
Slotted-FIFO Communication for Asynchronous Distributed Systems

ROBERTO BALDONI¹, ROBERTO BERALDI¹ AND RAVI PRAKASH²

¹*Dipartimento di Informatica e Sistemistica, Università di Roma 'La Sapienza', Via Salaria 113, Roma, Italy*

Email: baldoni,beraldi@dis.uniroma1.it

²*Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688, USA*

Email: ravip@utdallas.edu

Communication protocols designed for database applications are not necessarily suitable for other applications, like multimedia communication, due to the former's requirement of reliable and ordered communication, and the latter's ability to withstand occasional losses and misordering of messages as long as real-time communication can be supported. This paper presents the slotted-FIFO communication mode that supports communication primitives for the entire spectrum of reliability and ordering requirements of distributed applications: for example, FIFO as well as non-FIFO, and reliable as well as unreliable communication. It provides communication with a run-time variable degree of reliability and/or ordering. Hence, the slotted-FIFO communication mode is suitable for applications that can work with relaxed reliability and/or ordering constraints such as multimedia applications. The protocol is simple and has low overheads. As FIFO ordering is not required for all messages, message buffering requirements are considerably reduced. Also, message latencies are lower. We demonstrate 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.

Received February 19, 1998; revised June 2, 1998

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 as FIFO and causal ordering [1, 2, 3], and flow control using windowing and positive/negative acknowledgements [4] have been motivated by such research. At the conceptual level, communication subsystems have usually been 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 [2, 3, 5]. Moreover, many applications require communication with relaxed reliability and/or ordering constraints [6, 7]. 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 [8, 9]. New communication paradigms should thus be defined. One such communication paradigm is Δ -causal ordering [7, 10, 11, 12] 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 [13]. 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 of the two distinct data flows along different channels at the destination site.

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 : (i) a message sent before m along $c_{p,q}$, if delivered to q , is delivered before m ; (ii) 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 [14, 15] to increase concurrency on reliable channels compared to FIFO channels. We qualitatively compare the features of *slotted-FIFO* communication with F -channels [14, 15] and hierarchical channels [16, 17] in a later section.

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 messages m_1 and m_2 sent using $F\bar{R}$ -send primitives within S ensures 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 requirements [6] and to implement the transport layer of a group communication system to support multimedia streams of data [18].

Compared to the reliable FIFO channels, we get more concurrency and three basic advantages: (i) substantial reduction in the required buffer space at the receiver process, (ii) short message latencies, (iii) the ability to vary, at

run-time, the degree of ordering and/or reliability of the communication in each slot. Indeed, the destination process does not have to buffer the reliable messages that overtake the unreliable messages to ensure the delivery of the latter. Also, a message that must not be lost does not have to await the delivery of unreliable messages. The first two advantages are quantified by a simulation study, showing that the average buffer requirements and the message latency can be reduced up to one tenth. Point (iii) can allow an application to tune the reliability/ordering of the communication mode in order to meet its QoS; for example, varying the number of reliable messages sent in a slot or the ratio of reliable versus non-reliable messages in a slot.

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. Section 7 presents a qualitative comparison between slotted-FIFO communication mode, flush primitives [14] and hierarchical channels [17] and a general discussion of various issues and features of the proposed protocol. Finally, conclusions are presented in Section 8.

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 [6, 7]. We discuss the following examples.

2.1. 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 even though virtual channels may be exclusively assigned to the video and audio streams, demands are being placed on the underlying physical channel(s) 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 [9]. 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 FR 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 become significantly out of synch during a slot.

2.2. MPEG video transmission

MPEG-compressed video consists of three kinds of frames [13]. 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 the $\bar{F}\bar{R}$ primitive. As the decoding of predicted frames is dependent on the decoding of the preceding intraframe and predicted frames, employing the $\bar{F}\bar{R}$ 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 or semantics of the data stream.

2.3. Sliding window protocol

Enforcing FIFO order among the acknowledgements in a sliding window protocol is not necessary as well as expensive. An acknowledgement for a later packet implicitly acknowledges the reception of earlier packets at the receiver. Hence, to minimize the cost of acknowledgements they can be sent using the $\bar{F}\bar{R}$ primitive. If loss of acknowledgements is not acceptable, they can be sent using the $\bar{F}\bar{R}$ primitive. When acknowledgements for earlier packets are received by the sender after the acknowledgements for later packets, such acknowledgements are simply ignored. In doing so, we match the basic idea of a general-purpose sliding window protocol [4]. If acknowledgements are sent by using a modulo implementation of sequence numbers, we need to enforce some ordering among them in order to avoid the erroneous delivery of obsolete acknowledgements that could create some ambiguities. This can be corrected by requiring that every k th acknowledgement be sent as an FR message, where k is less than or equal to the size of the sliding window.

3. SYSTEM MODEL

A pair of processes p, q is connected by a communication network, or simply *network*. We assume that the network is connected, but unreliable and asynchronous. Each process

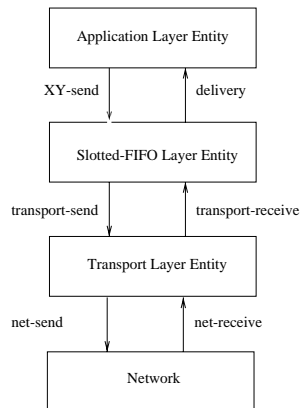


FIGURE 1. The structure of a process.

runs on a processor. The processors neither have a global clock, nor any shared memory.

We assume that each process consists of an application layer entity (AL), a slotted-FIFO layer entity (SL), and a transport layer entity (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 acknowledgement and retransmissions to ensure, if requested, reliable receipt of messages (the two reliable communication primitives invoke these mechanisms). TL generates a net-send event to the network and accepts a net-receive event from the network.¹

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 is sent. The messages in the sequence can be labeled as $m.0, m.1, \dots, m.k$, 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 and $\bar{F}\bar{R}$ -send. A channel can drop messages sent by unreliable primitives.

¹Note that the only assumption we make concerning TL is reliable delivering of messages upon request. So, from an implementation point of view, TL and SL could be merged in only one layer. In such a case slotted-FIFO could be used to implement a light sliding window protocol such as the one described in the previous section.

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 $XY \in \{FR, F\bar{R}, \bar{F}R, \bar{F}\bar{R}\}$). The messages in this subsequence can be represented as $XY.0, XY.1, \dots, XY.k_{XY}$, where $k_{XY} \leq k$ (where $m.k$ is the last message in the sequence M).

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 $f(FR.i) \leq m.x \leq f(FR.(i + 1))$.

DEFINITION 4.2. The projection of a slot $\Pi(S_i)$ is the set of messages, belonging to S_i , received by the SL of process q between the delivery of $FR.i$ and the delivery of $FR.(i + 1)$.

Then a pair of successive FR messages acts as delimiters of a slot and each FR message belongs to two slots, i.e. the ones for which it acts as a delimiter. Note that, by definition, $\Pi(S_i) \subseteq S_i$. 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 $m.x_h$ occurred before the delivery of $m.x_w$. Now, we are in a position to define the slotted-FIFO communication mode.

DEFINITION 4.1. A pair of processes p, q respects slotted-FIFO ordering on $c_{p,q}$ iff:

Safety:

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

Liveness:

Each message m in $\Pi(S_i)$ will eventually be delivered to process q .

The four rules of the safety part 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. The liveness part has been introduced to rule out trivial implementations of the slotted-FIFO paradigm which, for instance, drop all the (or a part of) unreliable messages

within SL. Examples of slotted-FIFO communications are depicted in Figure 2a.

5. A SIMPLE IMPLEMENTATION OF THE SLOTTED-FIFO LAYER

An implementation of the slotted-FIFO layer consists of defining a protocol between an XY -send procedure invoked by the application layer and a process (namely RECEIVE) which is instantiated each time a transport–receive event is generated by the transport level. In fact, 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. Next we will present the sender and receiver parts of the protocol, followed by an example demonstrating the protocol execution.²

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 non-reliable message in the current slot.

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

- *slot*: an integer indicating the slot associated with the message;
- *type*: a char indicating the type of the message ($FR, \bar{F}R, F\bar{R}$ or $\bar{F}\bar{R}$);
- *order*: an 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 next $F\bar{R}$ 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 non-reliable counter is reset (S4). When a non-FIFO reliable message has to be sent, the $S_{\bar{F}R}$ counter is incremented. When sending a non-reliable FIFO message,

²Readers can refer directly to the example in Section 5.3 before reading the protocol.

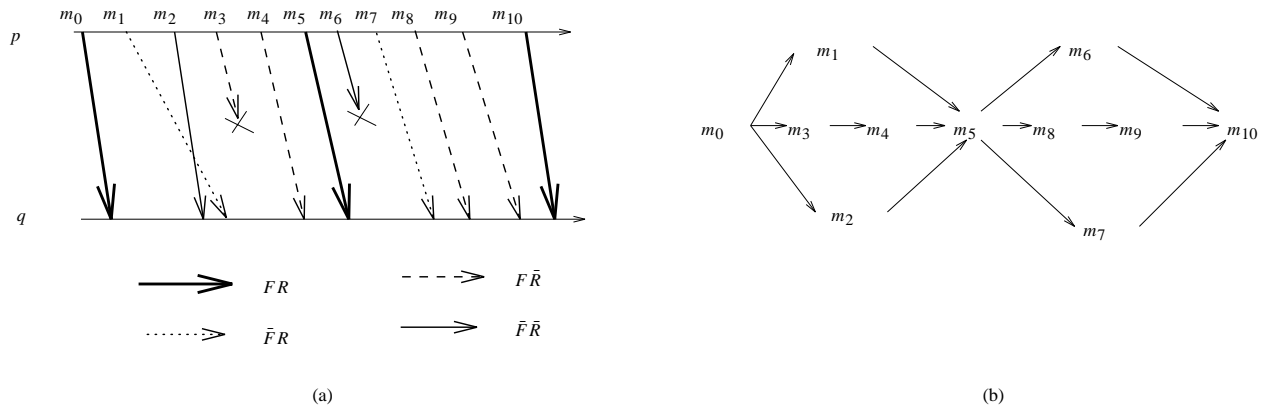


FIGURE 2. An example of slotted-FIFO communication and the corresponding partial order generated by the relation \leq .

```

init
slot=0; SF̄R=0; SF̄R̄=0;
procedure XY-SEND(m, t, j)
/*m: message content, t: the type */
/*j: destination process*/
begin
(S1) ST.slot = slot; ST.type = t;
    case t of
(S2) FR : ST.order = SF̄R; SF̄R = 0;
(S3)     slot = slot + 1;
(S4)     SF̄R̄ = 0; break;
(S5) F̄R : SF̄R̄ = SF̄R̄ + 1; break;
(S6) F̄R̄ : SF̄R̄ = SF̄R̄ + 1;
(S7)     ST.order = SF̄R̄; break;
(S8)     F̄R̄ : break;
    endcase
(S9) transport-send(m, ST);
end.
    
```

FIGURE 3. A simple XY-send primitive implementation.

the $S_{F̄R̄}$ counter is incremented and the order field of ST is set to the current value of the $S_{F̄R̄}$ counter.

It is to be noted that a process need not maintain sequence number information for non-FIFO non-reliable messages in the form of an $S_{F̄R̄}$ 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 $F̄R̄$ class of messages is the slot number.

The structure ST allows us to define a partial order on messages. 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 $m <_s m'$, iff $ST_m.slot < ST_{m'}.slot$. The precedence relation $<_s$ 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 $ST_m.type = ST_{m'}.type = F̄R$, we say that m precedes m' , denoted $m <_f m'$, iff $ST_m.order < ST_{m'}.order$. The precedence relation $<_f$ represents precedence within a slot due to FIFO ordering.
3. Let m and m' be two messages, we say that m precedes m' , denoted $m <_{es} m'$, iff $ST_m.slot = ST_{m'}.slot$ and $ST_m.type = FR$. The precedence relation $<_{es}$ denotes that messages in a slot precede the FR message that marks the end of the slot.

Now, we denote by \leq the transitive closure of the union of $<_s$, $<_f$ and $<_{es}$. The set of messages M can be then represented as a partial order of messages $\hat{M} = (M, \leq)$. Two messages m and m' are concurrent iff $\neg(m \leq m')$ and $\neg(m' \leq m)$. In Figure 2b the partial order $\hat{M} = (M, \leq)$ of the message scheduling of Figure 2a 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;
- $R_{F̄R}$: an integer that stores the number of the reliable non-FIFO message delivered in the current slot;
- $R_{F̄R̄}$: 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 process is shown in Figure 4. Each received message is associated with a WAIT condition which depends on the message type (R1, R7, R12, R16). Let the type of the message received be FIFO and reliable. In such a case there are two successive WAIT conditions (R1) and (R3). Statement (R1) permits further processing of the message 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 delivered in the current slot. This ensures that

```

init
slot=0;  $R_{\bar{F}\bar{R}}=0$ ;  $R_{F\bar{R}}=0$ ; end_slot=false;
when an  $(m, ST)$  event is received from the Transport-Layer
/*m: the message content, ST: the control information*/
begin
  case ST.type of
     $FR$ : WAIT (slot==ST.slot and ST.order== $R_{\bar{F}\bar{R}}$ ); (R1)
        end_slot =true; (R2)
        WAIT ( $\forall m' \in \Pi(ST.slot) : m' \preceq m :: m'$  has been delivered); (R3)
         $R_{\bar{F}\bar{R}}=0$ ;  $R_{F\bar{R}}=0$ ; slot=slot+1; (R4)
        deliver(m); (R5)
        end_slot=false; break; (R6)
     $\bar{F}R$ : WAIT(slot==ST.slot); (R7)
         $R_{\bar{F}\bar{R}}=R_{\bar{F}\bar{R}} + 1$ ; (R8)
        deliver(m); break; (R9)
     $\bar{F}\bar{R}$ : if (ST.slot < slot) (R10)
        then discard(m); break; (R11)
        else WAIT(ST.slot==slot); (R12)
            deliver(m); break; (R13)
     $F\bar{R}$ : if (ST.slot < slot) (R14)
        then discard(m); break; (R15)
        else WAIT((ST.slot==slot and (ST.order== $R_{F\bar{R}}+1$  or end_slot)) ); (R16)
             $R_{F\bar{R}}=ST.order$ ; (R17)
            WAIT ( $\forall m' \in \Pi(ST.slot) : m' \preceq m :: m'$  has been delivered); (R18)
            deliver(m); break; (R19)
  endcase
end .

```

FIGURE 4. A simple RECEIVE process implementation.

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. Then the *end_slot* variable is set to *true* indicating that the end of the slot is imminent as the *FR* message is being delivered (R4). The delivering of the message (R5) is permitted only when all previous messages belonging to $\Pi(ST.slot)$ have been delivered (R3). The presence of the second WAIT condition (R3) is due to the presence of *F \bar{R}* messages suspended on their WAIT condition (R16) as we will explain later. Finally, both $R_{\bar{F}\bar{R}}$ and $R_{F\bar{R}}$ counters are also reset (R4).

If a non-FIFO reliable message is received, the instance of the RECEIVE process waits until the *ST.slot* field equals the current slot (R7). The $R_{\bar{F}\bar{R}}$ field is incremented to reflect the delivery of a non-FIFO reliable message (R8), and the message is delivered (R9).

If a non-reliable non-FIFO message is received and the value of its *ST.slot* field is lower than the current slot, the message is discarded (R10, R11). 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 slot field is equal to the current slot (R12, R13).

Similarly, a non-reliable FIFO message is discarded if the value of its *ST.slot* field is less than the current slot. Otherwise its delivery is delayed until one of 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 buffered *F \bar{R}* messages (i.e. those whose processes are suspended on a delivery condition) must be immediately delivered. The order of delivery is imposed to be consistent with the relation \preceq (R18).

In other words, the WAIT conditions (R3) and (R18) are used to deliver messages consistently with the partial order given by the relation \preceq when delivery conditions of multiple suspended instances of the RECEIVE process become true simultaneously. This is the typical scenario that occurs when an *F \bar{R}* message gets lost in a slot. Then all successive *F \bar{R}* messages in the same slot will be suspended on their delivery conditions (R16). As soon as the *FR* message, delimiting the end of the slot, sets *end_slot* to true (R4), all delivery conditions of multiple suspended instances of the *F \bar{R}* messages then become true simultaneously. Hence they are activated in an order consistent with \preceq , i.e. let *m* and *m'* be two of these messages, *m* is activated before *m'* if $m \preceq m'$. If *m* and *m'* are concurrent they can be activated in any order.

5.2.1. A remark on protocol correctness

Let us briefly show how the proposed protocol ensures slotted-FIFO communication (Definition 4.3).

- *Safety.* Safety is ensured by the protocol as deliveries respect the partial order defined on messages by the relation \preceq and this order is consistent with the four rules in the safety part of Definition 4.3.
- *Liveness.* Liveness is ensured by the protocol as no message in slot S_i , received before delivering the message FR (R5) (that message defines the end of S_i), is discarded by the protocol. The only discarded messages (lines (R11) and (R15)) are those received by the protocol after the end of their slot (i.e. after the delivery of the FR message delimiting the beginning of the next slot). So, by Definition 4.2, they do not belong to $\Pi(S_i)$.

5.2.2. A remark on 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 process reaches a WAIT statement and the condition in the statement is not true, that particular instance of the RECEIVE process is suspended and gives up control of the receiver process;
- a suspended instance of the RECEIVE process 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 process at the same site;
- finally, to ensure consistency of updates to the condition variables the following statement sequences are executed in an atomic fashion: (R4)–(R6) and (R8)–(R9).

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

5.3. Examples

The behavior of the proposed slotted-FIFO layer can be illustrated by the following examples.

In the first example, shown in Figure 5, process p sends six messages to process q . Messages m_0 and m_5 are of type FR , m_1 , m_2 and m_3 are of type $\bar{F}\bar{R}$, while m_4 is of type $\bar{F}R$. The messages are received by process q in the order shown in the figure.

When m_2 and m_3 arrive at process q they cannot be immediately delivered as the values of their $ST.order$ fields are greater than the expected value. This happens because m_2 and m_3 have overtaken m_1 . So, m_2 and m_3 are

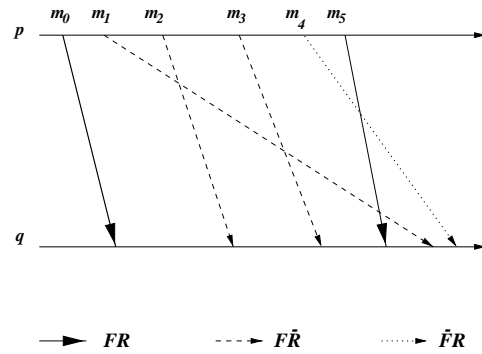


FIGURE 5. An example illustrating behavior of the proposed slotted-FIFO layer.

buffered and the corresponding processes are suspended on the condition in line (R16).

On the arrival of m_5 , it cannot be delivered immediately as its $ST.order > R_{\bar{F}R}$ due to the non-arrival of message m_4 , and the delivery condition in line (R1) is not satisfied. When m_1 arrives it is immediately delivered (as $ST.slot == slot$ and $ST.order == R_{\bar{F}R} + 1$ are satisfied) and the subsequent execution of line (R17) activates the processing of message m_2 that, in turn, will be delivered, activating finally the processing of message m_3 . When m_4 arrives it is delivered as its delivery condition is satisfied. On the delivery of m_4 the value of $R_{\bar{F}R}$ is incremented. As a result the delivery condition in line (R3) becomes true and message m_5 is delivered.

Let us now suppose that message m_1 would be received after the receipt of m_4 . As for the above case, when m_2 and m_3 arrive at process q they cannot be immediately delivered. However, the delivery of m_4 activates the process of message m_5 that sets end_slot to true (R2) and both m_2 and m_3 become eligible for delivery as the condition ($ST.slot == slot$ and end_slot) is satisfied. So, m_2 is delivered before m_3 as $m_2 \preceq m_3$. Finally, m_5 is delivered as the WAIT condition (R3) becomes true.

The second example, shown in Figure 6, consists of four messages sent by process p to process q . Message m_2 is lost in transit and never reaches process q . The remaining messages are received by q in the order shown.

Message m_4 that had overtaken m_1 is buffered on reception as its delivery condition is not satisfied (R1), and its corresponding receive process goes to sleep. When the receive process instantiated by the receipt of m_3 reaches the WAIT condition (R16), it is suspended as $ST.slot \neq slot$. Message m_1 is delivered to q as soon as it arrives. On the delivery of m_1 , message m_4 becomes eligible for delivery because its $ST.slot$ value is now equal to the value of the local $slot$ variable and there are no pending $\bar{F}R$ messages. So end_slot is set to true and this activates the process of message m_3 . Then messages m_3 and m_4 will be delivered according to the relation \preceq , i.e. m_3 is delivered first.

It is to be noted that process q did not have to wait for the arrival of m_2 before making a decision about delivering

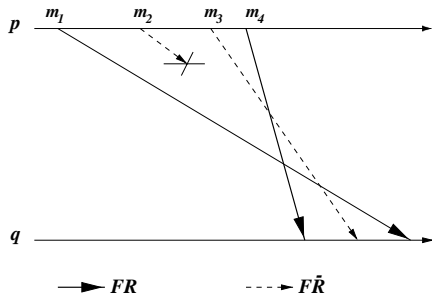


FIGURE 6. Behavior of the proposed slotted-FIFO layer on message loss.

m_3 even though both the messages are sent using the $F\bar{R}$ primitive. The logic behind such an implementation is that, once an FR message marking the end of a slot is received, its delivery should not be delayed on account of non-reliable FIFO messages sent earlier in the slot as loss of these non-reliable messages does not adversely affect the semantics of the slotted-FIFO communication mode (Section 4). An $F\bar{R}$ message which is not received within its slot is semantically similar to a lost $F\bar{R}$ message. As $F\bar{R}$ messages have no ordering constraints they can be delivered whenever they arrive as long as they arrive within the slot.

6. SIMULATION STUDY

In this section we report performance results of the slotted-FIFO channel. We assume FR -send primitives are generated by AL (application layer entity) at fixed time intervals of length T_s , referred to as the slot length. In the interval between two successive FR -send messages, primitives of the other types are generated according to a Poisson process with rate p (referred to as the load), and are labeled as XY type with probability P_{XY} . The SL stores waiting messages inside a *resequential buffer*. The network is characterized by *reliability*, which is the probability, P_{succ} , that a sent message is not lost. No correlation is assumed between losses, and the size of the message transmitted by each primitive does not require fragmentation.

In order to guarantee reliable transmissions 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 the TL layer to avoid message duplication. We assume no flow control and infinite buffer size at TL.

We define the network delay, T_{ntw} , as the time elapsed between 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 and 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 . Therefore, T_d is a measure of the duration for which messages may have to be buffered at the receiver between

their receipt by the slotted-FIFO layer and their delivery to AL.

For messages that are not lost, T_{ntw} is a normal distributed random variable with mean μ_{ntw} and variance σ^2 , truncated at values $x_1 = 0.1\mu_{ntw}$ and $x_2 = 1.9\mu_{ntw}$. We assume $T_{tra} = T_{ntw}$ for unreliable messages that are not lost³, and $T_{tra} = \alpha T_{to} + T_{ntw}$ for reliable message, where α is a geometrically distributed random variable denoting the number of retransmissions:

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

6.1. Results of the experiments

We study the case of the sender and the receiver attached to two distinct LANs connected by a backbone. In this scenario we can assume that the transmission rate is limited by the backbone throughput. Thus, by assuming a transmission speed of 10 Mbps on the backbone and an average packet size of 1 kbytes the value for p (load) is of the order of 100 messages/s. The values for μ and σ used in the simulation experiments were estimated by using a collection of about 10,000 round-trip delays values between two hosts on distinct LANs computed by the *ping* UNIX command. We observed a wide variability in the network delay, while the average value is about 200 ms.

Results presented in this section, unless stated otherwise, were obtained with the following parameters: $T_{to} = 2T_{ntw}$, $p = 150$ messages/s, $\mu_{ntw} = 200$ ms, $\sigma = 100$ ms, $P_{succ} = 0.999$, $T_s = 120$ ms. In order to study separately the effect of reliability and ordering requirements, we considered cases where the messages inside the slots are of the same type. Particularly, in scheme (a) $P_{FR} = 1$,⁴ in scheme (b) $P_{F\bar{R}} = 1$, in scheme (c) $P_{\bar{F}R} = 1$, and in scheme (d) $P_{F\bar{R}} = 1$. We also considered a mixed case (scheme (e)) where $P_{F\bar{R}} = P_{\bar{F}R} = 1/3$.

Performance results show the average delivery delay (T_d) experienced by messages of the same type and the average queue length (N) of the resequencing buffer embedded in the slotted-FIFO layer as a function of the main system parameters.

For every estimated point we performed several independent simulation experiments. The 95% confidence interval was computed using the t-student distribution. Confidence intervals are not reported in the figures since they are less than 4% of the sample mean.

In the following we first give an idea of how many messages are received out of order in our model by using the scheme (a). Then, for each scheme, we measure the delivery delay of the messages demarcating the end of a slot, one of the messages inside a slot and buffer requirements. Additional experiments have been carried out on two particularly interesting slotted-FIFO schemes which seem to offer best performance and reliability/ordering flexibility.

³We assume that no delay is involved in the TL when sending or receiving a message.

⁴To compare the slotted-FIFO performance with pure FIFO channels, we allow messages within a slot to be of the FR type.

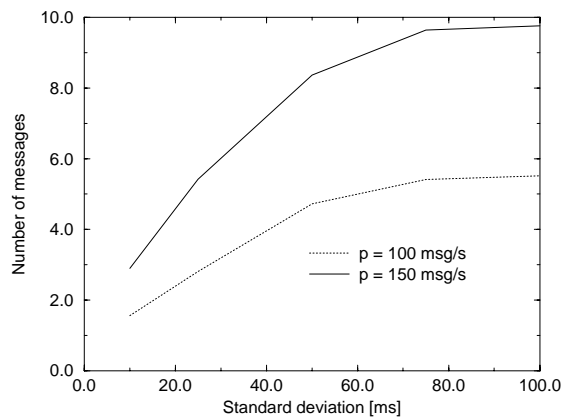


FIGURE 7. Average number of messages that a message must wait before its delivery.

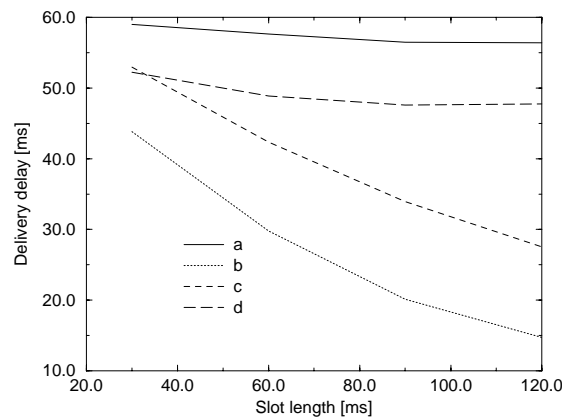


FIGURE 9. Delivery delay of messages inside a slot.

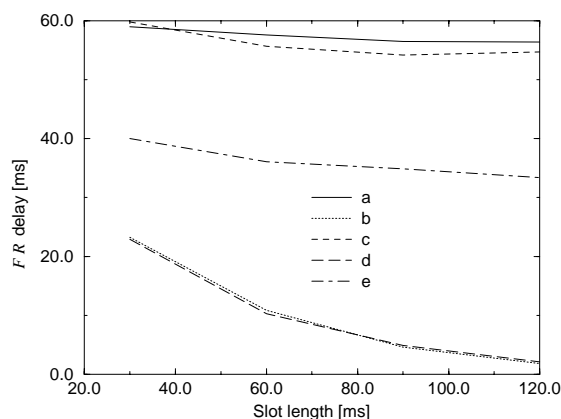


FIGURE 8. FR delivery delay versus slot length.

6.1.1. Number of messages that arrive out of order

A message m waiting in the resequential buffer indicates that m has overtaken some messages that have been sent before m . As soon as all such messages sent prior to m have been received, the message m is delivered.

Let us denote with k the average number of messages that m must wait for before its delivery, $k = pT_d$. Figure 7 shows k as a function of the standard deviation—expressed as a percentage of the average network delay—for $p = 100$ and $p = 150$ messages/s. As expected, the larger the standard deviation, the greater the number of messages for which m has to wait before its delivery. For high values of σ , k tends to a constant value as the probability density function of T_{ntw} approaches a truncated uniform distribution whose variance is constant.

6.1.2. Delivery delays

Figure 8 shows the delivery delay of FR messages as a function of the slot length (measured in ms). FR delivery delay denotes the time required for slot reorder and is affected only by reliable message arrival out of sequence.

An FR message delivery can in fact take place only after all the reliable messages inside its slot have been received. In schemes (b) and (d), we observed a significant reduction of the FR delivery delay. In such schemes 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. In schemes (a) and (c), one would expect a flat behaviour independent of the slot length. However, we note a drop of 10% as the slot length increases. This drop is due to the fact that the wider the slot is, the lower the number of out of sequence message will be. This reduces, in turn, the FR delivery delay. Scheme (e) is an example of FR delivery delay in which 33% of messages in a slot are sent in a reliable way.

Figure 9 shows the average delivery delay of the messages inside a slot. It can be seen as the sum of two delays associated with the wait condition of a message. The first delay, D_1 , is associated with the situation where a message m in a slot s reaches the destination prior to the delivery of the FR message marking the beginning of s . D_1 is higher for smaller slot intervals since small slot intervals increase the probability that messages arrive out of sequence. Once m is in its correct slot (which could possibly be the moment the message is received, i.e. D_1 equals zero), D_2 is the delay waiting for the other delivery conditions to be satisfied. In the non-FIFO schemes (b) and (c), $D_2 = 0$ and thus delivery delay decreases as the slot length increases. In the scheme (d), $D_2 > 0$ since some FR messages can be forced to wait until the end of the slot before they are delivered. As the slot length increases, D_2 dominates over D_1 .

6.1.3. Buffer requirements

The average queue length of the resequencing buffer, N , was used to compare the storage requirements of various schemes. Results are shown in Figure 10. These measures can be useful to design memory space allocation used for message reordering at the slotted-FIFO layer according to application requirements. We would like to remark that using the Little results the average queue length can also be

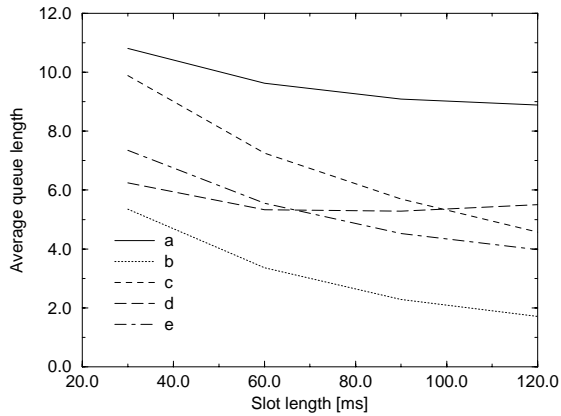


FIGURE 10. Average queue versus slot length.

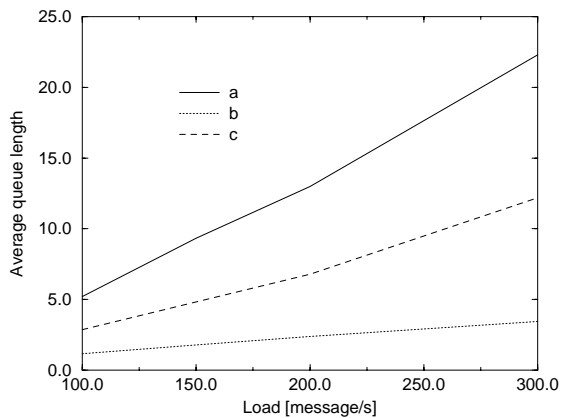


FIGURE 11. Average queue versus load.

useful to quantify the overall average message delivery delay (W). For example, in scheme (a) $W = (p + 1/T_s)^{-1}N$ and this corresponds to the plot of the average delivery delay reported in Figure 8. When $T_s = 120$ ms and a non-FIFO scheme ((b) or (c)) is used the average queue length is between two and five times less than the reliable FIFO scheme (a). This indicates that the buffer space and message latency can be reduced by using weak delivery ordering constraints.

Previous results show that, if an application can work with relaxed ordering and/or reliability constraints, slotted-FIFO schemes (b) and (c) are the best candidates to replace pure FIFO communication. Hence, a detailed performance study was carried out for such schemes and results are shown in the following section.

6.1.4. FIFO versus non-FIFO communications

Figure 11 shows the average queue length as a function of the load. Consider the reliable scheme (c) with load $p = 300$ messages/s. Using the Little result, W is about 35 ms while for pure FIFO communication (scheme (a)) W is about 70 ms. This denotes how the delivery order can

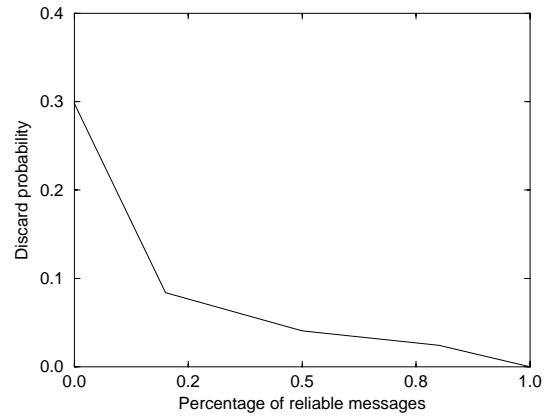


FIGURE 12. Discard probability versus $P_{\bar{F}R}$.

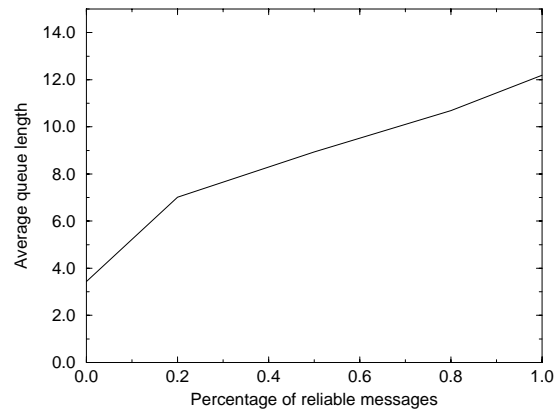


FIGURE 13. Average queue length versus $P_{\bar{F}R}$.

affect buffer space requirements and message latency.

If the transmitted messages can be discarded then the buffer space can be further reduced, as in scheme (c). The percentage of discarded messages and the average queue length can be controlled by using both reliable and unreliable primitives when considering $p = 300$ messages/s and $T_s = 120$ ms. Figure 12 shows the probability that a message is discarded by TL as a function of the percentage of messages sent using an $\bar{F}R$ primitive, $P_{\bar{F}R}$. Note that only $1 - P_{\bar{F}R}$ of the generated messages are transmitted in an unreliable way. Figure 13 shows the average queue length as a function of $P_{\bar{F}R}$.

The above results show that the slotted-FIFO communication mode offers great flexibility. An application can decide dynamically which types (and in what percentages) of messages are more convenient to transmit inside a slot on the basis of the desired QoS (acceptable delivery delays and/or discard probability) and available buffer space.

7. DISCUSSION

7.1. Comparison with flush and hierarchical channels

A novel feature of the slotted-FIFO communication mode is that it combines message ordering and reliability specifications. Therefore, slotted-FIFO mode supports powerful primitives to express the desired communication characteristics for a set of processes communicating asynchronously.

With respect to primitives for expressing message ordering, slotted-FIFO has been preceded by flush primitives [14] and hierarchical channels [17]. Four different flush primitives have been proposed in [14]: a two-way flush message (which does not overtake other messages and cannot be overtaken by other messages), a forward-flush message (which cannot overtake other messages), backward flush messages (which cannot be overtaken by other messages) and ordinary messages (with no ordering constraints). In hierarchical channels [17], each message has a level associated with it. A lower level message can overtake a higher level message, but a higher level message cannot overtake a lower level message. In slotted-FIFO communication a message m is said to overtake another message m' if both have a common sender, a common destination, m' is sent before m and m is delivered to the destination before m' .

Slotted-FIFO message primitives can mimic the capabilities of flush channels to a significant extent. For example, a message sent using the FR -send primitive, marking the end of one slot and the beginning of the next slot, is similar to a two-way flush message. This is because message delivery cannot spill into adjacent slots. A message sent using the $F\bar{R}$ -send primitive cannot overtake other FIFO messages (reliable and delivered unreliable ones). However, it can be overtaken by non-FIFO messages (both reliable and non-reliable). So, messages sent using the $F\bar{R}$ -send primitive are analogous to forward-flush messages. A non-FIFO reliable message cannot be overtaken by FIFO reliable messages. However, it can be overtaken by non-FIFO messages (both reliable and unreliable) and FIFO non-reliable messages. So, non-FIFO reliable messages are similar to backward-flush messages. Finally, non-FIFO non-reliable messages impose no constraints and are similar to ordinary messages.

With regard to message ordering, both flush and slotted-FIFO communication modes are weaker than hierarchical channels.

Both flush primitives and hierarchical channels insist on reliable message communication even if the application using these primitives does not require such reliability. Loss of messages will violate the semantics of these primitives. To avoid message loss, expensive buffering and acknowledgement protocols will need to be implemented at lower layers of the network. In contrast, slotted-FIFO communication mode provides more latitude as far as message loss is concerned. If the application does not mind some messages being lost, such messages can be sent using the $F\bar{R}$ -send and $\bar{F}\bar{R}$ -send primitives. Acknowledgements, transmission retries and buffering overheads will not be

incurred if such messages are lost. If the application insists on reliable delivery of messages all communication will be restricted to using only the FR -send and $\bar{F}\bar{R}$ -send primitives.

7.2. Further remarks

The possibility of changing the degree of reliability and ordering of the slotted-FIFO communication at run-time can be used by applications to meet the required QoS by, for example, varying the number of reliable messages sent in a slot, or the ratio of reliable versus non-reliable messages in a slot.

Moreover, the slotted-FIFO communication mode can be easily extended to handle applications whose messages have a lifetime (i.e. limited time validity) after which their data can no longer be used by the application. This could be done by using the technique proposed by Baldoni *et al.* in [11] in the specific context of causal ordering protocols. For example, a lifetime can be associated with each slot (i.e. with each FR message) if the slot does not start within its lifetime, all messages in that slot will be discarded. The computation of the lifetime implies the existence of a virtual global clock, formed by a set of local clocks, available at each process [19].

Compared to communication modes with no deadlines, slotted-FIFO may seem to reduce the probability of $F\bar{R}$ and $\bar{F}\bar{R}$ messages being delivered. This is especially so for the messages of these types that are sent towards the end of a slot. However, due to the reduction of ordering constraints, such messages also experience a reduction in the time they are buffered at the destination prior to their delivery. Thus, there is a tradeoff between chances of delivery of some messages marked as unreliable by the sender, on the one hand, and the overall communication latency and buffer requirements on the other.

8. CONCLUSIONS

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 provides communication with a run-time variable degree of reliability and ordering while maintaining a certain degree of ordering among them at slot level granularity. 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 out-of-slot arrival, and requires no special handling. Simulation results showed that weak constraints on ordering and/or reliability lead to large savings in message buffering and low message latencies compared to pure FIFO channels. In particular, we also showed how the number of reliable messages sent in a slot affects the performance of the communication mode.

We showed that designing a slotted-FIFO layer is simple and this should imply a simplicity of implementation. The bookkeeping overheads are also low. Hence, slotted-FIFO seems to be a nice compromise between a reliable and FIFO communication (TCP-like) and the unreliable and unordered communication (UDP-like). We are currently developing an implementation of the slotted FIFO protocol over IP.

ACKNOWLEDGEMENT

The authors would like to thank Roy Friedman (Technion University) for useful comments and discussions on an earlier version of the paper. Comments of the anonymous referees gave an invaluable help to improve the content and the presentation of the paper.

REFERENCES

- [1] Baldoni, R. (1998) A positive acknowledgement protocol for causal broadcasting. *IEEE Trans. Comput.*, to appear.
- [2] Birman, K. and Joseph, T. (1987) Reliable communication in presence of failures. *ACM Trans. Comput. Syst.*, **5**, 47–76.
- [3] Birman, K., Schiper, A. and Stephenson, P. (1991) Lightweight causal and atomic broadcast. *ACM Trans. Comput. Syst.*, **9**, 272–314.
- [4] Tanenbaum, A. S. (1996) *Computer Network*. Prentice-Hall International.
- [5] Baldoni, R., Friedman, R. and van Renesse, R. (1997) The hierarchical daisy architecture for causal delivery. In *Proc. 17th IEEE Int. Conf. on Distributed Computing Systems*, Baltimore, MD, pp. 570–577. IEEE Computer Society Press.
- [6] Gong, F. and Parulkar, G. M. (1996) An application-oriented error control scheme for high speed networks. *IEEE/ACM Trans. Networking*, **4**, 669–683.
- [7] Yavatkar, R. (1992) MCP: A protocol for coordination and temporal synchronization in multimedia collaborative applications. In *Proc. 12th IEEE Int. Conf. Distributed Computing Systems*, Yokohama, Japan, pp. 606–613. IEEE Press.
- [8] Houdoin, T. and Bonjour, D. (1994) ATM and AAL layer issues concerning multimedia applications. *Ann. Telecommun.*, **49**, 230–240.
- [9] Wakeman, I. (1993) Packetized video: options for interaction between the user, the network and the codec. *Comp. J.*, **36**, 55–66.
- [10] Adelstein, F. and Singhal, M. (1995) Real-time causal message ordering in multimedia systems. In *Proc. 15th Int. Conf. on Distributed Computing Systems*, Vancouver, Canada, pp. 36–43, IEEE Computer Society Press.
- [11] Baldoni, R., Mostefaoui, A. and Raynal, M. (1995) Causal delivery of messages with real-time data in unreliable networks. *J. Real-time Syst.*, **10**, 245–262.
- [12] Baldoni, R., Prakash, R., Raynal, M. and Singhal, M. (1996) Efficient Δ -causal broadcasting. In *Proc. 23rd EUROMICRO Conf.*, Vienna, Austria, pp. 617–626. IEEE Computer Society Press.
- [13] Gall, D. (1991) MPEG: a video compression standard for multimedia applications. *Commun. ACM*, **34**, 46–58.
- [14] Ahuja, M. (1990) Flush primitives for asynchronous distributed systems. *Inf. Proc. Lett.*, **34**, 5–12.
- [15] Ahuja, M. (1993) An implementation of F-channels. *IEEE Trans. Parallel Distribut. Syst.*, **4**, 658–667.
- [16] Ahuja, M. (1991) *Hierarchy of Communication Speeds for Designing Concurrent Systems*. Technical Report OSU-CISRC-1/91-TR1, The Ohio State University.
- [17] Ahuja, M. and Prakash, R. (1992) On the relative speed of messages and hierarchical channels. In *Proc. 4th IEEE Symp. Parallel and Distributed Processing*, Arlington, TX, pp. 246–253. IEEE Computer Society Press.
- [18] Clocker, G. V., Huleihel, N., Keidar, I. and Dolev, D. (1996) Multimedia multicast transport service for groupware. In *Proc. TINA'96 Conf.*, Heidelberg, Germany, pp. 43–54, VDE Verlag.
- [19] Verissimo, P. (1994) Ordering and timeliness requirements of dependable real-time programs. *J. Real-time Syst.*, **7**, 105–128.