss of acknowledgments is not acceptable, they can be sent using the $\bar{F}R$ primitive. When acknowledgments for earlier packets are received by the sender after the acknowledgments for later packets, such acknowledgments are simply ignored. In doing so, we match the basic idea of a general-purpose sliding window protocol [14].

3. System Model

A pair of processes p, q is connected by a communication network, or simply *network*. We assume the network is well connected, but unreliable and asynchronous. Each process runs on a processor. The processors do not have a global clock, they do not share memory. Failure handling is not considered.

We assume that each process consists of an *application layer* (AL), a *Slotted-FIFO layer*, (SL), and a transport layer (TL), as shown in Figure 1. The application layer can utilize the slotted-FIFO primitives, and generate *XY-send* events to the SL, where XY is a label belonging to the set $\{FR, F\bar{R}, \bar{F}R, \bar{F}\bar{R}\}$, and can accept *delivery* events from the SL. SL is responsible for message delivery according to the slotted-FIFO discipline. The SL layer can generate *transport-send* events to the transport layer and can accept *transport-receive* events from the transport layer. TL is endowed with mechanisms such as positive/negative acknowledgment and retransmissions to ensure, if requested, reliable receipt of messages (the two reliable communication primitives invoke these mechanisms). TL generates *net-send* event to the network and accepts *net-receive* event from the network.

We say that a message is *sent* when the corresponding XY-send event is generated; a message is *delivered* when the corresponding delivery event is generated; a message is *received* when the corresponding transport-receive event is generated; a message is said to have *arrived* when the corresponding net-receive event is generated.

At the application level a pair of processes, p, q, is connected by a directed and asynchronous logical channel $c_{p,q}$ (p is the sender and q is the destination process). Along this channel, a finite sequence of messages M are sent. The messages in the sequence can be labeled as $m.0, m.1, \ldots m.k$, where $m.i$ denotes the $(i+1)^{st}$ message sent by p. This channel supports four primitives to send a message in an ordered and/or reliable way: FR*–send*, $F\bar{R}$*–send*, $\bar{F}R$*–send*, $F\bar{R}$*–send* and $\bar{F}\bar{R}$*–send*. A channel can drop messages sent by unreliable primitives.

4. The slotted-FIFO Communication Mode

A message sequence sent along a channel can be divided into subsequences based on the ordering and reliability characteristics of the constituent messages. Let $XY.j$ be the $(j+1)^{st}$ message of the subsequence M_{XY} sent by invoking the $XY - send(m)$ primitive (with $XY \in$

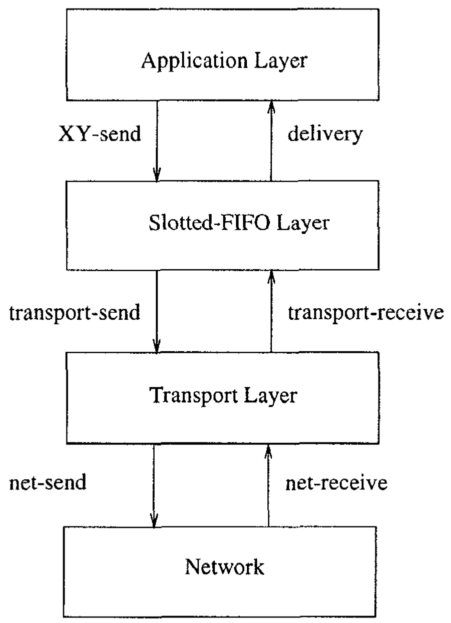

Figure 1. The structure of a process.

$\{FR, F\bar{R}, \bar{F}R, \bar{F}\bar{R}\}$). The messages in this subsequence can be represented as $XY.0, XY.1, \ldots XY.k_{XY}$, where $k_{XY} \leq k$.

Let there be a function $f : M_{XY} \rightarrow M$ which given the identity of a message sent by an $XY - send$ primitive determines the identity of the same message in M. We also assume that $m.0 = f(FR.0)$ and $m.k = f(FR.k_{FR})$, i.e., the first and last messages in the message sequence are sent using the $FR - send$ primitive. Now, we can define a "slot" as follows.

Definition 4.1 *A slot S_i is the set MS_i of messages $m.x$ (with $MS_i \subset M$),* sent *by p along $c_{p,q}$, such that $f(FR.i) < m.x < f(FR.(i+1))$. The* projection of a slot $\Pi(S_i)$ *is the set of messages* delivered *to process q.*

Also, let $m.x_h$ and $m.x_w$ be two messages belonging to $\Pi(S_i)$, we denote $m.x_h \rightsquigarrow m.x_w$ iff the delivery of $m.x_h$ occured before the delivery of $m.x_w$. We are in the position to define the slotted-FIFO communication mode:

Definition 4.2 *A pair of processes p, q respects slotted-FIFO ordering on $c_{p,q}$ iff:*

1. $\forall FR.h, FR.w \in M \;::\; (h < w)\; iff\; FR.h \rightsquigarrow FR.w;$

2. $\forall \bar{F}R.h \in S_i \;::\; \bar{F}R.h \in \Pi(S_i);$

3. $\forall \bar{F}\bar{R}.h \in \Pi(S_i) \;::\; \bar{F}\bar{R}.h \in S_i;$

4. $\forall F\bar{R}.h, F\bar{R}.w \in \Pi(S_i) \;::\; (h < w)\; iff\; F\bar{R}.h \rightsquigarrow F\bar{R}.w.$

The four rules mean the following: (i) FIFO and Reliable messages are always delivered in the order they are sent along a channel, (ii) a Reliable message is always delivered to its destination in the same slot, (iii) a non-FIFO non-Reliable message, if delivered to the destination, should be delivered in the same slot that it is sent, and (iv) two FIFO non-Reliable messages delivered in the same slot must have been sent in the order they were delivered. Examples of slotted-FIFO communications are depicted in Figure 2.a.

5. A Simple Implementation of the Slotted-FIFO layer

An implementation of the Slotted-FIFO layer consists of defining a protocol between a XY-send procedure invoked by the application layer and a message handler (namely RECEIVE) which is instantiated each time a transport-receive event is generated by the transport level. Actually, each message m, received at the slotted FIFO layer, is associated with a WAIT (or delivery) condition. If the condition is satisfied m, is delivered. If the message type is unreliable and is out of its slot, m is discarded. Otherwise m is buffered at the slotted-FIFO layer until its WAIT condition becomes true. When an FR message is going to be delivered, first buffered unreliable FIFO messages that should precede the FR message are delivered.

5.1. The Sending Process

The sending process maintains the following three variables:

- *slot*: the current slot number;
- $S_{\bar{F}R}$: the sequence number of the next Non-FIFO and reliable message in the current slot;
- $S_{F\bar{R}}$: the sequence number of the next FIFO and unreliable message in the current slot;

Each message is equipped with control information stored in its ST structure whose fields are as follows:

- *slot*: integer indicating the slot associated with the message;
- *type*: a char indicating the type of the message;

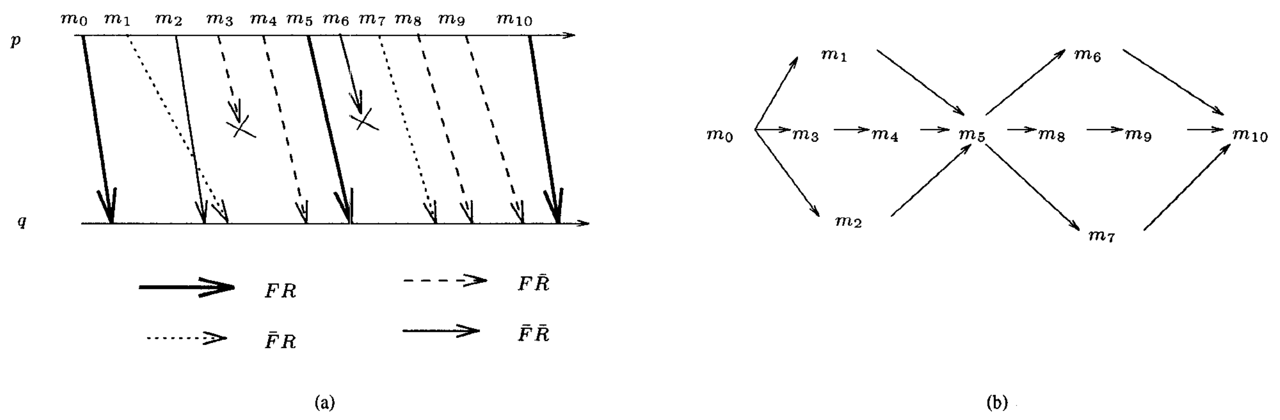
m_0 m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_{10}
p
q
FR
$\bar{F}R$
$F\bar{R}$
$\bar{F}\bar{R}$
(a)
m_0 m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_{10}
(b)

Figure 2. An example of slotted-FIFO communication and the corresponding partial order generated by the relation $\preceq$.

- *order*: integer whose value depends on the type of the message. If the message is of type FR, it represents the number of $\bar{F}R$ messages sent in the previous slot (i.e., $S_{\bar{F}R}$). If the message is of type $F\bar{R}$,it represents the number of FR' messages sent so far (including this message) in the current slot. i.e., $S_{F\bar{R}}$.

Implementation of the XY-send primitive is given in Figure 3. First, fields *slot* and *type* of ST are updated (S1). When a reliable FIFO message has to be sent, the order field is set to the current value of the $S_{\bar{F}R}$ counter (S2). This is needed to inform the receiving process of the number of reliable messages sent during the current slot (i.e., the slot being closed by the FR message). The FR message should not be delivered to the receiving process until all the preceding reliable messages in the slot have been delivered. The slot counter is incremented (S3), and the FIFO unreliable counter is reset (S4). When a non-FIFO reliable message has to be sent, the $S_{\bar{F}R}$ counter is incremented. When sending an unreliable FIFO message, the $S_{F\bar{R}}$ counter is incremented, and the order field of ST is set to the current value of the $S_{F\bar{R}}$ counter.

It is to be noted that a process need not maintain sequence number information for non-FIFO unreliable messages in the form of an $S_{\bar{F}\bar{R}}$ variable. Also, the *order* field of the ST structure sent with such messages is unassigned. This is because the only constraint relevant for the delivery of $\bar{F}\bar{R}$ class of messages is the slot number.

The structure ST allows us to model messages as a partial order. This will be useful to define an ordering in the activation of multiple suspended WAIT conditions as explained later.

1. Let ST_m be the control information structure for the message m. Let m and m' be two messages, we say that m precedes m', denoted $m \prec_s m'$, iff $ST_m.slot < ST_{m'}.slot$. The precedence relation $\prec_s$ represents the transitive closure of precedence among successive slots.

2. Let m and m' be two messages such that $ST_m.slot = ST_{m'}.slot$ and $ST_m.type = ST_{m'}.type = F\bar{R}$, we say that m precedes m', denoted $m \prec_f m'$, iff $ST_m.order < ST_{m'}.order$. The precedence relation $\prec_f$ represents precedence within a slot due to FIFO ordering.

3. Let m and m' be two messages, we say that m precedes m', denoted $m \prec_{es} m'$, iff $ST_m.slot = ST_{m'}.slot$ and $ST_{m'}.type = FR$. The precedence relation $\prec_{es}$ denotes that messages in a slot precede the FR message that marks the end of the slot.

Now, we denote $\preceq$ the transitive closure of the union of $\prec_s$, $\prec_f$, and $\prec_{es}$. The set of messages M can be then represented as a partial order of messages $\hat{M} = (M, \preceq)$. Two messages m and m' are concurrent iff $\neg(m \preceq m')$ and $\neg(m' \preceq m)$. In Figure 2.b the partial order $\hat{M} = (M, \preceq)$ of the message scheduling of Figure 2.a is shown.

5.2 The Receiving Process

The receiving process at the slotted-FIFO layer manages the following four variables:

- *slot*: an integer that stores the current slot number;

```
init
slot=0; $S_{\bar{F}R}$=0;$S_{F\bar{R}}$=0;
procedure XY-SEND($m,t,j$)
/*m is the message content*/
/* t is the type */
/* j is the destination process*/
begin
    $ST.slot = slot$;  $ST.type = t$;                    (S1)
    case    t  of
    $FR$ :  $ST.order = S_{\bar{F}R}; S_{\bar{F}R} = 0$;       (S2)
            $slot = slot + 1$;                         (S3)
            $S_{F\bar{R}} = 0$;  $break$;                    (S4)
    $\bar{F}R$ :  $S_{\bar{F}R} = S_{\bar{F}R} + 1$;  $break$;         (S5)
    $F\bar{R}$ :  $S_{F\bar{R}} = S_{F\bar{R}} + 1$;                 (S6)
            $ST.order = S_{F\bar{R}}$;  $break$;            (S7)
    $\bar{F}\bar{R}$ :  $break$;                               (S8)
    endcase
    transport-send($m,ST$);                          (S9)
end.
```

Figure 3. A simple XY-send primitive implementation.

- $R_{\bar{F}R}$: an integer that stores the number of reliable non-FIFO messages delivered in the current slot;

- $R_{F\bar{R}}$: an integer that stores the sequence number of the last non-reliable FIFO message delivered in the current slot;

- *end_slot*: a Boolean variable indicating that an FR message has arrived and the end of the slot is imminent;

The RECEIVE message handler is shown in Figure 4. Each received message is associated with a WAIT condition which depends on the message type (R2, R8, R11, R15). Let the type of the message received be FIFO and reliable. Statement (R2) permits further processing of the message, and ultimately its delivery, only when the value of the local *slot* variable is equal to the value of the message's $ST.slot$ field and the message's $ST.order$ field is equal to the number of reliable messages sent in the current slot. This ensures that the reliable messages sent by the source in the current slot are delivered to the destination before messages belonging to the next slot are delivered. Both $R_{\bar{F}R}$ and $R_{F\bar{R}}$ counters are also reset (R3,R4).

If a non-FIFO reliable message is received, the receiving process waits until the $ST.slot$ field equals the current slot (R8). The $R_{\bar{F}R}$ field is incremented to reflect the delivery of a non-FIFO reliable message (R9), and the message is delivered (R10).

If an unreliable non-FIFO message is received and the value of its $ST.slot$ field is lower than the current slot, the message is discarded (R11, R12). Such a situation arises if the message has been overtaken on its way to the destination by a reliable FIFO message and arrives after the expiration of its slot. Otherwise it is delivered as soon as the the slot field is equal to the current slot (R13, R14).

Similarly, an unreliable FIFO message is discarded if the value of its $ST.slot$ field is either less than the current slot or if an FR message has arrived and the end of its slot is imminent (R15, R16). Otherwise its delivery is delayed until one of the the conditions in (R17) is satisfied. The first condition (i.e., $ST.slot == slot$ and $ST.order{==}R_{F\bar{R}} + 1$)) is the typical FIFO condition. The second condition (i.e., $ST.slot == slot$ and *end_slot*) indicates that the end of the slot is imminent. So, all pending $F\bar{R}$ messages must be immediately delivered in the correct order before the received FR message be delivered.

Deadlock Avoidance

If the WAITs in the case statements are busy-waits, deadlocks can arise. For example, let a sender send an $\bar{F}R$ message followed by an FR message to the receiver. Also, let the FR message overtake the $\bar{F}R$ message. If busy-waits are employed at the receiver, the receiver process will be spinning on the condition $ST.order == R_{\bar{F}R}$. On the arrival of the $\bar{F}R$ message the processor will not be able to handle it and increment $R_{\bar{F}R}$ as the processor cycles are being monopolized by the busy-wait.

Hence, it is important to ensure the following:

- When an instance of the RECEIVE message handler reaches a WAIT statement and the condition in the statement is not true, that particular instance of the RECEIVE message handler is suspended and gives up control of the receiver process.

- A suspended instance of the RECEIVE message handler is activated when the condition on which the procedure is waiting becomes true due to execution of statements in other instance(s) of the RECEIVE message handler at the same site.

- If delivery conditions of multiple suspended instances

```
init
slot=0; $R_{\bar{F}R}$=0;
$R_{F\bar{R}}$=0; end_slot=$false$;
when an $(m, ST)$ event is
accepted from the Transport-Layer
/*m is the message content, */
/* ST is the control information*/
begin
case  ST.type  of
$FR$:  end_slot =$true$;                    (R1)
       wait(slot==ST.slot $and$             (R2)
       ST.order==$R_{\bar{F}R}$);           (R2)
       $R_{\bar{F}R}$=0;                    (R3)
       $R_{F\bar{R}}$=0;                    (R4)
       end_slot=$false$;                    (R5)
       slot=slot+1;                         (R6)
       deliver($m$);                        (R7)
       break;
$\bar{F}R$:  wait(slot==ST.slot);           (R8)
       $R_{\bar{F}R}$=$R_{\bar{F}R}+1$;     (R9)
       deliver($m$);                        (R10)
       break;
$\bar{F}\bar{R}$:  if (ST.slot < slot)      (R11)
       then discard($m$); break;            (R12)
       else wait(ST.slot==slot);            (R13)
          deliver($m$);                     (R14)
          break;

$F\bar{R}$:  if ((ST.slot < slot) $or$      (R15)
       end_slot)
       then discard($m$);                   (R16)
       else wait((ST.slot==slot $and$(R17)
       (ST.order==$R_{F\bar{R}}$+1 $or$ end_slot));
          $R_{F\bar{R}}$=$ST.order$;        (R18)
          deliver($m$);                     (R19)
          break;
endcase
end.
```

Figure 4. A simple RECEIVE message handler implementation.

of the RECEIVE message handler become true simultaneously (for example the ones associated with messages m and m'), they are activated in an order consistent with the partial order given by the relation $\preceq$, i.e., m is activated before m' if $m \preceq m'$. If m and m' are concurrent they can be activated in any order.

- Finally, to ensure consistency of updates to the condition variables the following statement sequences are executed in an atomic fashion: $(R3) - (R7)$, $(R9) - (R10)$, and $(R18) - (R19)$.

We would like to remark that a busy-wait implementation using preemptable threads is possible. However, in that case we need to take care of priority among threads.

6. Simulation Study

In this section we measure the performance of the slotted-FIFO layer proposed in the previous section. We assume FR-send primitives are generated by AL (Application Layer) at fixed time intervals of length T_s, referred to as the *slot length*. In the interval between two FR-send messages, primitives of the other types are generated according to a Poisson process with rate λ, and are labelled as XY type with probability P_{XY}. The SL stores waiting messages inside a *resequential buffer*. The network is characterized by the *reliability*, given as the probability, P_{succ}, that a sent message is not lost. No correlation is assumed between losses.

In order to guarantee reliable trasmissions, an error-detection mechanism, based upon a classical positive retransmission protocol using time-out, is embedded in TL. Specifically, after a transport-send is executed for a reliable message m, m is buffered and a net-send event is generated by TL each time a *time-out* period T_{to} expires. Moreover, we use message numeration at TL layer to avoid message duplication. We assume no flow control and infinite buffer size at TL.

We define *network delay*, T_{ntw}, as the time elapsed between the the net-send event of a message m and the *arrival* (see Section 3) of m at the receiver; the *transport delay*, T_{tra}, as the time elapsed between the transport-send event of a message m and the *receipt* of m; the *delivery delay*, T_d, as the time elapsed between the acceptance of the transport-receive event of a message m by SL and the *delivery* of m.

For messages that are not lost T_{ntw} is an exponentially distributed random variable with mean μ_{ntw}. We assume

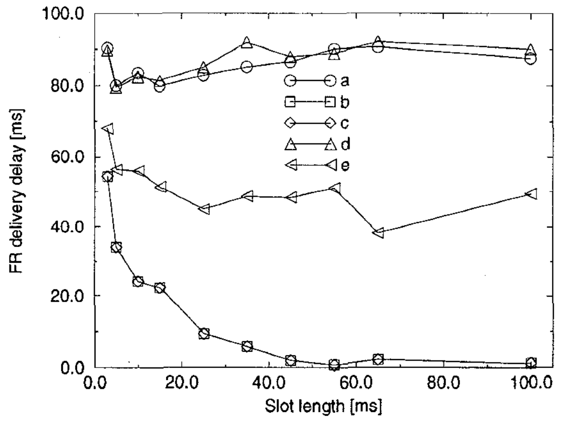

Figure 5. FR delivery delay vs slot length.

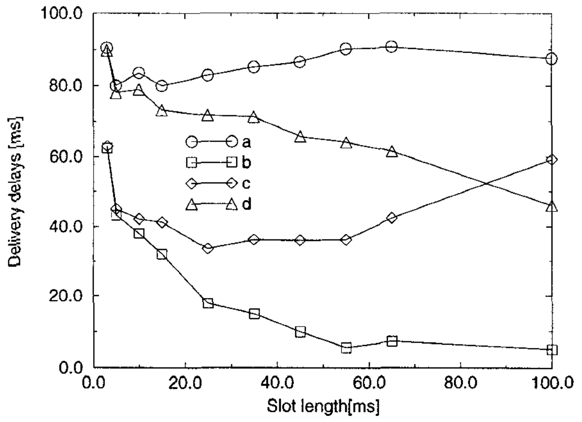

Figure 6. Delivery delay of messages inside a slot.

$T_{tra} = T_{ntw}$ for unreliable messages that are not lost [1], and $T_{tra} = \alpha T_{to} + T_{ntw}$ for reliable message, where α is a geometrically distributed random variable denoting the number of retransmissions:

$$prob\{\alpha = k\} = P_{succ}(1 - P_{succ})^k, \;\; k \geq 0.$$

6.1 Results of the Experiments

Results presented in this section were obtained with the following parameters. λ= 1 packets/ms, μ_{ntw} = 25 ms, T_{to}=50 ms, P_{succ} = 0.999. We considered five different transmission schemes: (a) $P_{FR} = 1^2$; (b) $P_{\bar{F}\bar{R}} = 1$; (c) $P_{F\bar{R}} = 1$; (d) $P_{\bar{F}R} = 1$; (e) $P_{\bar{F}\bar{R}} = P_{F\bar{R}} = P_{\bar{F}R} = 1/3$. Experiments were conducted by varying the slot length from 2 ms to 100 ms and by measuring the *average delivery delay* experienced by the different types of messages and the *average queue length* of the resequencing buffer embedded in the slotted-FIFO layer. For each value of the slot lenght we did several runs with different seeds and the result were within four percent of each other, thus variance is not reported.

Fig. 5 shows the FR delivery delay as a function of the slot length (measured in ms). FR delivery delay denotes the time required for slot reorder and is affected by reliable message arrival out of sequence. In schemes (a) and (d), it is in the range 70-80 ms and is almost independent of the slot length, since reliable transmissions are used for both messages. In schemes (b) and (c), we observed a significant reduction of the FR delivery delay. Indeed, non-reliable messages, that are received too late (i.e., after their slot) by SL, are discarded and FR messages are not forced to wait for such messages. As the slot length increases, the probability that the received FR messages are out of sequence decreases and thus the resequencing cost falls down. Scheme (e) is an example of FR delivery delay in which 33% of messages in a slot are sent in a reliable way.

[1]We assume that no delay is involved in the TL when sending a message.

[2]To compare the slotted-FIFO performance with pure FIFO channels, we allow messages within a slot to be of the FR type.

Figure 6 shows delivery delay of the messages inside a slot. It is simple to see that delivery delay is the sum of two delays. The first delay, D_1, is associated with the situation where a message in a slot reaches the destination prior to the FR message marking the beginning of that slot/end of previous slot. D_1 is higher for smaller slot intervals since small slot intervals increase the probability that reliable messages arrive out of sequence. The second delay, D_2, measures the time elapsed between the time a message m enters its slot and the delivery of m. In the non-FIFO schemes (b) and (d), $D_2 = 0$ and thus $\bar{F}R$ delivery delay decreases as the slot lenght increases. In the scheme (c), some $F\bar{R}$ messages are forced to wait until the end of the slot before they are delivered. As the slot length increases, D_2 dominates over D_1.

As far as the required buffer size is concerned, the average queue length of the resequencing buffer was used to compare the different scheme requirements. Results are shown in Figure 7. For slot wider than 20 ms, the average queue length of scheme (b) is ten times less than scheme (a). In scheme (c), until D_2 does not dominate over D_1 (till 40 msec), the average queue lenght was from one half up to one

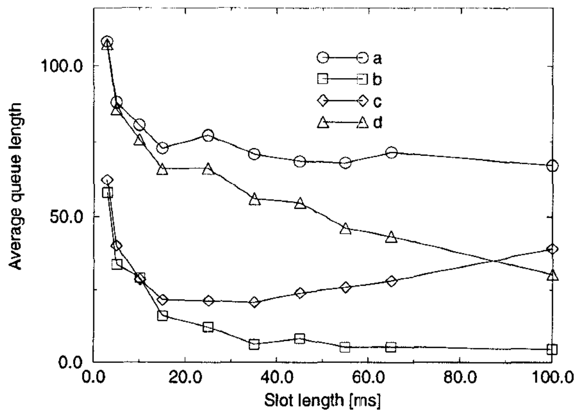

Figure 7. Buffer requirements.

third the average queue length of the reliable schemes (a) and (d). As the slot length increases, the non-FIFO scheme (d) decreases the average queue length as $D_2 = 0$ while the unreliable FIFO scheme (c) increases it as D_2 dominates over D_1. These measures can be useful to design memory space allocation used for message reordering at the slotted-FIFO layer according application's requirements.

Finally, we would like to remark that the percentage of messages dropped by the proposed slotted-FIFO layer is always below 10% for any slot width and that using a scheme (b), the one with least ordering and reliability constraints, the average buffer requirements and the message latency can be reduced up to one tenth compared to a pure FIFO channel (scheme a).

7. Conclusion

Reliable FIFO message communication has received great attention in the past. This is primarily because database applications have been the dominant network applications so far, and these applications require reliable FIFO communication. However, many applications need communication with relaxed reliability and/or ordering constraints. Multimedia applications, for example, have different quality of service requirements from database applications, i.e., loss of some packets and occasional non-FIFO delivery is acceptable. Hence, communication protocols for database applications are not suitable as well as expensive for multimedia applications.

In this paper, we presented the slotted-FIFO communication mode. It supports reliable FIFO, unreliable FIFO, reliable non-FIFO, and unreliable non-FIFO flow of data while maintaing a certain degree of ordering among them. The entire communication can be divided into slots demarcated by successive reliable FIFO messages. The reliable messages sent within a slot are always delivered to the destination process after the beginning and before the end of the slot. FIFO messages in a slot, if delivered to the destination, are delivered in the correct order. The delivery of reliable messages is never delayed on account of unreliable messages. This leads to a reduction in message buffering and latency. Loss of unreliable messages in transit is semantically similar to their late arrival, and requires no special handling.

Simulation results also show that in applications where some loss of sent messages can be allowed (such as the multimedia applications), a slotted-FIFO channel can be used to obtain better performance, in terms of buffer requirements and delivery delay, when compared to a pure FIFO channel. We showed that designing a slotted-FIFO layer is simple and this should imply a simplicity of implementation. The bookkeeping overheads are also low.

Acknowledgments

The authors would like to thank Roy Friedman of Cornell University for useful comments and discussions on an earlier version of the paper.

References

[1] M. Ahuja. Flush Primitives for Asynchronous Distributed System. *Information Processing Letters*, 34(1):5–12, February 1990.

[2] M. Ahuja. Hierarchy of Communication Speeds for Designing Concurrent Systems. Technical Report OSU-CISRC–1/91–TR1, The Ohio State University, 1991.

[3] M. Ahuja. An implementation of F-channels. *IEEE Transactions on Parallel and Distributed Systems*, 658–667, June 1993.

[4] M. Ahuja and R. Prakash. On the relative speed of messages and hierarchical channels. In *Proceedings of the* 4^{th} *IEEE Symposium on Parallel and Distributed Processing*, pages 246–253. IEEE Computer Society Press, 1992.

[5] R. Baldoni, R. Beraldi, and R. Prakash, Slotted-FIFO Communication for Asynchronous Distributed Systems. Technical Report n. 10-97, Dipartimento di

Informatica e Sistemistica, Universita' di Roma "La Sapienza", April 1997.

[6] R. Baldoni, A. Mostefaoui, and M. Raynal, Causal delivery of messages with real-time data in unreliable networks. *Journal of Real-time Systems*, Vol 10, no. 3, 245–262, 1996.

[7] R. Baldoni, R. Prakash, M. Raynal, and M. Singhal. Efficient Δ-causal broadcasting. *International Journal of Computer Systems Science and Engineering (to appear)*, 1997.

[8] K. Birman and T. Joseph. Reliable communication in presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[9] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic broadcast. *ACM Transactions on Computer Systems*, 9(3):272–314, 1991.

[10] G. V. Clocker, N. Huleihel, I. Keidar, D. Dolev. Multimedia multicast transport service for groupware. *Proceedings of the TINA'96 Conference*, 43–54, VDE Verlag, 1996.

[11] D. Gall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[12] F. Gong, G. M. Parulkar. An application-oriented error control scheme for high speed networks . *IEEE/ACM Transactions on Networking*, 4(5), 669–683, October 1996.

[13] T. Houdoin and D. Bonjour. ATM and AAL layer issues concerning multimedia applications. *Annals of Telecommunications*, 49(5):230–240, 1994.

[14] A.S. Tanenbaum. Computer Network. *Prentice-Hall International Editions*, 1996.

[15] R. Yavatkar. MCP: A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications. In *Proceedings of the* 12^{th} *IEEE International Conference on Distributed Computing Systems*, 606–613. IEEE Press, 1992.

[16] I. Wakeman. Packetized video: options for interaction between the user, the network and the codec. *The Computer Journal*, 36(1):55–66, 1993.