September 1990

A Performance Analysis of Timed Synchronous Communication Primitives

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Susan B. Davidson
University of Pennsylvania, susan@cis.upenn.edu

A Performance Analysis of Timed Synchronous Communication Primitives

Abstract

Two algorithms for timed synchronous communication between a single sender and single receiver have recently appeared in the literature. Each weakens the definition of "correct" timed synchronous communication in a different way, and exhibits a different "undesirable" behavior. In this paper, their performance is analyzed, and their sensitivity to various parameters is discussed. These parameters include how long the processes are willing to wait for communication to be successful, how well synchronized the processes are, the assumed upper bound on message delay, and the actual end-to-end message delay distribution. We conclude by discussing the fault tolerance of the algorithms, and propose a mixed strategy that avoids some of the performance problems.

Keywords

ada rendezvous, distributed systems, performance analysis, predictability, real-time systems, synchronous communication

A Performance Analysis of Timed Synchronous Communication Primitives

INSUP LEE, MEMBER, IEEE, AND SUSAN B. DAVIDSON, MEMBER, IEEE

Abstract—**Two algorithms for timed synchronous communication between a single sender and single receiver have recently appeared in the literature. Each weakens the definition of "correct" timed synchronous communication in a different way, and exhibits a different "undesirable" behavior. In this paper, their performance is analyzed, and their sensitivity to various parameters is discussed. These parameters include how long the processes are willing to wait for communication to be successful, how well synchronized the processes are, the assumed upper bound on message delay, and the actual end-to-end message delay distribution. We conclude by discussing the fault tolerance of the algorithms, and propose a mixed strategy that avoids some of the performance problems.**

Index Terms—**Ada rendezvous, distributed systems, performance analysis, predictability, real-time systems, synchronous communication.**

I. Introduction

In DISTRIBUTED *real-time* systems, arbitrary delays associated with synchronous communication cannot be tolerated due to the time-dependent nature of computation. Languages such as Ada[1] [1] and Occam[2] [2], which are designed for real-time programming, therefore, support the notion of a *deadline* with their synchronous communication constructs. Real-time processes written in these languages can attach deadlines to their communication requests specifying how long they are willing to wait for successful communication. This gives rise to a notion of *timed synchronous communication,* which can be defined as follows. At time RS, the sender becomes ready to communicate, and sends a message containing information and its deadline, DS, which is the latest time the sender is willing to wait for successful communication. At some other point in time, unrelated to RS, the receiver becomes ready to receive a message, RR. The receiver also has its own deadline, DR. At DS and DR, the sender and receiver, respectively, will continue processing and must *know* the success or failure of communication. The problem is to design a nontrivial protocol (i.e., one which allows the possibility of success) which guarantees that the sender and receiver will meet their deadlines and agree on whether or not communication is successful.

Manuscript received December 1, 1987; revised April 26, 1988. This work was supported in part by NSF DCR 8501482, NSF DMC 8512838, NSF MCS 8219196-CER, ARO DAA6-29-84-k-0061, and a grant from AT&T's Telecommunications Program at the University of Pennsylvania.

The authors are with the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104.

IEEE Log Number 9037190.

[1]Ada is a trademark of the U.S. Department of Defense (Ada Joint Program Office).

[2]Occam is a trademark of the INMOS Group of Companies.

The correct implementation of timed synchronous communication primitives is complicated by the fact that the end-to-end delay of messages can be arbitrary or even infinite, i.e., messages can get lost. In fact, if the assumed failure environment includes message loss, it can be shown that such a protocol does not exist. This is the "Two Generals" problem, in which two generals are trying to agree upon a common time of attack but can only communicate via unreliable messengers. Knowing that they will be defeated unless the attack is simultaneous, neither general will attack unless he is certain that agreement has been reached. However, agreement between the two generals is impossible as long as there is more than one value which can be decided upon. To show this, suppose that there exists a minimal sequence of messages between general A and general B for reaching agreement. Let M_k to be the last message in this sequence, and let it be sent from general A to general B. Since general A cannot be sure that general B received M_k, general A will attack regardless of the outcome of the message delivery. Thus, M_k could be removed from the sequence of messages, contradicting the claim that the sequence is minimal. Such an argument could be extended to show that if a minimal sequence of messages did exist, it would be an empty sequence thus allowing agreement in the case where only one value can be decided upon [3].

Since our intention is to develop algorithms for timed-synchronous communication that work in reasonable operating environments, an upper bound on message delivery cannot be assumed. Two "relaxations" of the problem statement can therefore be offered, and have recently appeared in the literature. In the first, the sender's deadline is "absolute," i.e., the sender decides that the message is unsuccessful if it does not hear to the contrary by its deadline [4]. Since an acknowledgment message from the receiver as to the success or failure of the communication could take longer than expected, there is a possibility for *inconsistent decisions*. This means that the sender believes that communication has failed, whereas the receiver knows that communication has been successful. In the second relaxed problem statement, the deadline of the sender is *not* absolute, i.e., the sender waits until it receives the acknowledgment [5]. Although inconsistent decisions are not possible, the sender can be delayed past its deadline. Thus, although neither approach fully implements timed synchronous communication, each approximates it to some extent; furthermore, each exhibits a different "undesirable" behavior.

The goal of this paper is to show under which operating conditions protocols for these relaxed problem statements perform well, and to measure how "badly" each can perform. This is done by probabilistic analysis, which gives insight into what values should be chosen for parameters in the protocols to yield "acceptable" performance. However, the choice between the potential for inconsistent decisions and missed deadlines is controversial. Some feel that inconsistent decisions should never be allowed in real-time applications since conflicting actions may be taken. Others feel that deadlines should never be missed; in hard real-time applications, a missed deadline usually triggers an emergency shut-down of the system. One missed deadline can also cause other deadlines to be missed, creating a cascading effect. A potentially more severe problem with these algorithms is that if messages can be lost, then two communicating processes can reach an inconsistent decision which can never be detected, or the sender may wait forever. To remedy these problems, we extend the notion of consistency to a three value semantics: *successful, unsuccessful* or *don't know*. We then describe an algorithm in which the sender and receiver always reach a decision by their deadlines, and the sender either definitely knows the decision, or knows that it is not sure as to the decision. The algorithm is attractive since if the sender's deadline can be extended, the answer is ensured to monotonically improve with time [6], [7].

The rest of this paper is organized as follows. In the next section, we present protocols for timed-synchronous communication with inconsistent decisions and for timed-synchronous communication with missed deadlines, and discuss performance measures. Section III outlines the probability model. In Section IV, we discuss the sensitivity of the protocols to such parameters as the assumed upper bound on message delivery, how long the processes are willing to wait for communication to be successful, how well synchronized the processes are, and the end-to-end message delay distribution. Section V discusses the fault tolerance of the protocols and extends the probability model to include the effect of message loss. After describing ways to cope with message loss, we present a better algorithm that combines the first two algorithms. We conclude by summarizing general results, discussing the usefulness of the model, and proposing future research.

II. Overview of Protocols and Performance Measures

In order to simplify the algorithms and analysis, we make the following assumptions.

Assumption 1: Clocks are perfect. That is, when the sender's clock reads time X, the receiver's clock also reads time X.

Relaxing this assumption is not difficult, and is discussed in [4]. We also restrict ourselves initially to a failure model which includes only arbitrary message delays.

Assumption 2: Processes are perfect.

Assumption 3: Messages are eventually delivered, and are delivered correctly.

Removing this assumption, even to allow transient message loss, results in very undesirable behaviors: undetected inconsistent decisions in the first algorithm, and the sender waiting forever in the second. A more complete discussion of this problem, along with an extended analysis and a compromise solution, will be presented in Section V.

```
/* When Sender P becomes ready to communicate, */
/* P executes as follows: */
    send(message,DS)
    within DS do
        when (receive acknowlegement from Q) do
            return(success)
        end when
    exception
        return(failed)
    end within

/* If a message arrives while Receiver Q is not ready to communicate, */
/* Q executes as follows: */
    within LT do
        when (Q becomes ready) do
            send("Yes");
            return(message,success)
        end when
    end within

/* When Receiver Q becomes ready to communicate, */
/* Q executes as follows: */
    within DR do
        when (receive message from P) do
            if (LT is passed)
                then return(failed)
                else send("Yes");
                    return(message,success)
        end when
    exception
        return(failed)
    end within
```

Fig. 1. Outline of Algorithm 1.

When presenting the algorithms we need to express timing constraints within a program, and will therefore use the notion of temporal scopes [8]. The syntax of a temporal scope is

within D **do**
 when ⟨condition⟩ **do**
 ⟨statement list⟩
 end when
exception
 ⟨exception handling statements⟩
end within.

The semantics of temporal scope is that a process can, after entering a scope, wait for ⟨condition⟩ until its deadline D is reached. If the current time reaches deadline D while the process is waiting for message arrival and process becoming ready to communicate, the deadline exception is raised and handled within the exception handler part of the temporal scope.

A. Algorithms for the Relaxed Problem Statement

Fig. 1 outlines Algorithm 1 for timed-synchronous communication with inconsistent decisions. When the sender becomes ready to communicate, it sends the message together with its deadline DS to the receiver. If the sender does not receive an acknowledgment by its deadline, it decides that the communication has failed.

When the receiver becomes ready to communicate, it waits for a message until its deadline, at which point it decides the communication has failed. When a message arrives, the receiver calculates the "latest time to respond" to the sender, LT=DS$-d$, where d is chosen so that the sender will, with some high probability, receive the acknowledgment by its deadline. Thus, d could be the mean end-to-end message delay, or some larger value. If the message arrives before the

```
/* When Sender P becomes ready to communicate, */
/* P executes as follows: */
    send(message,DS)
    when receive acknowlegement from Q do
        if "Yes"
            then return(message,success)
            else return(failed)
    end when

/* If a message arrives while Receiver Q is not ready to communicate, */
/* Q executes as follows: */
    within LT do
        when (Q becomes ready) do
            send("Yes");
            return(message,success)
        end when
    exception
        send("No");
    end within

/* When Receiver Q becomes ready to communicate, */
/* Q executes as follows: */
    within DR do
        when (receive message from P) do
            send("Yes");
            return(message,success)
        end when
    exception
        return(failed)
    end within
```

Fig. 2. Outline of Algorithm 2.

receiver is ready to communicate and LT does not expire before the receiver becomes ready, or if the receiver receives a message after it becomes ready and the current time is earlier than LT, then it accepts and acknowledges the message. Otherwise, it decides that communication has failed.

As mentioned in the beginning of this section, no matter what value of d is chosen there is a chance that inconsistent decisions will be made. If a "Yes" response arrives after DS, the sender must take some appropriate recovery actions. Making d very large reduces the probability of this occurring; however, it also reduces the probability that the receiver will ever say "Yes."

Fig. 2 outlines Algorithm 2 for timed-synchronous communication with missed deadlines. The sender sends a message and then waits for an acknowledgment. Unlike Algorithm 1, the sender receives an explicit "Yes/No" answer which may arrive after its deadline.

If a message arrives at the receiver before it is ready to communicate, it waits until LT and then answers "No." However, once it has become ready to communicate, the receiver answers "Yes" *even if LT has already expired* since the sender is guaranteed to wait for the response regardless of the outcome.

To contrast the two algorithms, Table I shows how and when the receiver decides. Letting M represent the time at which the initial message arrives at the receiver, the events of interest are M, LT, RR, and DR. There are 12 possible orderings of these events since RR always happens before DR. Whenever two events happen simultaneously, we interpret them as if one event happened before another so as to favor the success of communication. For example, if M and LT occur simultaneously, we assume $M <$ LT. Our treatment of simultaneous events also makes the 12 orderings mutually exclusive.

```
/* When Sender P becomes ready to communicate, */
/* P executes as follows: */
    send(message,DS)
    within DS do
        when (receive acknowlegement from Q) do
            if "Yes"
                then return(message,success)
                else return(failed)
        end when
    exception
        return(don't know)
    end within

/* If a message arrives while Receiver Q is not ready to communicate, */
/* Q executes as follows: */
    within LT do
        when (Q becomes ready) do
            send("Yes");
            return(message,success)
        end when
    exception
        send("No");
    end within

/* When Receiver Q becomes ready to communicate, */
/* Q executes as follows: */
    within DR do
        when (receive message from P) do
            send("Yes");
            return(message,success)
        end when
    exception
        return(failed)
    end within
```

Fig. 3. Outline of Algorithm 3.

In the remainder of this paper, ordering i is referred to as Ord(i), for $i = 1, \cdots, 12$.

B. Performance Measures

Although each algorithm can easily be shown to be a correct solution to their respective relaxed problem statement, they only *approximate* the correct statement since the sender may make an inconsistent decision or miss its deadline. Our goal is to measure how well these algorithms approximate the correct statement. We therefore wish to measure the occurrence of the following events:

- *Success*: The receiver says "Yes" and the sender knows by its deadline.
- *Inconsistent*: Inconsistent decisions are made.
- *Late-Yes*: The receiver says "Yes" and the sender is delayed past its deadline.
- *Late-No*: The receiver says "No" and the sender is delayed past its deadline.

Recall that late decisions cannot be made in Algorithm 1, although the decision made at the sender's deadline may be inconsistent. Furthermore, inconsistent decisions cannot be made in Algorithm 2, although the sender may be delayed past its deadline waiting for either a "Yes" or "No" response. In the next section, we therefore develop a model to measure the probability of success for both algorithms ($Success_1$ and $Success_2$), the probability of inconsistent decisions for Algorithm 1 (*Inconsistent*), and the probability of a late response for Algorithm 2 (*Late-Yes*, in which the response is "Yes," and *Late-No,* in which the response is "No").

TABLE I
SPECIFICATION OF ALGORITHMS

Ordering	Event Sequence	Algorithm 1	Algorithm 2
1	$RR \leq M \leq LT \leq DR$	Yes	Yes
2	$RR \leq M \leq DR < LT$	Yes	Yes
3	$M < RR \leq LT \leq DR$	W-RR-Yes	W-RR-Yes
4	$M < RR < DR < LT$	W-RR-Yes	W-RR-Yes
5	$RR \leq LT < M \leq DR$	No	Yes
6	$LT < RR \leq M \leq DR$	No	Yes
7	$LT < M < RR < DR$	No	No
8	$LT < RR < DR < M$	No	No
9	$RR \leq LT \leq DR < M$	No	No
10	$RR < DR < LT < M$	No	No
11	$RR < DR < M \leq LT$	W-LT-No	W-LT-No
12	$M \leq LT < RR < DR$	W-LT-No	W-LT-No

All responses are assumed immediate, except "W-X" which signifies "wait until event X occurs." Note also that in Ordering 11, the receiver does not remember that its deadline has already expired; that is, rather than responding "No" immediately, it delays until LT and then answers "No".

III. THE PROBABILITY MODEL

To evaluate the performance measures enumerated in the previous section, we use the following random variables:

- X_m = delay of initial message.
- X_a = delay of acknowledgment message.
- RS = time at which sender is ready.
- RR = time at which receiver is ready.

We also use the following parameters, which will be altered to see the effect on the performance of the algorithms:

- Δ_s = length of time sender will wait.
- Δ_r = length of time receiver will wait.
- d = time receiver allows for acknowledgment to be delivered.
- T = relative synchronization of the sender and receiver.

By "relative synchronization" we mean that the sender and receiver will each become ready to communicate exactly once during the time period $[1, T]$. The smaller T is, the better the synchronization; if the sender and receiver are very poorly synchronized, T is arbitrarily large. Note also that we do not consider *multiple* receiver ready intervals during $[1, T]$. That is, if the message arrives after the receiver's ready interval, both algorithms wait until LT and then decide that communication fails. If multiple receiver intervals were allowed, the receiver could become ready to communicate again before LT and decide that communication is successful. This assumption was made not only to simplify the analysis, but because we found that multiple ready intervals were a second-order effect.

To simplify the presentation, we make the following definitions:

- DS = RS + Δ_s, deadline of the sender.
- DR = RR + Δ_r, deadline of the receiver.
- LT=DS − d, latest time receiver should respond to sender.
- M=RS + X_m, time at which initial message arrives at receiver.
- X_t = time at which sender knows of outcome.

A. Measurements

In order for Algorithm 1 to succeed, the receiver must say "Yes" (which happens for Ord(1), ⋯, Ord(4), see Table I), and the sender must receive the acknowledgment by its deadline. Since replies are sent at time M in Ord(1) and Ord(2), and at time RR in Ord(3) and Ord(4), X_t is computed by $M + X_a$ in the former and by RR $+ X_a$ in the latter. Thus,

$$\begin{aligned} Success_1 &= \Pr[X_t \leq \mathrm{DS} \mid \mathrm{Ord}(1), \cdots, \text{or } \mathrm{Ord}(4)] \\ &\quad \cdot \Pr[\mathrm{Ord}(1), \cdots, \text{or } \mathrm{Ord}(4)] \\ &= \Pr[M + X_a \leq \mathrm{DS} \mid \mathrm{Ord}(1) \text{ or } \mathrm{Ord}(2)] \\ &\quad \cdot \Pr[\mathrm{Ord}(1) \text{ or } \mathrm{Ord}(2)] \\ &\quad + \Pr[\mathrm{RR} + X_a \leq \mathrm{DS} \mid \mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)] \\ &\quad \cdot \Pr[\mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)]. \end{aligned}$$

Since the sender assumes that communication has failed if it does not receive an acknowledgment by its deadline, inconsistent decisions are possible only when the receiver decides "Yes" but the acknowledgment does not arrive in time. Thus,

$$\begin{aligned} Inconsistent &= \Pr[X_t > \mathrm{DS} \mid \mathrm{Ord}(1), \cdots, \text{ or } \mathrm{Ord}(4)] \\ &\quad \cdot \Pr[\mathrm{Ord}(1), \cdots, \text{ or } \mathrm{Ord}(4)] \\ &= \Pr[M + X_a > \mathrm{DS} \mid \mathrm{Ord}(1) \text{ or } \mathrm{Ord}(2)] \\ &\quad \cdot \Pr[\mathrm{Ord}(1) \text{ or } \mathrm{Ord}(2)] \\ &\quad + \Pr[\mathrm{RR} + X_a > \mathrm{DS} \mid \mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)] \\ &\quad \cdot \Pr[\mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)]. \end{aligned}$$

Success$_2$ is computed similarly to *Success*$_1$.

$$\begin{aligned} &Success_2 \\ &= \Pr[X_t \leq \mathrm{DS} \mid \mathrm{Ord}(1), \cdots, \text{ or } \mathrm{Ord}(6)] \\ &\quad \cdot \Pr[\mathrm{Ord}(1), \cdots, \text{ or } \mathrm{Ord}(6)] \\ &= \Pr[M + X_a \leq \mathrm{DS} \mid \mathrm{Ord}(1), \mathrm{Ord}(2), \mathrm{Ord}(5), \text{ or } \mathrm{Ord}(6)] \\ &\quad \cdot \Pr[\mathrm{Ord}(1), \mathrm{Ord}(2), \mathrm{Ord}(5), \text{ or } \mathrm{Ord}(6)] \\ &\quad + \Pr[\mathrm{RR} + X_a \leq \mathrm{DS} \mid \mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)] \\ &\quad \cdot \Pr[\mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)]. \end{aligned}$$

Late-Yes is defined on the same orderings as *Success*$_2$ except that replies arrive *after* the deadline of the sender.

TABLE II
BOUNDS FOR DISCRETE EVENTS

Ordering	bounds	RS	X_m	RR	X_a for $X_t \leq DS$	X_a for $X_t > DS$
Ord(1)	upper	T	Δ_1	$min(\Delta_4, T)$	Δ_7	∞
	lower	1	1	$max(1, \Delta_3)$	1	$max(1, \Delta_7 + 1)$
Ord(2)	upper	T	$\Delta_1 - 1$	$min(\Delta_3 - 1, \Delta_4, T)$	Δ_7	∞
	lower	1	1	$max(1, \Delta_5)$	1	$max(\Delta_7 + 1)$
Ord(3)	upper	T	$\Delta_1 - 1$	$min(\Delta_2, T)$	Δ_6	∞
	lower	1	1	$max(\Delta_4 + 1, \Delta_3)$	1	$max(1, \Delta_6 + 1)$
Ord(4)	upper	T	$\Delta_1 - 1$	$min(\Delta_3 - 1, T)$	Δ_6	∞
	lower	1	1	$\Delta_4 + 1$	1	$max(1, \Delta_6 + 1)$
Ord(5)	upper	T	∞	$min(\Delta_2, T)$	Δ_7	∞
	lower	1	$\Delta_1 + 1$	$max(1, \Delta_5)$	1	$max(1, \Delta_7 + 1)$
Ord(6)	upper	T	∞	$min(\Delta_4, T)$	Δ_7	∞
	lower	1	$\Delta_1 + 1$	$max(1, \Delta_5, \Delta_2 + 1)$	1	$max(1, \Delta_7 + 1)$
Ord(7)	upper	T	∞	T	Δ_7	∞
	lower	1	$\Delta_1 + 1$	$\Delta_4 + 1$	1	$max(1, \Delta_7 + 1)$
Ord(8)	upper	T	∞	$min(\Delta_5 - 1, T)$	Δ_7	∞
	lower	1	$\Delta_1 + 1$	$max(1, \Delta_2 + 1)$	1	$max(1, \Delta_7 + 1)$
Ord(9)	upper	T	∞	$min(T, \Delta_2, \Delta_5 - 1)$	Δ_7	∞
	lower	1	$\Delta_1 + 1$	$max(1, \Delta_3)$	1	$max(1, \Delta_7 + 1)$
Ord(10)	upper	T	∞	$min(\Delta_3 - 1, T)$	Δ_7	∞
	lower	1	$\Delta_1 + 1$	1	1	$max(1, \Delta_7 + 1)$
Ord(11)	upper	T	Δ_1	$min(\Delta_5 - 1, T)$	d	∞
	lower	1	1	1	1	$d + 1$
Ord(12)	upper	T	Δ_1	T	d	∞
	lower	1	1	$\Delta_2 + 1$	1	$d + 1$

Note:

1. $\Delta_1 = \Delta_s - d$
2. $\Delta_2 = RS + \Delta_s - d$
3. $\Delta_3 = RS + \Delta_s - \Delta_r - d$
4. $\Delta_4 = RS + X_m$
5. $\Delta_5 = RS + X_m - \Delta_r$
6. $\Delta_6 = RS + \Delta_s - RR$
7. $\Delta_7 = \Delta_s - X_m$

Late-Yes

$$= \Pr[X_t > \mathrm{DS} | \mathrm{Ord}(1), \cdots, \text{ or } \mathrm{Ord}(6)]$$
$$\cdot \Pr[\mathrm{Ord}(1), \cdots, \text{ or } \mathrm{Ord}(6)]$$
$$= \Pr[M + X_a > \mathrm{DS} | \mathrm{Ord}(1), \mathrm{Ord}(2), \mathrm{Ord}(5), \text{ or } \mathrm{Ord}(6)]$$
$$\cdot \Pr[\mathrm{Ord}(1), \mathrm{Ord}(2), \mathrm{Ord}(5), \text{ or } \mathrm{Ord}(6)]$$
$$+ \Pr[\mathrm{RR} + X_a > \mathrm{DS} | \mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)]$$
$$\cdot \Pr[\mathrm{Ord}(3) \text{ or } \mathrm{Ord}(4)].$$

Since the sender waits for an explicit acknowledgment, the sender may also have to wait beyond its deadline for "No" message. X_t equals $M + X_a$ in Ord (7), Ord (8), Ord (9), and Ord (10), and LT $+ X_a$ in Ord (11) and Ord (12). Thus,

$$\textit{Late-No} = \Pr[X_t > \mathrm{DS} | \mathrm{Ord}(7), \cdots, \text{ or } \mathrm{Ord}(12)]$$
$$\cdot \Pr[\mathrm{Ord}(7), \cdots, \text{ or } \mathrm{Ord}(12)]$$
$$= \Pr[M + X_a > \mathrm{DS} | \mathrm{Ord}(7), \cdots, \text{ or } \mathrm{Ord}(10)]$$
$$\cdot \Pr[\mathrm{Ord}(7), \cdots, \text{ or } \mathrm{Ord}(10)]$$
$$+ \Pr[\mathrm{LT} + X_a > \mathrm{DS} | \mathrm{Ord}(11) \text{ or } \mathrm{Ord}(12)]$$
$$\cdot \Pr[\mathrm{Ord}(11) \text{ or } \mathrm{Ord}(12)].$$

B. Individual Probabilities

Table II shows the bounds needed to compute the probability that $X_t \leq$ DS and $X_t >$ DS for each ordering; the derivation of these bounds can be found in the Appendix. Recall that the values of d, Δ_s, and Δ_r are variables whose values are known.

For example, using this table we calculate

$$\Pr[X_t \leq DS | \text{ Ord } (1)] \Pr \text{ [Ord } (1)]$$
$$= \sum_{i=1}^{T} \sum_{j=1}^{\Delta_s - d} \sum_{k=\max(1, i+\Delta_s-\Delta_r-d)}^{\min(i+j, T)} \sum_{l=1}^{\Delta_s - j} \Pr[\mathrm{RS} = i]$$
$$\cdot \Pr[X_m = j] \Pr[\mathrm{RR} = k] \Pr[X_a = l].$$

Since the orderings are mutually exclusive, *Success₁*, *Inconsistency*, *Success₂*, *Late-Yes,* and *Late-No* can be computed by summing appropriate probabilities from events in

Ord(i). For example,

$$\begin{aligned}
\mathit{Success}_1 &= \Pr[M + X_a \le \mathrm{DS}|\mathrm{Ord}(1)]\Pr[\mathrm{Ord}(1)] \\
&\quad + \Pr[M + X_a \le \mathrm{DS}|\mathrm{Ord}(2)]\Pr[\mathrm{Ord}(2)] \\
&\quad + \Pr[\mathrm{RR} + X_a \le \mathrm{DS}|\mathrm{Ord}(3)]\Pr[\mathrm{Ord}(3)] \\
&\quad + \Pr[\mathrm{RR} + X_a \le \mathrm{DS}|\mathrm{Ord}(4)]\Pr[\mathrm{Ord}(4)] \\
&= \sum_{i=1}^{T}\sum_{j=1}^{\Delta_s-d}\sum_{k=\max(1,i+\Delta_s-\Delta_r-d)}^{\min(i+j,T)}\sum_{l=1}^{\Delta_s-j} \Pr[\mathrm{RS}=i]\Pr[X_m=j]\Pr[\mathrm{RR}=k]\Pr[X_a=l] \\
&\quad + \sum_{i=1}^{T}\sum_{j=1}^{\Delta_s-d}\sum_{k=\max(1,i+j-\Delta_r)}^{\min(i+\Delta_s-\Delta_r-d-1,i+j,T)}\sum_{l=1}^{\Delta_s-j} \Pr[\mathrm{RS}=i]\Pr[X_m=j]\Pr[\mathrm{RR}=k]\Pr[X_a=l] \\
&\quad + \sum_{i=1}^{T}\sum_{j=1}^{\Delta_s-d-1}\sum_{k=\max(i+j+1,i+\Delta_s-\Delta_r-d)}^{\min(i+\Delta_s-d,T)}\sum_{l=1}^{i+\Delta_s-k} \Pr[\mathrm{RS}=i]\Pr[X_m=j]\Pr[\mathrm{RR}=k]\Pr[X_a=l] \\
&\quad + \sum_{i=1}^{T}\sum_{j=0}^{\Delta_s-d-1}\sum_{k=i+j+1}^{\min(i+\Delta_s-\Delta_r-d-1,T)}\sum_{l=1}^{i+\Delta_s-k} \Pr[\mathrm{RS}=i]\Pr[X_m=j]\Pr[\mathrm{RR}=k]\Pr[X_a=l].
\end{aligned}$$

IV. Performance Evaluation

As shown in the previous section, the five performance measures depend on the probabilities of the orderings of events M, LT, RR, and DR. Furthermore, these probabilities depend on the values of d, Δ_r, Δ_s, and T. Thus, it is possible to predict how these parameters influence the performance of the two algorithms. In this section, we discuss the effects of individual parameters on the five measures.

To illustrate their effects, we present sample graphs. The distributions used for the random variables in these examples are as follows.

Assumption 4: Message delay (used for X_m and X_a) is geometrically distributed. That is, $\Pr[X_m = k] = \Pr[X_a = k] = P(1-P)^{k-1}$, with the mean

$$\sum_{i=1}^{\infty} kP(1-P)^{k-1} = 1/P.$$

The values of P used in the evaluation are 1/2, 1/4, and 1/8; Fig. 4 shows these distributions. The geometric message delay distribution was used because it is easy to compute, and has a similar shape to that predicted by Wong [9].

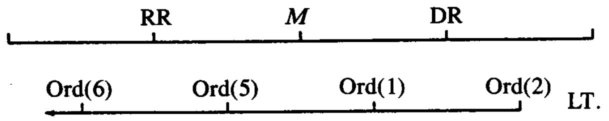
We also assume the following distribution of ready times for the sender and receiver:

Assumption 5: The time at which a process becomes ready to communicate (i.e., RR for the receiver and RS for the sender) is uniformly distributed over the interval $[1, T]$.

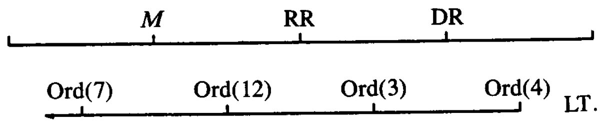
Note that this does not imply that DR and DS must occur before T.

A. *Effect of d*

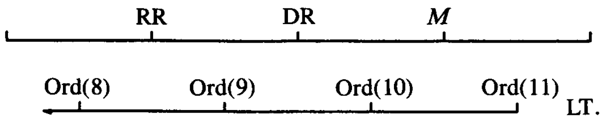
As d increases, LT becomes earlier. When M is fixed to arrive *during* the receiver's ready interval, $\mathit{Success}_1$ deteriorates as d increases since it says "No" if the message arrives after LT, while $\mathit{Success}_2$ remains unaffected. This effect is represented in the following diagram, which shows how the orderings change as LT becomes earlier relative to fixed RR, M, and DR:



When M is fixed to arrive *before* the receiver's ready interval, both $\mathit{Success}_1$ and $\mathit{Success}_2$ deteriorate by the same amount since they both say "No" if LT occurs before the receiver is ready:



However, when M is fixed to arrive *after* the receiver's ready interval, there is no effect on either $\mathit{Success}_1$ or $\mathit{Success}_2$ since the receiver will always say "No:"



Note that the only possible orderings when $d > \Delta_s$ are Ord$(5, 6, \cdots, 10)$, since messages must take some positive amount of time to be delivered. In these orderings, the receiver always says "No" for Algorithm 1, whereas the receiver only says "No" in Ord$(7, \cdots, 10)$ for Algorithm 2. Thus, the net effect is that $\mathit{Success}_1$ deteriorates more rapidly than $\mathit{Success}_2$ as d increases; when $d = \Delta_s$, $\mathit{Success}_1$ becomes zero while $\mathit{Success}_2$ remains constant at some possibly nonzero value, due to Ord$(5\,\&\,6)$. In practice, Δ_s should be chosen to be at least twice the average message delay if communication is to succeed; it is therefore unlikely that d should be chosen to exceed Δ_s, since this is a very pessimistic estimate of how long the return message should take. *Inconsistent* improves (decreases) as d increases and becomes zero when $d = \Delta_s$; the sender will always be correct if it assumes that communication has failed. Similarly, *Late-Yes* improves (decreases) as d increases and then remains constant for $d > \Delta_s$. *Late-No,* however, deteriorates (increases) as d increases and then remains constant when $d > \Delta_s$.

Fig. 5 shows the effect of d for $T = 16$, $P = 0.25$, and $\Delta_r = \Delta_s = 24$. Note that these parameter values imply that the ready intervals of the sender and receiver must overlap by at least 8 time units, which is about twice the average message delay. Even in the worst case where RS $= 1$ and RR $= T$, any reply will almost always reach the sender by

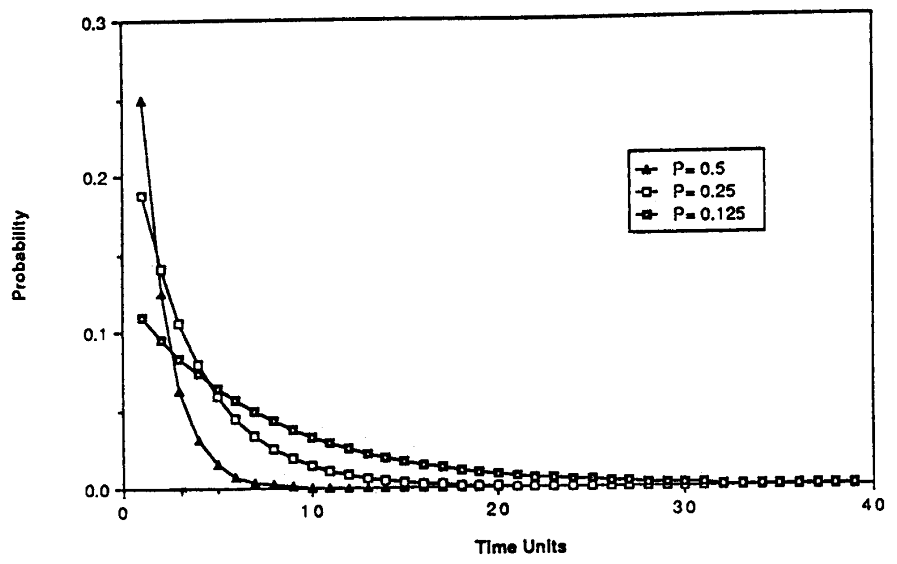


Fig. 4. Message delay distributions used for X_m and X_a.

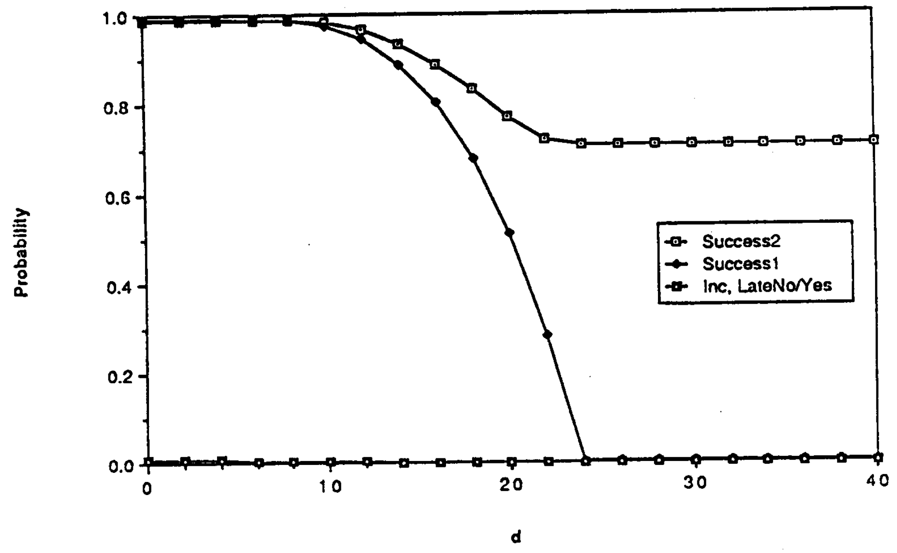
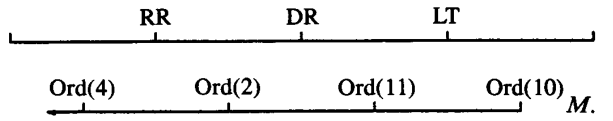


Fig. 5. Effect of d. ($T = 16$, $P = 0.25$, $\Delta_s = \Delta_r = 24$.)

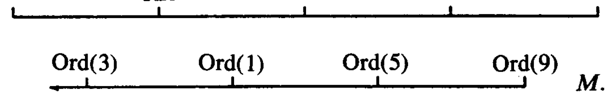
DS. Thus, *Inconsistent, Late-Yes*, and *Late-No* are very small. What is more interesting is the behavior of success for the algorithms. When $d = 0$, $Success_1 = Success_2$ since $\Pr[\text{Ord}(5)] = \Pr[\text{Ord}(6)] = 0$. The performance of both algorithms starts to drop off when d becomes larger than the minimum overlap of ready intervals (i.e., $d > 8$). As predicted, $Success_1$ drops sharply and hits zero when $d = \Delta_s = 24$, whereas $Success_2$ drops less sharply and remains constantly high due to the effect of Ord (5) and Ord (6).

B. Effect of P

As P increases, the average message delay decreases and M occurs earlier. If LT is fixed to occur after DR, $Success_1$ and $Success_2$ both improve since they both say "Yes" if the message arrives before DR:



When LT is fixed to occur during the receiver's ready interval, the success of both algorithms again improves. Note that $Success_1$ improves less than $Success_2$ since it says "No" if the message arrives after LT even if the message arrives during the receiver's ready interval.



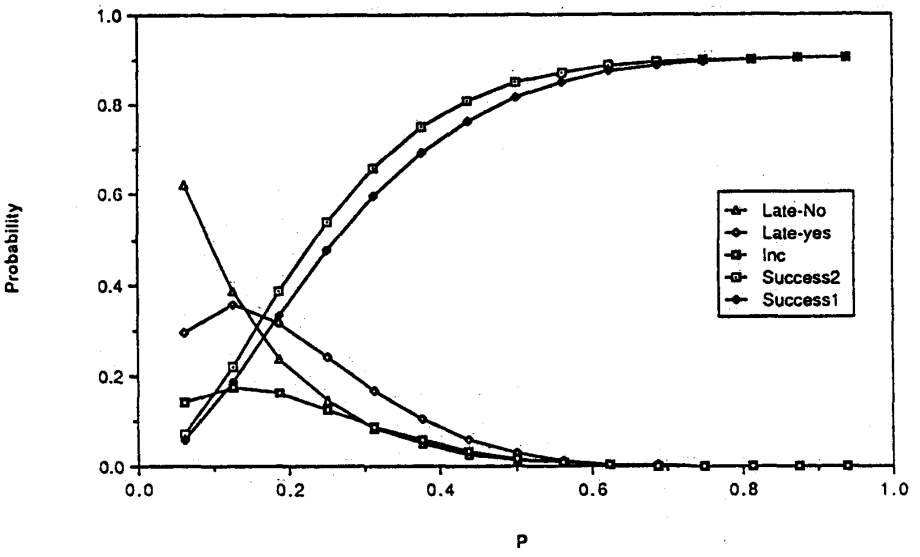
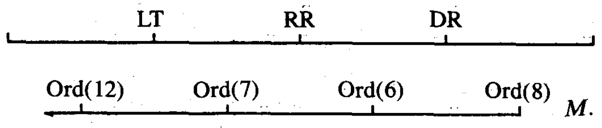


Fig. 6. Effect of P. ($T = 8, d = 4, \Delta_s = \Delta_r = 8$.)

Lastly, when LT is fixed to occur before RR, there is no effect on *Success*$_1$, while *Success*$_2$ improves *transiently* since it says "Yes" if the message arrives during its ready interval even if this is after LT (Ord (6)):



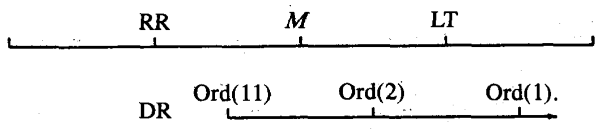
Since the average message delay is very large when P is small, the original message will almost certainly arrive after the receiver's ready interval and the "No" reply will arrive at the sender long after DS. Therefore, *Late-No* starts at 1 and diminishes to zero as P approaches 1. On the other hand, *Success*$_1$ and *Success*$_2$ start small and approach $1 - \Pr[\mathrm{Ord}(12)]$ as P approaches 1, since Ord(12) is the only case in which the algorithms say "No" as the message delay becomes very small. *Inconsistent* and *Late-Yes* are also small when P is small, but increase as *Success*$_1$ and *Success*$_2$ increase. However, they eventually *decrease* as P approaches 1 since a "Yes" message is more likely to arrive within the sender's deadline if the average message delay is small, all other factors being held constant.

Fig. 6 shows the effect of P for $T = 8$, $d = 4$, and $\Delta_r = \Delta_s = 8$. P varies as $i/16$ for $i = 1, 2, \cdots, 15$. The shapes of the slopes confirm the above prediction. Note that this choice of parameters does not guarantee that the ready intervals of the sender and receiver will overlap. In the worst case, the sender's deadline may occur exactly when the receiver becomes ready. In this case, no matter what the average message delay, a response cannot be received by the sender before its deadline. Therefore, *Success*$_2$ does not converge to 1 as P approaches 1.

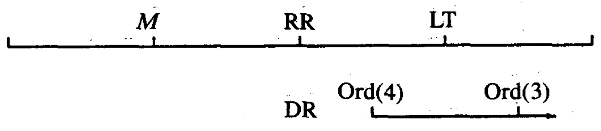
Figs. 7 (Algorithm 1) and 8 (Algorithm 2) contrast the effects of P and d for $T = 16$, $\Delta_r = \Delta_s = 24$. As d increases, all five probabilities decrease, whereas as P increases, all five probabilities increase. It is interesting to note in Fig. 8 that when $d > \Delta_s$, the curve initially increases, and then slowly decreases. This is not due to late message arrival since *Late-No* and *Late-Yes* do not increase. Rather it is due to the shift from Ord (6) (in which the receiver says "Yes") to Ord (7) (in which the receiver says "No" because M and LT occur before RR).

C. *Effect of Δ_r and Δ_s*

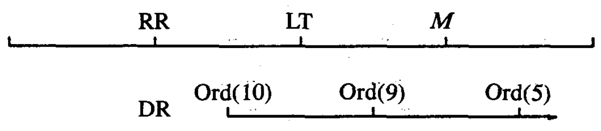
As Δ_r increases, DR increases, and the success of both algorithms improves. When RR $< M <$ LT, the success of both algorithms improves :

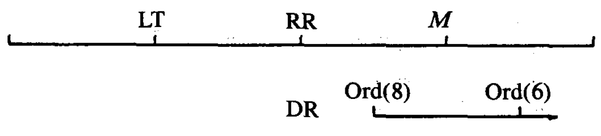


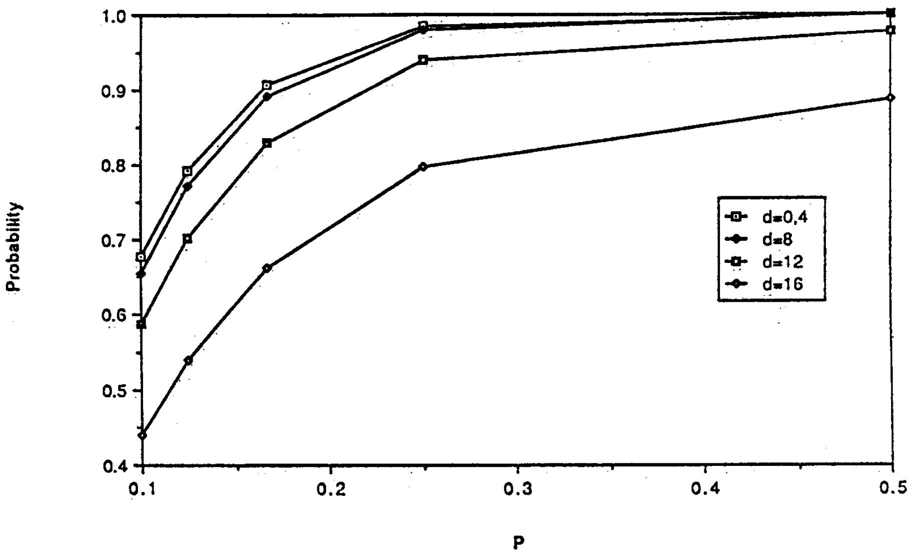
When M is fixed to arrive before the receiver's ready interval, changing Δ_r has no effect on either algorithm:



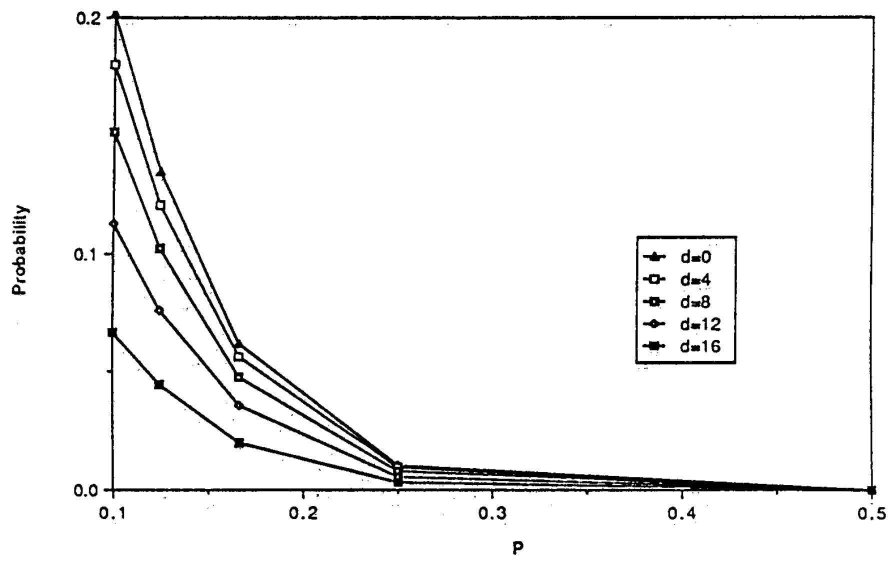
When M arrives after LT and RR, the success of Algorithm 1 is unaffected, but the success of Algorithm 2 improves as DR becomes later than M:







(a) *Success*$_1$



(b) *Inconsistent*

Fig. 7. Effect of d and P on Algorithm 1. ($T = 16$, $\Delta_s = \Delta_r = 24$.)

However, if M and LT occur before RR, DR can only be the last event (Ord(7 & 12)): both algorithms say "No" and there is no effect on either of their success. Thus, as Δ_r increases, *Late-No* decreases and then remains constant at

$$\Pr[X_t > \mathrm{DS}|\mathrm{Ord}(7) \text{ or } \mathrm{Ord}(12)]\Pr[\mathrm{Ord}(7) \text{ or } \mathrm{Ord}(12)].$$

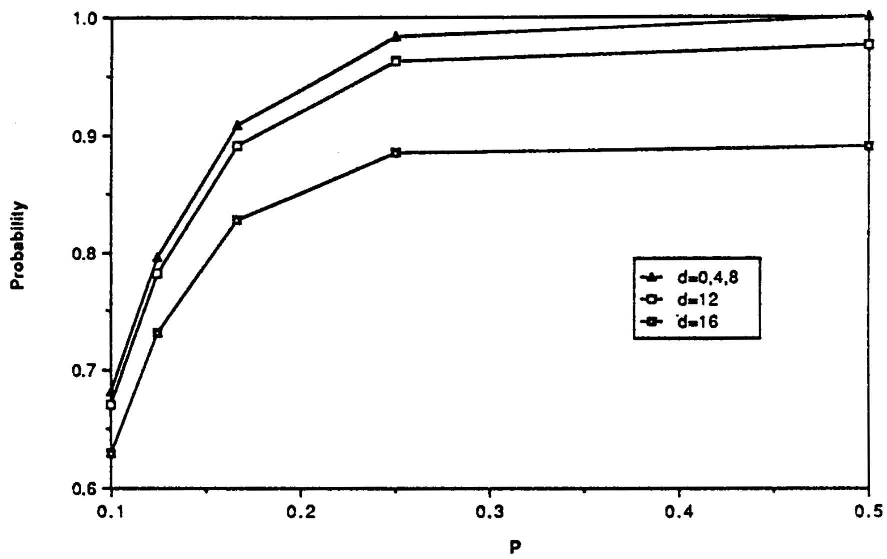
Furthermore, *Success*$_1$, *Success*$_2$, *Inconsistent,* and *Late-Yes* increase and then remain constant at

$$\mathit{Success}_1 = \Pr[X_t \le \mathrm{DS}|\mathrm{Ord}(1 \text{ or } 3)]\Pr[\mathrm{Ord}(1 \text{ or } 3)]$$

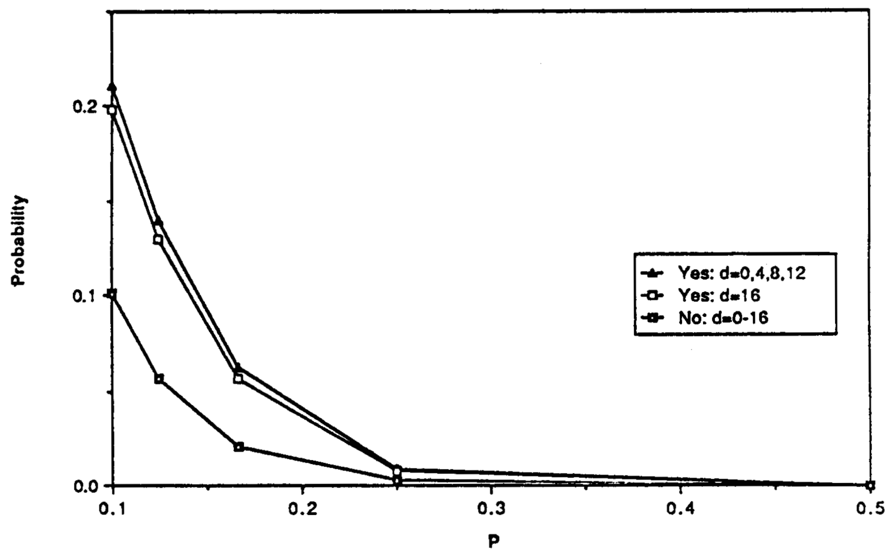
$$\mathit{Success}_2 = \Pr[X_t \le \mathrm{DS}|\mathrm{Ord}(1, 3, 5 \text{ or } 6)] \cdot \Pr[\mathrm{Ord}(1, 3, 5 \text{ or } 6)]$$

$$\mathit{Inconsistent} = \Pr[X_t > \mathrm{DS}|\mathrm{Ord}(1 \text{ or } 3)]\Pr[\mathrm{Ord}(1 \text{ or } 3)]$$

$$\mathit{Late\text{-}Yes} = \Pr[X_t > \mathrm{DS}|\mathrm{Ord}(1, 3, 5 \text{ or } 6)] \cdot \Pr[\mathrm{Ord}(1, 3, 5 \text{ or } 6)].$$



(a) *Success₂*



(b) *Late-Yes* and *Late-No*

Fig. 8. Effect of d and P on Algorithm 2. ($T = 16$, $\Delta_r = \Delta_s = 24$.)

Fig. 9 shows an example of this for $T = 16$, $P = 0.25$, $d = 4$, and $\Delta_s = 24$. Note that *Success₁* and *Success₂* both stabilize by $\Delta_r = 24$, since in the worst case RS $= T$ and RR $= 1$, but the receiver will still receive most messages in its ready interval and be able to respond "Yes."

As Δ_s increases, LT becomes later. The opposite effect has already been discussed (see Section IV-A, the effect of changing d). Note that this time, Ord(2), Ord(4), and Ord(11) are unaffected. As Δ_s increases, the sender is more likely to receive an acknowledgment message within its deadline. Thus, as Δ_s increases, *Success₁* and *Success₂* approach $1 - \Pr[\text{Ord}(11)]$, and *Inconsistent, Late-Yes,* and *Late-No* approach zero.

Fig. 10 shows the effect of increasing Δ_s for $T = 16$, $P = 0.25$, $\Delta_r = 24$, and $d = 4$. This time, the success of both algorithms stabilizes later ($\Delta_s = 32$) since there must be enough time for the initial message to be sent to the receiver as well as for the reply to be received. The behavior of *Inconsistent* is also very interesting as it initially increases and then decreases, peaking at $\Delta_s \approx 8$; there is also a gap between the

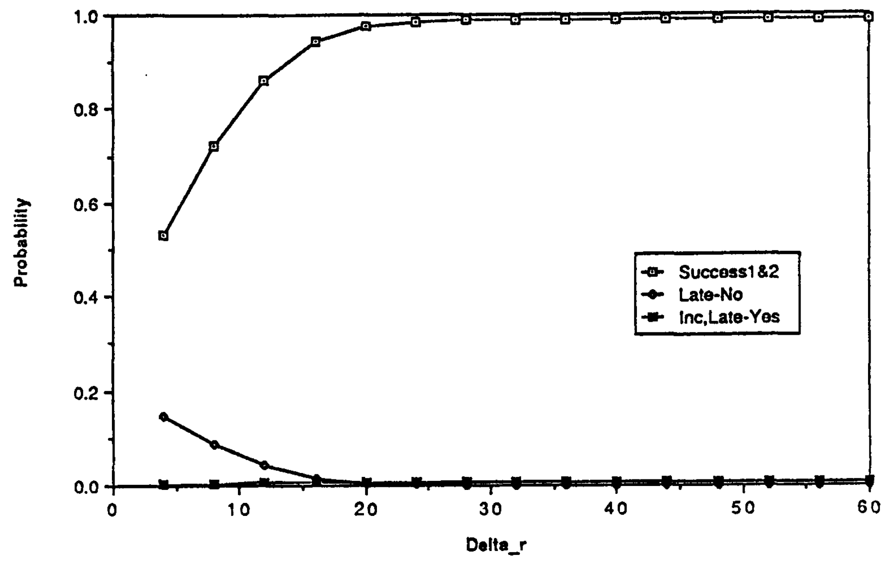


Fig. 9. Effect of Δ_r. ($T = 16$, $P = 0.25$, $d = 4$, $\Delta_s = 24$.)

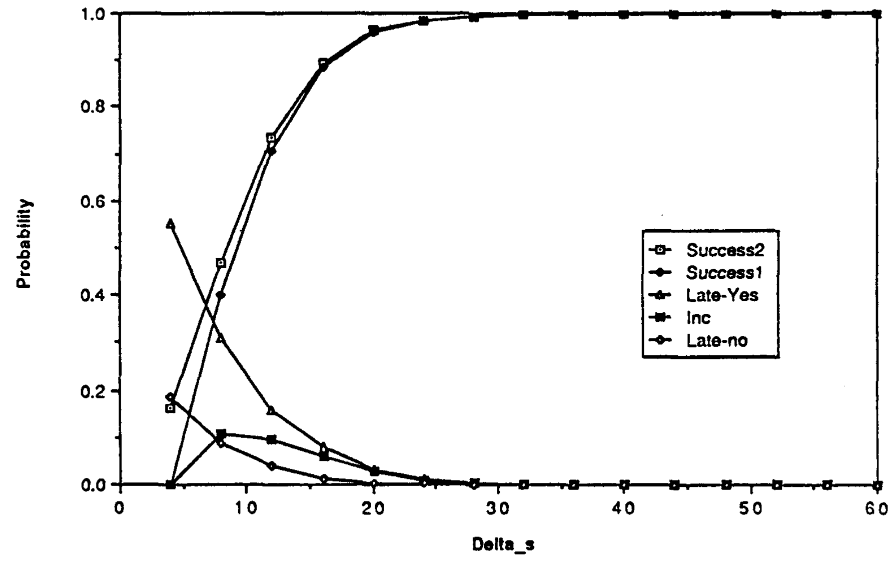


Fig. 10. Effect of Δ_s. ($T = 16$, $P = 0.25$, $d = 4$, $\Delta_r = 24$.)

curves for *Success₁* and *Success₂* at this point. The explanation for both of these effects is that d has been chosen to be the mean message delay; furthermore, since Δ_s is small, if the message arrives before LT it is probably *just* before LT. The decision to send "Yes" to the sender is therefore not very safe, i.e., it will very probably arrive late.

D. Effect of T

As mentioned earlier, T reflects how well the executions of the sender and receiver are synchronized. As T increases, communication is less likely to succeed since the ready intervals are less likely to overlap; that is, as T increases, $\Pr[\mathrm{Ord}(i)]$ for $i = 1, \cdots, 6, 8, 9$ decrease to zero. Thus, *Success₁*, *Success₂*, *Inconsistent*, and *Late-Yes* approach zero as T gets arbitrarily large. On the other hand, *Late-No* increases as T increases and converges to

$$\begin{aligned}&\Pr[M + X_a > \mathrm{DS} \mid \mathrm{Ord}(7) \text{ or } \mathrm{Ord}(10)]\\&\cdot \Pr[\mathrm{Ord}(7) \text{ or } \mathrm{Ord}(10)]\\&+ \Pr[\mathrm{LT} + X_a > \mathrm{DS} \mid \mathrm{Ord}(11) \text{ or } \mathrm{Ord}(12)]\\&\cdot \Pr[\mathrm{Ord}(11) \text{ or } \mathrm{Ord}(12)].\end{aligned}$$

Thus, when Δ_r, Δ_s, d, and the message distribution are fixed, this value can be explicitly calculated as

$$\Pr[X_m + X_a > \Delta_s \text{ and } X_m > \Delta_s - d] + \Pr[X_a > d \text{ and } X_m \le \Delta_s - d].$$

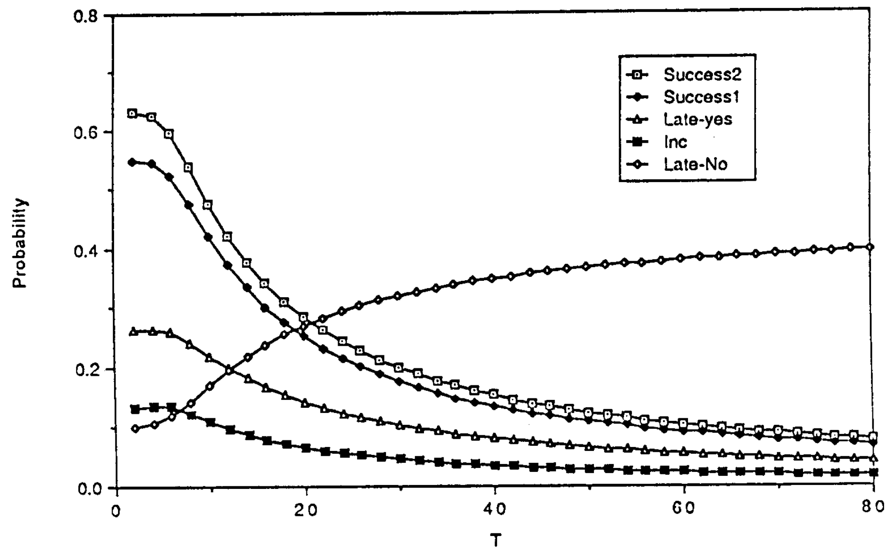


Fig. 11. Effect of T. ($\Delta_r = \Delta_s = 8, P = 0.25, d = 4$.)

Fig. 11 shows the effect of T for $\Delta_r = \Delta_s = 8, P = 0.25$, and $d = 4$. As T becomes very large, the sender is blocked beyond its deadline waiting for belated "No" message with the probability of 0.4499, which is

$$\Pr[X_m + X_a > 8 \text{ and } X_m > 4] + \Pr[X_a > 4 \text{ and } X_m \leq 4]$$
$$= 0.2336 + 0.2163 = 0.4499.$$

V. Coping with Failures

So far in evaluating the performance of the two algorithms, we have not considered their fault tolerance. We now discuss the effect of message loss, failure of the sender or receiver, and communication link failures.

A. Message Loss

Let f be the probability that any given message is lost. To distinguish this from the case of a message being delivered after an arbitrarily long delay (∞), we assume a very large constant ub after which messages are said to be lost. The probability that a given message will take longer than ub to be delivered approaches 0; any message that takes longer than ub to be delivered is thrown away. Thus,

$$\sum_{i=1}^{ub} \Pr(X_m = i) = 1 - f.$$

We therefore assume that ∞ in the upper bounds for X_a in Table II is replaced by ub.

In extending the analysis of Algorithms 1 and 2 from the previous section, we use primes (′) to distinguish the analysis *with* transient message loss from that *without*. We also assume that

$$\Pr[X'_m = i] = (1 - f)\Pr[X_m = i]$$

to simplify the extensions. Thus,

$$\Pr[\text{Ord}'(i)] = (1 - f)\Pr[\text{Ord}(i)].$$

In Algorithm 1, communication is only successful if both M and the response Ack are delivered without failure and in time:

$$\mathit{Success}'_1 = (1 - f)^2\, \mathit{Success}_1.$$

If M is lost then the sender will time out and correctly assume that communication has failed. However, if the response Ack is lost, the receiver may have decided "Yes," and an inconsistent decision will be made.

$$\mathit{Inconsistent}' = (1 - f)\, \mathit{Inconsistent} + f(1 - f)\, \mathit{Success}_1.$$

Thus, as f increases, $\mathit{Success}'_1$ will decrease. The effect on *Inconsistent′* is harder to predict since increasing f increases the probability of failure of the final message, in which case the sender may decide "unsuccessful" when the reply is actually "Yes;" whereas, increasing f also increases the probability that the initial message will fail, in which case the sender makes the correct decision. However, since *Inconsistent* is normally dominated by $\mathit{Success}_1$, and since f should be very small, *Inconsistent′* should also increase (but less rapidly than $\mathit{Success}'_1$) as f increases.

In Algorithm 2, communication is again successful only if both M and a positive response Ack are delivered in time:

$$\mathit{Success}'_2 = (1 - f)^2\, \mathit{Success}_2.$$

Since loss of either M or Ack results in the sender waiting forever, a case that is not covered in either *Late-Yes* or *Late-No, Late-Yes′* and *Late-No′* can be calculated as follows:

$$\mathit{Late\text{-}Yes}' = (1 - f)^2\, \mathit{Late\text{-}Yes}$$
$$\mathit{Late\text{-}No}' = (1 - f)^2\, \mathit{Late\text{-}No}.$$

The analysis of Algorithm 2 must also be expanded to include the probability that the sender will wait forever for a response:

$$\textit{Wait-Forever} = f + (1-f)f.$$

Thus, as f increases *Success*$_2$, *Late-Yes,* and *Late-No* all decrease, and *Wait-Forever* increases at their combined rate.

The effect of message loss highlights the deficiencies of both Algorithms 1 and 2: the potential for (undetected) inconsistent decisions in Algorithm 1, and the potential of waiting for the decision past the sender's deadline (perhaps indefinitely) in Algorithm 2. A solution to these problems is to blend Algorithms 1 and 2 (see Fig. 3). When the receiver receives a message, it must respond with "Yes" or "No," and behaves as in Algorithm 2. The sender, however, only waits until DS for the receiver's decision, as in Algorithm 1. If the deadline expires without the decision being received, the sender concludes "Don't know." An exception handler is then triggered, and the programmer defines the appropriate action to be taken [10]. Note that in this approach a "three-value" semantics is used for the success or failure of communication: *successful, unsuccessful,* or *don't know*. The meaning of consistent decisions would be: if the receiver says "No" or never receives the initial message, the sender must decide "unsuccessful" or "don't know;" and if the receiver says "Yes" the sender must decide "successful" or "don't know."

In the analysis of this new algorithm, note that *Success*$'_3$ is the same as *Success*$'_2$. The sender will never wait forever, and inconsistencies are not possible; however, there is the possibility that the sender is undecided by its deadline. This happens if the original message is lost, the reply is lost, or if the reply is not received in time:

$$\textit{Undecided} = \textit{Wait-Forever} + \textit{Late-No} + \textit{Late-Yes}.$$

One action that could be taken when the potential for an inconsistent decision is detected (i.e., when the sender concludes "don't know") is to poll the receiver at timeout intervals to determine what, if any, decision had been made. If the receiver never received the message, it could respond "No," as if the message had not been received in time (Ord(7–10)). Whatever the response of the receiver, it must improve *monotonically* with time [6], [7]; that is, if the receiver ever answers with "Yes" it must always answer "Yes," and the same must be true for a negative response. However, if the receiver answers with "don't know," it can improve its answer to either "Yes" or "No" at a future point in time. On the other hand, message loss is most likely to occur under peak load conditions, and it is precisely at this time that we do not want to start flooding the network with messages from the sender to the receiver asking for the results of communication. It is therefore important not to built this type of error recovery into the implementation of timed synchronous communication primitives, but to have it used explicitly by the programmer.

B. Other Failures

Processor Failures: In both algorithms, the sender is "in-doubt" after it sends the initial message, and before it has made a decision (either explicitly or by reaching DS). If the sender fails in this window, upon recovery it can either try to resume where it left off when it crashed, or decide "don't know" (as discussed above). To resume where it left off, the sender must know what the receiver has decided; that is, recovery must include placing the receiver's decision on the sender's queue, if any has been made. Recovery must also involve setting the clock to the current time so the sender can determine whether or not the deadline has expired in Algorithm 1.

To cope with failure of the receiver, recovery must involve: reinitializing its state from the time of failure, placing any message received during the failure on its queue, and reinitializing its clock. If the message arrives during the failure, or if the failure occurs after the message was received but before it was answered, the receiver takes appropriate action based on the current time at recovery. Note that the effect on the sender is the same as if the *initial* message took longer to be delivered: in Algorithm 1, the sender will make a decision at DS (and be correct with the same probability as if the receiver had not failed), whereas in Algorithm 2, the sender will wait until the receiver recovers and responds.

Communication Link Failures: Apart from message loss, the effect of communication link failures is that messages may be consistently rerouted causing the average message delay to increase. The probability of inconsistent decisions in Algorithm 1 will therefore increase unless the value chosen for d is also increased. The success rate of both algorithms will also deteriorate since d increases.

VI. Conclusions

Protocols for timed-synchronous communication can be defined by how the receiver responds in each of the orderings in Table I. Two such protocols were described in this paper, which have previously appeared in the literature: Algorithm 1 was originally proposed in [4]. Algorithm 2 was adapted from [5], which discussed various implementations of the rendezvous construct in Ada. The protocols differ primarily in how they "fail" to meet the absolute correctness statement: Algorithm 1 allows inconsistent decisions, whereas Algorithm 2 allows the sender to be delayed past its deadline. Algorithm 1 also implicitly assumes failure of the message by expiration of the deadline, while Algorithm 2 requires a negative acknowledgment.

Although the performance analysis in this paper is specific to Algorithms 1 and 2, the model developed can be used to analyze the performance of *any* protocol for timed-synchronous communication by simple modifications of the orderings included in each criterion measured: *Success, Late-Yes, Late-No,* and *Inconsistent*. The model and cost equations can also be used to derive good values for d, Δ_s, and Δ_r, as long as the end-to-end message delay distribution and relative synchronization of sender and receiver (i.e., the distributions for RR and RS) are known.

Even without knowledge of specific operating conditions, our analysis provides some general conclusions about the relative performance of Algorithms 1 and 2.

- From Table I, it is easy to see that *Success*$_1$ is always less than or equal to *Success*$_2$. However, this does not mean that

Algorithm 2 is *better* than Algorithm 1 since *Inconsistent* is also always less than or equal to *Late-Yes*. That is, Algorithm 1 is generally not as successful as Algorithm 2, but also does not fail to meet the original problem statement as often.

- $Success_1$ and $Success_2$ can be improved by decreasing T or increasing d, P, Δ_r, or Δ_s. However, *Inconsistent* and *Late-Yes* also increase as $Success_1$ and $Success_2$ increase, except when their improvements are due to an increase in Δ_s. In this case, *Inconsistent* and *Late-Yes* also decrease. On the other hand, as $Success_2$ increases, *Late-No* decreases. Thus, increasing Δ_s not only improves the success of both algorithms, but decreases their errors.
- Δ_s should be larger than $2 \times$ *average message delay*; otherwise, communication will almost always fail or errors will occur frequently.
- d should be larger than the *average message delay* to avoid a "Yes" message arriving after the sender's deadline in Algorithm 1.

Whether it is better to make inconsistent decisions or to allow the sender to be delayed past its deadline is controversial. On the one hand, inconsistent decisions may cause conflicting actions to be taken. On the other hand, hard real-time systems frequently shut down completely if deadlines cannot be met, since missing one deadline may cause a cascade of missed deadlines. Neither of these outcomes is desirable. It is therefore important to design the system correctly (by appropriately defining T, Δ_s, Δ_r, and message delivery guarantees) so that *Inconsistent* and *Late-No+Late-Yes* are virtually 0.

Another option is to further weaken the definition of "correctness," to allow a three-valued semantics: "Yes," "No," and "Don't know." This definition is achieved by a combination of Algorithms 1 and 2 (Algorithm 3). Algorithm 3 guarantees that the sender will not be delayed past its deadline, and also that it will never make an inconsistent decision; however, the sender may only be able to conclude "Don't know" rather than "Yes" or "No" at time DS. The motivation for allowing this is that the sender may be able to take some corrective action when in doubt at time DS (for example, polling the receiver for its response), or may be able to take a "soft" form of error recovery or perform defensive action if it knows for sure that communication has failed at time DS (for example, the sender may assume that the receiver has used filtering or estimation as is working with an incomplete knowledge of the environment). This also avoids certain pathological behaviors when transient message failures occur: the sender can detect that an inconsistent decision has been made (unlike Algorithm 1), and the sender will not wait forever (unlike Algorithm 2). The analysis of this protocol, with an additional assumption about the probability that a given message fails (f), was described in Section V.

Although this discussion has ignored clock discrepancy, the original algorithms take this into consideration [4], [5]. To adjust for clock drift, the receiving process needs to adjust its estimate of LT using either a worst case assumption about the difference between clocks (clock drift), or a worst case assumption about the rate at which the clocks tick and how long the original message from the sender took to be delivered (clock rate). The effect on the algorithms is the same as increasing d: the worse the clock drift, the earlier LT becomes in absolute time, and the success of both algorithms deteriorates.

Other directions for research include using a more realistic message distribution based on observations from "real" systems, and evaluating the performance of N-way timed-synchronous communication primitives (a form of two-phase commit with timing constraints, see [4]).

Appendix

A. Orderings and Derivation of Problems

To calculate the probability of each ordering shown in Table I, we assume that all random variables used are discrete. Each probability therefore has the general form

$$\sum_{i=lb_i}^{ub_i} \sum_{j=lb_j}^{ub_j} \sum_{k=lb_k}^{ub_k} \sum_{l=lb_l}^{ub_l} \Pr[\mathrm{RS}=i]\Pr[X_m=j] \cdot \Pr[\mathrm{RR}=k]\Pr[X_a=l].$$

Loose bounds for some of these upper and lower bounds are derived from our assumptions: $1 \le \mathrm{RS}$, $\mathrm{RR} \le T$, and $1 \le X_a$, $X_m < \infty$. However, the upper and lower bounds must be tightened to reflect relative positions of events in each ordering (RR, M, LT, and DR). Since RR < DR always, we must only consider the relative positions of M to LT, RR to LT, DR to LT, RR to M, and DR to M. This is translated to bounds on the random variables in each ordering as follows:

1) M $(= \mathrm{RS} + X_m)$ to LT $(= \mathrm{RS} + \Delta_s - d)$: Let $\Delta_1 = \Delta_s - d$.

- $M \le \mathrm{LT}$ in Ord(1, 11, 12): $X_m \le \Delta_1$ (upper bound)
- $M < \mathrm{LT}$ in Ord(2–4): $X_m \le \Delta_1 - 1$ (upper bound)
- $M > \mathrm{LT}$ in Ord(5–10): $X_m \ge \Delta_1 + 1$ (lower bound).

2) RR to LT $(= \mathrm{RS} + \Delta_s - d)$: Let $\Delta_2 = \mathrm{RS} + \Delta_s - d$.

- $\mathrm{RR} \le \mathrm{LT}$ in Ord(1, 3, 5, 9): $\mathrm{RR} \le \Delta_2$ (upper bound)
- $\mathrm{RR} < \mathrm{LT}$ in Ord(2, 4, 10, 11): $\mathrm{RR} \le \Delta_2 - 1$ (upper bound)
- $\mathrm{RR} > \mathrm{LT}$ in Ord(6, 7, 8, 12): $\mathrm{RR} \ge \Delta_2 + 1$ (lower bound).

3) DR $(= \mathrm{RR} + \Delta_r)$ to LT $(= \mathrm{RS} + \Delta_s - d)$: Let $\Delta_3 = \mathrm{RS} + \Delta_s - \Delta_r - d$.

- $\mathrm{DR} < \mathrm{LT}$ in Ord(2, 4, 10, 11): $\mathrm{RR} \le \Delta_3 - 1$ (upper bound)
- $\mathrm{DR} > \mathrm{LT}$ in Ord(5–8, 12): $\mathrm{RR} \ge \Delta_3 + 1$ (lower bound)
- $\mathrm{DR} \ge \mathrm{LT}$ in Ord(1, 3, 9): $\mathrm{RR} \ge \Delta_3$ (lower bound).

4) RR to M $(= \mathrm{RS} + X_m)$: Let $\Delta_4 = \mathrm{RS} + X_m$.

- $\mathrm{RR} \le M$ in Ord(1, 2, 6): $\mathrm{RR} \le \Delta_4$ (upper bound)
- $\mathrm{RR} < M$ in Ord(5, 8–11): $\mathrm{RR} \le \Delta_4 - 1$ (upper bound)
- $\mathrm{RR} > M$ in Ord(3, 4, 7, 12): $\mathrm{RR} \ge \Delta_4 + 1$ (lower bound).

5) DR $(= \mathrm{RR} + \Delta_r)$ to M $(= \mathrm{RS} + X_m)$: Let $\Delta_5 = \mathrm{RS} + X_m - \Delta_r$.

- $\mathrm{DR} < M$ in Ord(8–11): $\mathrm{RR} \le \Delta_5 - 1$ (upper bound)
- $\mathrm{DR} > M$ in Ord(3, 4, 7, 12): $\mathrm{RR} \ge \Delta_5 + 1$ (lower bound)
- $\mathrm{DR} \ge M$ in Ord(1, 2, 5, 6): $\mathrm{RR} \ge \Delta_5$ (lower bound).

For example, in Ord(7),

$$RR \geq \max(1, \Delta_2 + 1, \Delta_3 + 1, \Delta_4 + 1, \Delta_5 + 1).$$

However, some of these terms dominate:

- $\Delta_4 > 1$, $\Delta_3 < \Delta_2$, $\Delta_5 < \Delta_4$ always.
- When $X_m < \Delta_1$, then $\Delta_5 < \Delta_2$, $\Delta_4 < \Delta_2$, $\Delta_5 < \Delta_3$, but Δ_3 and Δ_4 are unrelated.
- When $X_m > \Delta_1$, then $\Delta_2 < \Delta_4$ and $\Delta_3 < \Delta_5$, but Δ_2 and Δ_5 are unrelated, and Δ_3 and Δ_4 are unrelated.

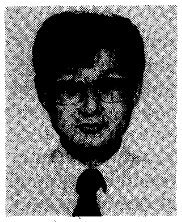
Thus, since $X_m > \Delta_1$ in Ord(7), this expression can be simplified to

$$RR \geq \Delta_4 + 1,$$

as shown in Table II.

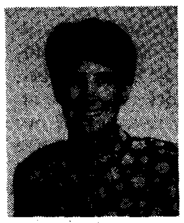
References

[1] U.S. Department of Defense, "Ada Programming Language," 1983. ANSI/MIL-STD-1815A-1983.
[2] INMOS Ltd., *Occam Programming Manual*. Englewood Cliffs, NJ: Prentice-Hall Int., 1984.
[3] J. Gray, "Notes on data base operating systems," in *Operating Systems: An Advanced Course,* R. Bayer, R. Graham, and G. Seegmuller, Eds. New York: Springer-Verlag, 1979, pp. 393–481.
[4] I. Lee and S. Davidson, "Adding time to synchronous process communications," *IEEE Trans. Comput.*, pp. 941-948, Aug. 1987.
[5] R. A. Volz and R. N. Mudge, "Timing issues in the distributed execution of Ada programs," *IEEE Trans. Comput.*, vol. C-36, pp. 449–459, Apr. 1987.
[6] K. Lin, S. Natarajan, and J. Liu, "Imprecise results: Utilizing partial computations in real-time systems," in *Proc. Real-Time Syst. Symp.*, Dec. 1987, pp. 210–217.
[7] S. Davidson and A. Watters, "Partial computation in real-time database systems," in *Workshop Real-Time Oper. Syst.*, May 1988.
[8] I. Lee and V. Gehlot, "Language constructs for distributed real-time programming," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 1985.
[9] J. W. Wong, "Distribution of end-to-end delay in message-switched networks," *Comput. Networks,* vol. 2, pp. 44–49, 1978.
[10] I. Lee, S. B. Davidson, and V. Wolfe, "Motivating time as a first class entity," Tech. Rep. MS-CIS-87-54, Dep. Comput. Inform. Sci., Univ. of Pennsylvania, 1987.



Insup Lee (S'80–M'83) received the B.S. degree in mathematics from the University of North Carolina, Chapel Hill, in 1977, and the M.S. and Ph.D. degrees in computer science from the University of Wisconsin, Madison, in 1978 and 1983.

He is currently an Associate Professor in the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, where he has been since 1983. His research interests include programming languages and operating systems for distributed real-time computing, specification and verification of time dependent systems and protocols, and interconnection network synthesis algorithms.



Susan B. Davidson (M'83) received the B.A. degree in mathematics from Cornell University, Ithaca, NY, in 1978, and the M.A. and Ph.D. degrees in electrical engineering and computer science from Princeton University, Princeton NJ, in 1980 and 1982.

She is currently an Associate Professor in the Department of Computer and Information Science, University of Pennsylvania, where she has been since 1982. Her research interests include the design and analysis of fault-tolerant distributed systems, database systems and real-time systems, and programming languages for complex object databases and real-time systems.