

Real-Time Synchronization of Interprocess Communications

JOHN H. REIF and PAUL G. SPIRAKIS

Harvard University

This paper considers a fixed (possibly infinite) set of distributed asynchronous processes, which at various times are willing to communicate with each other. Each process has various ports, each of which is used for communication with a distinct neighbor process. Each process can have at most one port open at any time, and its other ports must be closed. Two processes *handshake* over a time interval Δ if their respective ports are open for mutual communication during this interval. Note that the handshake relation is a matching. *Successful communication* requires a handshake of at least one step of each process; during the one-step overlap a message can be transmitted between processes. The problem is to synchronize processes (via a distributed scheduler) so that they can successfully handshake at their will, given that the means of synchronization is some low-level construct that does not guarantee the handshake property if used in an unsophisticated way.

Probabilistic distributed algorithms for synchronizing processes so that they can handshake at will are described. A process is considered to be *tame* over a time interval Δ if its speed varies within certain arbitrarily fixed nonzero bounds. Our synchronization algorithms are shown to have *real-time response*: If a pair of processes are mutually willing to communicate within a time interval Δ of length at least a given constant and the pair are tame on Δ , then they establish communication within Δ with high likelihood (for the worst case behavior of the system), and the expected time for establishment of communication is also constant. Our model and algorithms are applied to solve a large class of real-time resource allocation problems, as well as real-time implementation of the synchronization primitives of Hoare's multiprocessing language CSP.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*distributed networks*; C.2.4 [Computer-Communication Networks]: Distributed Systems—*distributed applications*; D.1.3 [Programming Techniques]: Concurrent Programming; D.2.8 [Software Engineering]: Metrics—*complexity measures; performance measures*; D.3.3 [Programming Languages]: Language Constructs—*concurrent programming structures*; D.4.4 [Operating Systems]: Communications Management—*message sending*; D.4.7 [Operating Systems]: Organization and Design—*distributed systems; real-time systems*; E.4 [Data]: Coding and Information Theory—*formal models of communication*; F.2.0 [Analysis of Algorithms and Problem Complexity]: General; G.3 [Mathematics of Computing]: Probability and Statistics—*probabilistic algorithms*; H.2.4 [Database Management]: Systems—*distributed systems; transaction processing*

General Terms: Algorithms, Designs, Languages, Performance

Additional Key Words and Phrases: Interprocess communication, synchronization, response time, probabilistic choice, handshake protocols, expected time, asynchronous systems, symmetry, communicating sequential processes, randomized algorithms

This work was supported in part by National Science Foundation grants NSF-MCS 82-00269 and NSF-MCS 83-00630, and Office of Naval Research Contract N00014-80-C-0674.

Authors' present addresses: J. H. Reif, Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138; P. G. Spirakis, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012.

Preprint of paper appearing in ACM Journal of Transactions on Programming Languages and Systems, Vol. 6, No. 2, April 1984, pp. 215-238.

1. INTRODUCTION

Recently, Rabin [9, 10], Lehman and Rabin [6], and Francez and Rodeh [4] have proposed probabilistic algorithms for a number of synchronization problems. In this *probabilistic approach* no probabilistic assumptions are made about the system behavior, but the algorithms are allowed to make probabilistic choices. This leads to considerably *simpler algorithms*, perhaps because of the locality of their decisions. It also leads to *shorter proofs*, perhaps because the proofs of the corresponding deterministic algorithms had to consider complex situations that would have very low probability if probabilistic choices were taken, whereas, in proofs of probabilistic algorithms, we only need to consider those simple situations that occur with high probability. The probabilistic approach may also lead to improvement in the efficiency of synchronization algorithms. An improvement in space efficiency is seen in [10]. We demonstrate here that a considerable improvement in time efficiency can be made by probabilistic synchronization.

This paper takes the probabilistic approach to *synchronization of communication in a network* of distributed, asynchronous processes. We are interested in *direct* interprocess communication, rather than packet switching as considered in [15]. Furthermore, we consider *handshake* communication, as in Hoare's communicating sequential processes (CSP), rather than *buffered* communication (which is very easy to implement by message queues).

We describe probabilistic distributed algorithms for synchronizing processes so that they can handshake at will. The means of synchronization are Boolean "flag" variables, each of which can be written by only one process and read by at most one other process. The use of flag variables seems to require the fewest assumptions possible without considering specific systems. Note that we do not use any standard high-level synchronization construct, such as shared variables with a mutual exclusion mechanism. If we did, then we would have to assume an implementation of such a mechanism and there are no real-time implementations of such mechanisms (in fact, there is no bounded time implementation of such mechanisms when processes run on different processors). We hope in the future that our techniques rather than other "standard" but inefficient synchronization mechanisms will be utilized for real-time process synchronization.

We have very few assumptions:

- (1) Tameness is required of a process only during the interval in which it is willing to communicate (if the tameness property is violated during that interval, then there may be lower probability of successful communication); at other times any process may dynamically vary its speed arbitrarily and may even die.
- (2) The processes may be willing to communicate with a time-varying set of processes that are only bounded in number. There are *no* probability assumptions about system behavior.

Previously Schwarz [14] proposed a deterministic synchronization algorithm for implementing CSP [5] on a fixed acyclic distributed network. Also Lynch [8] gave a related algorithm for resource synchronization problems. Both algorithms are considerably less time efficient than our proposed algorithm (for specific comparison of time performance, see Section 2.5). Francez and Rodeh [4] have

also proposed a probabilistic solution to the synchronization of communication but have not considered the time efficiency of their solution.

Our paper is organized as follows: In Section 2 we present a model for distributed communication systems; the model ignores the details of message transmission but gives a precise combinatorial specification (by time-varying graphs) of the communication synchronization problem. This model also allows a precise definition of the relevant complexity measures of synchronization algorithms, such as response time. In Section 3 we present our real-time synchronization algorithms, and in Section 4 we prove various properties of the synchronization algorithms that must hold with certainty, regardless of probabilistic choice. Sections 5 and 6 give a probabilistic analysis of the performance of our algorithms. We feel the term *real time* is merited, since the actual time needed for establishment of communication is upper bounded by a constant with overwhelming probability. Furthermore, violations of this property occur with vanishingly low likelihood. We have made considerable effort in the design of our synchronization algorithms to improve their expected time performance. Nevertheless, our algorithms are very simple in conception and practice. The Appendix provides a real-time implementation of the CSP. And in [13] we present a further application: a real-time resource-granting system. We feel these applications demonstrate the broad applicability of our synchronization algorithms.

2. OUR MODEL FOR A DISTRIBUTED COMMUNICATION SYSTEM (DCS) AND ITS COMPLEXITY MEASURES

Let $\Pi = \{1, 2, \dots\}$ be a fixed (possibly infinite) collection of *processes*. We assume a (global) *time* t on the nonnegative real line $[0, \infty]$, whereby events of the system are totally ordered. The processes of Π are *asynchronous*; their speeds may dynamically vary arbitrarily over time and may even be 0. (Thus we allow processes to die). The processes have no access to any global clock giving the time.

We assume that the effect of a read or write is instantaneous and that these events occur at distinct time instants, so there are never any read/write conflicts. In general, a *step* of a process is a finite time interval Δ in which a single instruction is instantaneously executed at the last moment of Δ .

We also assume a global oracle \mathcal{A} , which directs the willingness of processes to communicate with each other. (Note that, in applications of our distributed communication system (DCS) occurring in practice, no such oracle exists, but instead each process is running some program that requires, from time to time, communication with other processes. An implementation of the DCS synchronizes this communication. The oracle \mathcal{A} is utilized as an artificial device for specifying worst case situations of our system, in which communications are required by \mathcal{A} to be made at times most difficult for our implementation.)

Intuitively, each process i wishes at various times to communicate with processes in $\Pi - \{i\}$. All communication required by the oracle is implemented by i rather than by a global centralized synchronization mechanism. Thus systemwide communication is implemented by a distributed scheduler, the processes of Π .

The formal model *DCS* described below has been designed with as few assumptions as possible and to be as general as possible. We are not concerned with the *values of the messages* communicated between the processes but, instead, with simply the *establishment of communication*. This allows us to avoid any message-system-dependent assumptions that may vary for any given application.

We now introduce some graphs to describe the DCS model precisely. The graphs allow us to state the synchronization problems precisely as combinatorial problems on time-varying graphs. We give an intuitive description of the importance of these graphs as they are defined.

Let the *connections graph* $H = (\Pi, E)$ be a (possibly infinite) undirected graph with vertex set Π and undirected edge set $E \subseteq (\Pi \times \Pi) - \{(i, i) \mid i \in \Pi\}$. Then $\{i, j\} \in E$ denotes that $i \in \Pi$ is *physically able to communicate* with $j \in \Pi - \{i\}$ (see Figure 1a). H is fixed for all time and can be considered to be essentially the hardware connections between processes of Π . We assume that H has finite valence (i.e., only a finite number of processes are connected to any given process $i \in \Pi$).

For each time $t \geq 0$, the *willingness digraph* $G_t = (\Pi, \rightarrow_t)$ is a possibly infinite digraph with vertices Π and directed edges given by relation $\rightarrow_t \subseteq \Pi \times \Pi$ (see Figure 1b). Then $i \rightarrow_t j$ denotes that $i \in \Pi$ is *willing to communicate* with $j \in \Pi - \{i\}$ at time t . In that sense we say that i is the *source* and j is the *target*. We require that $i \rightarrow_t j$ imply $\{i, j\} \in E$, and so i is willing to communicate only with processes with which i is able to communicate. Also, let $i \leftrightarrow_t j$ iff both $i \rightarrow_t j$ and $j \rightarrow_t i$. We use \rightarrow_Δ and \leftrightarrow_Δ to denote that the willingness to communicate holds over time intervals. For each time interval Δ on $(0, \infty)$, let $i \rightarrow_\Delta j$ if $i \rightarrow_t j$ for all $t \in \Delta$ and let $i \leftrightarrow_\Delta j$ if both $i \rightarrow_\Delta j$ and $j \rightarrow_\Delta i$. The edges of G_t departing from $i \in \Pi$ are assumed to be stored locally at i in the form of a variable set E_i , which, at time t , contains the names of the targets of i . E_i is specified by the oracle and known and read only by i .

In the following we assume that there exists a given fixed integer constant $v > 0$ such that $\forall i \in \Pi, \forall t \geq 0$, the *outdegree* of i in G_t (i.e., the cardinality of $\{j \mid i \rightarrow_t j\}$) is bounded above by v .

Assumption A1. Two-way communication between any two processes $i, j \in \Pi$, requires only one step of i and j . (Thus i, j are assumed to communicate in short "bursts.")

2.1 Implementation of a DCS

An *implementation of a DCS* assigns a fixed program to each of the processes of Π . The implementation is *symmetric* if the programs are independent of the position of i in the connections graph H .

For each $i, j \in \Pi$, such that $\{i, j\} \in E$, we have a *communication port flag* $\text{PORT}_{i,j}$ (written only by process i), which is 1 at time $t \geq 0$ if i has opened its port for communication with j at t , and 0 otherwise (indicating that the communication port from i to j is closed at t). We assume that two-way communication between i, j is possible at any time that both $\text{PORT}_{i,j}$ and $\text{PORT}_{j,i}$ are simultaneously 1, but we make no particular assumptions (beyond A1 and A2 (below)) about this communication.

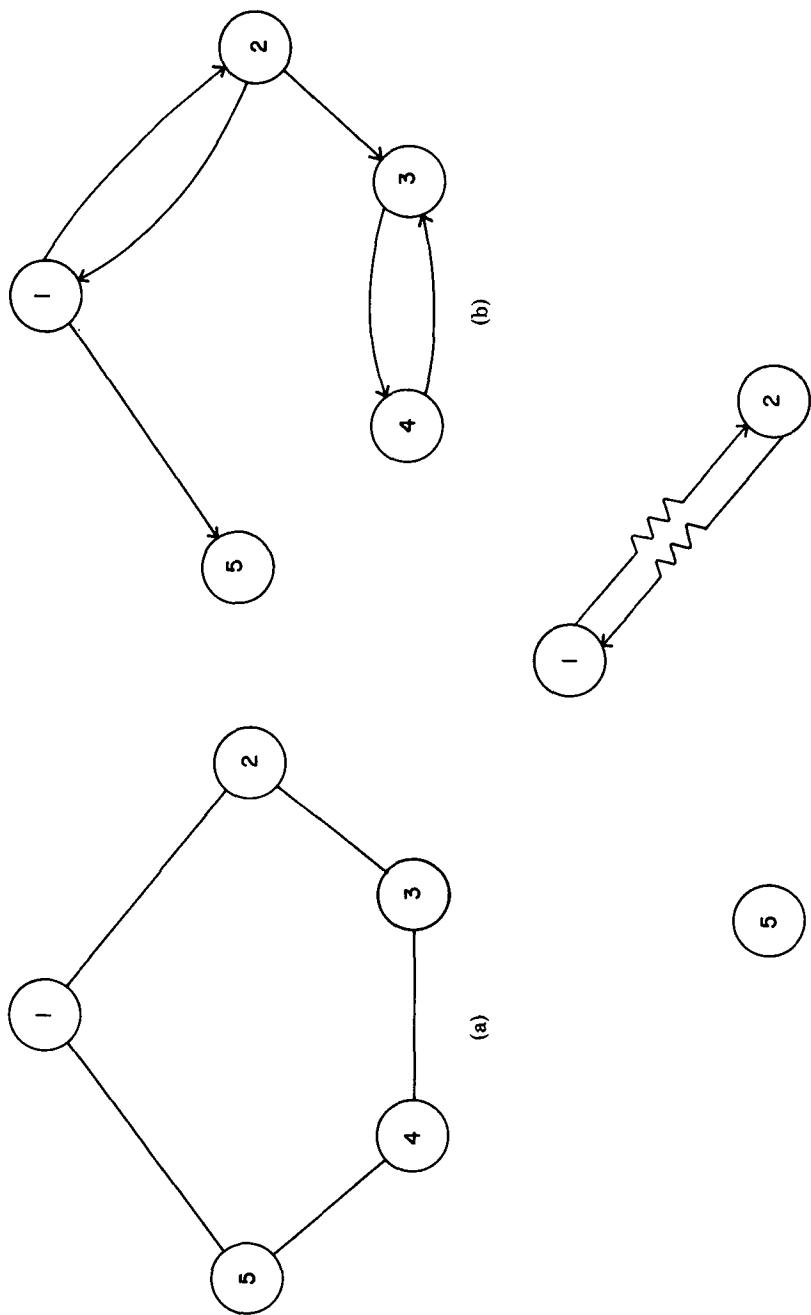


Fig. 1. (a) The connections graph H in the case of a ring network of five processes. (b) The willingness digraph G_r . (c) The open-channel graph M_r .

Let $i \rightsquigarrow_t j$ denote that $\text{PORT}_{i,j} = 1$ at time t . For each $t \geq 0$ our implementation defines a (possibly infinite) directed graph M_t with vertices Π and directed edges given by the relation $\rightsquigarrow_t \subseteq \Pi \times \Pi$. Let $i \leftrightarrow_t j$ if both $i \rightsquigarrow_t j$ and $j \rightsquigarrow_t i$. If $i \rightsquigarrow_t j$, then we say i has *opened communication* with $j \in \Pi - \{i\}$ at t . If $i \leftrightarrow_t j$, then we say i, j achieve *mutual communication* at time t . Also, we extend the notation to intervals Δ on $(0, \infty)$ as for G_t .

Assumption A2. If $i \rightarrow_{t_1} j$ and not $i \rightarrow_{t_2} j$, $t_2 > t_1$, then $i \leftrightarrow_{\Delta} j$ for some $\Delta \in [t_1, t_2]$, where Δ contains at least one step of each i and j (i.e., the oracle \mathcal{A} can withdraw the willingness of i to communicate with j only after communication between i and j has been established and completed).

Note. In practice, assumption A2 can be easily circumvented. Suppose a process i is initially willing to communicate with process j , but later decides that it is no longer interested in communicating with j , before mutual communication has been achieved. By assumption A2, process i must, nevertheless, not withdraw willingness to communicate until the implementation has achieved mutual communication between i and j . However, at this time a null value can simply be sent.

An implementation is *proper* if it satisfies the following restrictions:

R1 $i \rightsquigarrow_t j$ only if $i \leftrightarrow_t j$.

R2 \leftrightarrow_t is a (partial) matching; if $i \leftrightarrow_t j$, then $\neg i \leftrightarrow_t j'$ for any $j' \in \Pi - \{j\}$.

Note that R1 implies that i opens communication with j only if i, j are simultaneously willing to communicate. R2 implies that i does not communicate with more than one process at a time.

It is standard in the study of combinatorial algorithms to specify the combinatorial problem before giving algorithms for the solution. We have precisely described the problem of determining a DCS implementation as a combinatorial problem on dynamic graphs. Later we shall propose two implementations satisfying both these restrictions. Still another implementation is described in [13].

2.2 Global State of the DCS

For each $t \geq 0$, let R_t be a mapping from Π to the nonnegative reals giving the speed of each process of Π at time t . We assume the speed schedule $R = \{R_t \mid t \geq 0\}$ is chosen by an adverse oracle \mathcal{A} (possibly our scheduler's worst "enemy") a priori (at time $t = 0$). Also, we assume that, for each $t \geq 0$, \mathcal{A} chooses for the processes of Π the willingness digraph G_t at time t . Thus G_t may vary dynamically in time, depending on the choices of the oracle \mathcal{A} . However, for each $t \geq 0$, the digraph M_t is defined by the processes of Π , which attempt a distributed synchronization of the DCS, depending on our given implementation. In addition, we allow the processes of Π to make independent probabilistic choices.

Let L_t , the *luck up to time t* , be the probabilistic choices made by the processes of Π up to time t . Then, the *global system state at time t* is given by

$$\Sigma_t = \langle R_t, G_t, M_t, L_t, t \rangle$$

and the *global history* up to time t is

$$\Gamma_t = \{\Sigma_{t'} \mid 0 \leq t' \leq t\}.$$

Thus, we have a probabilistic multiplayer game of incomplete information, where the omnipotent oracle \mathcal{A} plays against the team of processes of Π , which have only incomplete information on the current state of the system. We wish measures of the success of the processes of Π .

2.3 Time Complexity of a DCS Implementation

A process *step* consists of either an assignment of a variable, a test, a logical or arithmetic operator, or a no-op.

Let process i be *tame* on an interval Δ , if for any interval $\Delta' \in [0, \infty)$, if Δ' intersects Δ and Δ' is a single step of process i , than $|\Delta'| \in [r_{\min}, r_{\max}]$, where r_{\min}, r_{\max} are fixed real constants and $0 < r_{\min} \leq r_{\max}$. (Without loss of generality we assume that r_{\max}/r_{\min} is an integer.)

We shall *not* assume that processes are tame at all times. Our DCS implementation will be proper regardless of whether processes are tame as long as their speeds are nonzero.

Let processes i, j have *successful communication* at interval Δ if $i \leftrightarrow_{\Delta} j$ and Δ contains at least one step of both i and j . We say Δ is a *response interval* for processes i, j if Δ is a maximal time interval such that

- (1) $i \leftrightarrow_{\Delta} j$,
- (2) i, j are both tame on Δ , and
- (3) i, j have successful communication at most just at the end of Δ , if at all.

Note that if there is successful communication during an interval Δ' within Δ , then, by (3), Δ' is a suffix of Δ . Note also that since Δ is maximal, either i, j were not mutually willing to communicate immediately before Δ , or Δ begins at time 0, or the instant immediately before Δ is the end of a previous response interval.

Let a *communication request* be $R = (t, i, j)$ such that t is the starting instance of a response interval for processes i, j .

Note that there is a unique communication request associated with each response interval.

Let the *response time* of a DCS implementation, for any oracle \mathcal{A} and communication request R , be the random variable $\tau_{\mathcal{A}, R}$, giving the length of the response interval associated with R . Let $\bar{\tau} = \max\{\text{mean}\{\tau_{\mathcal{A}, R}\} \text{ for each oracle } \mathcal{A} \text{ and communication request } R\}$.

For each ϵ , $0 \leq \epsilon \leq 1$, let the ϵ -*response* $\tau(\epsilon)$ (note that this is a function, not a random variable) be the least upper bound on the set of values of the inverse functions of the cumulative distribution functions of $\tau_{\mathcal{A}, R}$ at $1 - \epsilon$, for all \mathcal{A} and R . Thus, if we have a finite interval Δ , $|\Delta| \geq \tau(\epsilon)$, and any two processes i, j that are tame on Δ , for all oracles \mathcal{A} , $i \leftrightarrow_{\Delta} j$ implies i, j have successful communications some time within Δ with probability $\geq 1 - \epsilon$.

Note that the time response, as defined above for pairs of processes, also holds for communication between sets of processes. Suppose we have finite sets or processes $\Pi_1, \Pi_2 \subseteq \Pi$, such that $|\Pi_1|, |\Pi_2| \leq v$, and for the same interval Δ of length $\geq \tau(\epsilon)$ and for all i in Π_1 and all j in Π_2 , $i \leftrightarrow_{\Delta} j$. Then each process i of Π_1 is guaranteed at least $(1 - \epsilon)^{|\Pi_2|}$ probability of successful communication with all the processes of Π_2 within Δ . This implies a very robust type of fairness.

The DCS implementation is *real time* if for all ϵ , $0 < \epsilon \leq 1$, $\tau(\epsilon)$ is a constant dependent only on v (assumed to be a constant upper bound on the outdegree of

vertices of G_t). Note then that $\bar{\tau}$ is also bounded above by a fixed constant dependent only on v .

2.4 Preferential DCS Implementations

We also consider the cases in which any given process $i \in \Pi$ may assign a *priority* to the processes $j \in \Pi - \{i\}$ that i wishes to communicate with. In the simplest case, which we only consider here, i distinguishes the *first target* of communication $E_i(1)$, with which i prefers to communicate. (Process i may communicate with the other processes of E_i , but i prefers to communicate with $E_i(1)$.)

For each $t \geq 0$, \rightarrow'_t is the relation on $\Pi \times \Pi$ such that $\forall i, j \in \Pi, i \rightarrow'_t j$ iff $E_i(1) = j$ at time t . Also let $i \rightarrow'_\Delta j$ if $i \rightarrow'_t j \forall t \in \Delta$.

We say Δ is a preferential *response interval* for i, j if Δ is a maximal interval such that

- (1) $i \rightarrow'_\Delta j$ and $j \rightarrow_\Delta i$,
- (2) i, j are both tame on Δ , and
- (3) i, j have successful communication at most just at the end of Δ , if at all; that is, if $i \leftrightarrow_{\Delta'} j$, then Δ' is a suffix of Δ .

(Note that only the first process has to distinguish the other as the first target.)

Let a *preferential communication request* $R = (t, i, j)$ be such that t is the starting instant of a preferential response interval for i, j . Note that there is a unique R associated with each response interval. We now define the time complexity of preferential DCS implementations in a similar way: Let the *preferential response time* of a DCS implementation for any oracle \mathcal{A} and preferential communication request R be the random variable $\tau'_{\mathcal{A}, R}$, which gives the length of the preferential response interval associated with R .

Let $\bar{\tau}' = \max\{\text{mean}(\tau'_{\mathcal{A}, R}) \text{ for all oracles } \mathcal{A} \text{ and communication requests } R\}$.

For each $\epsilon, 0 < \epsilon \leq 1$, let the ϵ -preferential response $\tau'(\epsilon)$ be the least upper bound on the set of the values of the inverse functions of the cumulative distribution functions of $\tau'_{\mathcal{A}, R}$, each evaluated at $1 - \epsilon$, for all \mathcal{A} and R .

Thus, if we have a finite interval $\Delta, |\Delta| \geq \tau'(\epsilon)$, and any two processes i, j that are tame on Δ , then for every oracle \mathcal{A} , $(i \rightarrow'_\Delta j \text{ and } j \rightarrow_\Delta i)$ implies i, j have successful communication sometimes within Δ , with probability $\geq 1 - \epsilon$.

The DCS implementation has *real-time preferential response* if for all $\epsilon, 0 < \epsilon \leq 1$, $\tau'(\epsilon)$ is a constant dependent only on v (and not on any parameter of H). Note then that $\bar{\tau}'$ is also bounded above by a constant dependent only on v .

It is useful to observe that, given $\tau'(\epsilon)$, any given process $i \in \Pi$ may determine (with any given probability) whether any process $j \in \Pi - \{i\}$ is willing to communicate with i over a given time interval in which both i, j are tame, given $\{i, j\} \in H$. The same holds if $\tau(\epsilon)$ is given instead of $\tau'(\epsilon)$.

PROPOSITION 2.1. *Let \mathcal{A} be any oracle and Δ be any time interval of finite length $\geq \tau(\epsilon)$ ($\tau'(\epsilon)$ in the case of preferential DCS). Suppose i, j are tame on Δ and $\{i, j\} \in H$. If there is no $t \in \Delta$ such that $i \leftrightarrow_{\Delta-t} j$, then j is not willing to communicate with i at some time within Δ , with probability $\geq 1 - \epsilon$.*

This proposition may be used for *timing out* requests (or preferential requests) to communicate with a specific process.

Note. Suppose we are given a process j , a set of processes $\Pi_1 \subseteq \Pi$, and an interval $\Delta \geq \tau'(\epsilon)$ such that for all $i \in \Pi_1$, $i \rightarrow'_\Delta j$ and $j \rightarrow_\Delta i$. Assume also that all processes in $\Pi_1 \cup \{j\}$ are tame on Δ . Then, for each $i \in \Pi_1$ and for all oracles \mathcal{A} , i, j will have successful communication some time within Δ , with probability $\geq 1 - \epsilon$. Furthermore, if $|\Pi_1| \leq v$, then, for all oracles \mathcal{A} , j will have successful communication with all $i \in \Pi_1$ within Δ , with probability $\geq (1 - \epsilon)^{|\Pi_1|}$.

2.5 Results and Previous Work

The primary results of this paper are

- (1) There is a *proper real-time implementation* of DCS such that
 - (a) the worst case mean response $\bar{\tau}$ is $O(v^2)$;
 - (b) the ϵ -response $\tau(\epsilon)$ is $O(v^2 \log(1/\epsilon))$.
- (2) Also, there is a *real-time preferential implementation* of DCS such that
 - (a) the worst case mean preferential response $\bar{\tau}'$ is $O(v)$;
 - (b) the ϵ -preferential response $\tau'(\epsilon)$ is $O(v \log(1/\epsilon))$.

Our implementations are proper, symmetric, and completely independent of the connection graph H (H may be any finite or infinite graph with finite valence). We allow processes to make probabilistic choices and show that our algorithms have real-time response.

The best previous result is due to Schwarz [14] and is restricted to the case \tilde{H} is finite; its edges can be directed to form a digraph H' , which is acyclic. Let $\chi(H)$ be the minimum vertex coloring of any such H' . Essentially, the technique of Schwarz is to color H' and order the precedence of message transmissions by the coloring. Delays in message transmissions can be as long as $\chi(H)$ since chains of processes (of length $\chi(H)$), in which each process waits for the next to reply, can be formed. So the deterministic DCS implementation of Schwarz [14] has preferential response time τ' lower bounded by $v \cdot \chi(H)$. Note that his implementation is *not real time*, since, in general $\chi(H)$ is of size $|\Pi|$. In contrast, in our implementation, the time-varying willingness digraph is assumed to have bounded outdegree, but we see no way for Schwarz's algorithm to take advantage of this. Also, his DCS implementation is *not symmetric*, since processes are required to know their color in H' .

Also, Lynch [8] gives a solution to a distributed resource allocation problem which in [13] is adopted to yield a DCS implementation with response time $v \cdot \chi(H)$. In [13] we show that a class of generalized resource allocation problems related to those of [8] may be solved in real time by our DCS implementation (with vanishingly small probability of violation of the real-time property).

Francez and Rodeh [4] proposed a probabilistic synchronization algorithm that can be considered to be a DCS implementation. An important difference between our implementation and theirs is that, in our algorithms, in the responding phase, each process responds to all processes with which it is willing to communicate, whereas in [4] only one process is considered at a time. Although Francez and Rodeh make no explicit timing assumptions, they do assume that the setting and resetting of shared variables takes only a negligible time compared with the waiting time of processes, which is a much stronger assumption than ours. The

careful consideration of timing in our paper is crucial to our achievement of real-time response (see also the analysis), and such timing considerations have essentially not been considered in previous papers on synchronization.

3. OUR IMPLEMENTATION OF A DCS

To implement a DCS, we must give an algorithm for each process in Π . We present here two such implementations. Both satisfy restrictions R1 and R2 required by proper implementations, and both are symmetric: Each process has the same algorithm regardless of its position in the graph H . Processes have Algorithm 1 in our “nonpreferential” implementation, and Algorithm 2 in our “preferential” implementation. We show in Section 4 that both implementations have real-time response.

Each program variable X of the system may be written by exactly one process $i \in \Pi$, and either X is read by only one other process $j \in \Pi - \{i\}$ (in this case X is a *flag* from i to j) or X is *local* to i (X is read only by i).

The following description of the DCS implementations is given top-down, with a high-level specification of the algorithms given first and then a specification of the procedures ASK, RESPOND, which they call. (The procedures ASK, RESPOND utilize numerous flag variables that are irrelevant to the overall understanding of our algorithms.) Also, before giving the formal specifications of any algorithm or procedure, we provide an informal description of its actions. The actual formal algorithms have been carefully written to satisfy certain timing restrictions required by our analysis to achieve real-time response.

In both algorithms each process repeatedly throws a fair coin and then executes a *phase*. Each phase is either *asking* or *responding* and is chosen by the coin throw with independent probability $\frac{1}{2}$. Then, on any step of a process, the probability of its being in the asking phase is exactly $\frac{1}{2}$. Furthermore, the duration of any step is, by assumption, fixed by the oracle \mathcal{A} *a priori*. This implies that, on the average, each process is in the asking phase half of the time.

3.1 Informal Description of the Nonpreferential Algorithm 1

In a responding phase, process i repeats a loop m times, where $m = (v + 3) \cdot r_{\max}/r_{\min} + 1$. On each iteration of the loop, process i chooses at random a process j from the processes with which i is willing to communicate and executes procedure $\text{RESPOND}_i(j)$. This procedure takes constant c_R number of steps. During these steps process i reads a flag to determine whether j has recently been willing to talk to i and then sets a flag to later verify that j pays attention to i . These verifications are done by handshakes. (A *handshake* is the use of Boolean flags to verify exchange of a single bit of information.) If so, processes i and j synchronize their steps and then both open communication with each other. In either case, i repeats the loop until the corresponding phase finishes.

In an asking phase, process i chooses *only once* at random a process j with which i is willing to communicate, and then i executes procedure $\text{ASK}_i(j)$. This procedure takes $c_A = c_R \cdot m$ steps (so that both phases take exactly the same number of steps. As a consequence, process i is in each phase, on the average, half of the time. This is important to the analysis). During procedure $\text{ASK}_i(j)$, process i raises a flag to show to j that it is currently willing to communicate

with j , and then pays attention to j for a limited number of steps to test whether j is responding to the attempt and wants to proceed in communication. If it does, then processes i and j synchronize their steps and then both open communication to each other. If not, then i finishes its current phase by setting its flags to 0.

3.2 Informal Description for the Preferential Algorithm 2

Each process i executes forever the following loop:

It chooses with probability $\frac{1}{2}$ to execute a respond phase or a modified ask phase. The respond phase is identical to that of Algorithm 1. However, in the modified ask phase, process i chooses the distinguished first process $E_i(1)$ as the process to which it will apply the procedure ASK_i .

3.3 Formal Definitions of Algorithms 1 and 2

We now give Algorithms 1 and 2 in full detail.

Algorithm 1 (Nonpreferential Implementation)

Program for process $i \in \Pi$

```
INITIALIZEi( );
While TRUE do
  begin
    L2: CHOOSE a random  $b \in \{0, 1\}$ 
    if  $b = 0$  then
      begin
        comment: respond phase
        L3: for  $x = 1$  to  $m$  do
          begin
            CHOOSE at random  $j \in E_i$ 
            RESPONDi( $j$ );
          end
        end
      else
        begin
          comment: ask phase
          L4: CHOOSE at random  $j \in E_i$ 
          ASKi( $j$ )
        end
      end
    od
```

Algorithm 2 (Preferential Implementation)

Program for process $i \in \Pi$

```
INITIALIZEi( )
while TRUE do
  begin
    L1: CHOOSE a random  $b \in \{0, 1\}$ 
    if  $b = 0$  then
      begin
        comment: respond phase
        L3: for  $x = 1$  to  $m$  do
          begin
            CHOOSE a random  $j \in E_i$ 
            RESPONDi( $j$ )
          end
      end
```

```

    end
else
begin
comment: ask phase
L4: ASKi(Ei(1))
end
end
od

```

3.4 Intuitive Description of the Procedures ASK, RESPOND

The procedures ASK_i, RESPOND_i are utilized by both algorithms.

For each $i, j \in \Pi$, such that $\{i, j\} \in H$ there are three *flags* (Boolean variables), Q_{ij} , A_{ij} , B_{ij} , which are written only by i and read only by j .

- (1) Flag Q_{ij} : Just before each phase, $Q_{ij} = 0$. Then i asks j by setting Q_{ij} to 1 in the ask phase. Q_{ij} is reset to 0 before the end of the ask phase.
- (2) Flag A_{ij} : Just before each phase, $A_{ij} = 0$. If i is in the responding phase and detects $Q_{ji} = 1$ (indicating j "asks" i), then i answers j by setting $A_{ij} = 1$. Before the end of the answer phase, i resets A_{ij} to 0.
- (3) Flag B_{ij} : This variable is set to 1 by i only during the *watching window*, which is the interval when i is in the asking phase and is watching for an answer ($A_{ij} = 1$) from j . At all other times B_{ij} is set to 0 to indicate i is *blind* to answers by j .

Another flag $PORT_{ij}$ is utilized by the low-level procedure OPEN-COM to specify the state of the communication port from i to j . As defined in Section 2, $i \leftrightarrow_t j$ iff $PORT_{ij} = 1$ at time t . (OPEN-COM is called by ASK_i and RESPOND_i as the final act in a successful communication attempt.)

If process i executes ASK_i, then it first sets a flag variable $Q_{i,j}$ to 1 (to indicate to j that it asks) and sets another flag $B_{i,j}$ to 1 (to indicate to j that it pays attention to it; i.e., i is not blind to answers by j). It keeps these flags raised for at most a constant number c_B steps, and during these steps it continuously examines the flag $A_{j,i}$ (the answer flag of j). If the interval finishes with no answer from target, then i first sets $B_{i,j}$ to 0 (to show that it stops paying attention to j) and then it sets $Q_{i,j}$ to 0 to drop the question. This order of actions guarantees that process j will interpret correctly what it sees from the flags of i .

If i gets an answer from j (i.e., if $A_{j,i}$ is set to 1) during the (previously discussed) c_B steps, then i first sets $Q_{i,j}$ to 0 (but keeps $B_{i,j}$ to its current value to indicate that it continues to pay attention to j). Process i waits until j also zeros its flag $A_{i,j}$ and then process i calls OPEN-COM_i(j) immediately. As the analysis shows, the events leading to this call guarantee that communication is achieved between i and j during the execution of OPEN-COM, assuming i and j are tame. We do not use a handshake protocol within OPEN-COM since certain technical constraints (see Lemma 4.6) of our analysis would be violated (i.e., if i were tame but j were not, i would unnecessarily delay in OPEN-COM and this would cause problems to communication between i and other tame processes). At the end of OPEN-COM, i sets $B_{i,j}$ to 0 (showing that it stops paying attention to j) and exits procedure ASK_i.

If process i executes procedure RESPOND_i (asker), then it first examines whether $Q_{\text{asker},i}$ is 1 (i.e., whether asker is interested in communicating with i). If so, then i sets $A_{i,\text{asker}}$ to 1 and waits until process asker zeros its question flag (this is the handshake technique). When this happens, then i tests $B_{\text{asker},i}$ to see whether process asker still pays attention to i . If not, then i zeros its answer flag $A_{i,\text{asker}}$ and exits. Else, i knows that asker waits for step synchronization and communication. So i zeros its flag $A_{i,\text{asker}}$ and calls OPEN-COM_i (asker). The analysis shows that the events leading to this call guarantee that communication will be achieved.

We now introduce some terminology and then develop the algorithms in full detail.

A process i is in the *asking mode* when it executes procedure ASK_i , and it is in the *responding mode* when it executes the procedure RESPOND_i . If i is executing $\text{ASK}(j)$ and $B_{ij} = 1$, then i is in a *watching window* for process j , else i is *blind* with respect to j . We say i is *answered by j* if i is in its watching window for j and i exits loop A3 of procedure ASK_i with $a = 1$. A *phase* of the algorithm consists of the steps between random choices of the variable $b \in \{0, 1\}$. If $b = 0$, the process is in a *responding phase*, else it is in an *asking phase*.

We have not elaborately commented on our procedures because of the extensive informal description preceding them.

The variables of process i are initialized as follows:

```
INITIALIZE $_i$ ( );
begin
  for all  $j \in \Pi$  such that  $\{i, j\} \in H$  do
    begin
       $Q_{ij} \leftarrow 0$ 
       $A_{ij} \leftarrow 0$ 
       $B_{ij} \leftarrow 0$ 
       $\text{PORT}_{ij} \leftarrow 0$ 
    end
  end
```

In the following two procedures, we assume a register CURSTEP , which gives the current number of the steps executed by process i since it was last zeroed. (CURSTEP is assumed here only as a convenience; it is clear that we could substitute instead a new variable that is incremented on every step of the original algorithm.)

We have made extensive use of time-outs to guarantee that the number of steps for the execution of procedures RESPOND , ASK are always exactly the same for each. (This is crucial to our proof of real-time response.)

We define the parameters appearing in the procedures:

Let $c_R = 7 + (12 + 4(r_{\max}/r_{\min}))(r_{\max}/r_{\min})$; this will be precisely the number of steps always required by procedure RESPOND (see justification in Lemma 4.3). Let $m = (v + 3)(r_{\max}/r_{\min}) + 1$. Let $c_A = c_R \cdot m$; this will be the number of steps required by procedure ASK . Let $c_B = c_A - c_R$; this is the number of steps required for a watching window. Let $c_P = 2 + 3(r_{\max}/r_{\min})$; this is the number of steps required in procedure OPEN-COM . Let $c_D = c_A - c_P - 2$ and $c_E = c_R - 7$. These parameters are used to time-out the execution of various loops in our algorithms.

```

PROCEDURE ASKi(target)
local  $a$ 
begin
  A1: CURSTEP  $\leftarrow 0$ 
  A2:  $Q_{i,\text{target}} \leftarrow 1$ 
       $a \leftarrow 0$ 
       $B_{i,\text{target}} \leftarrow 1$ 
      comment: Begin watching window for target
  A3: while CURSTEP  $< c_B$  and  $a = 0$  do  $a \leftarrow A_{\text{target},i}$ 
      if CURSTEP  $\geq c_B$  and  $a = 0$  then  $B_{i,\text{target}} \leftarrow 0$ 
       $Q_{i,\text{target}} \leftarrow 0$ 
      if  $a = 1$  then
        begin
          A4: while ( $A_{\text{target},i} = 1$  and CURSTEP  $< c_D$ ) do  $a \leftarrow A_{\text{target},i}$ 
          A5: if  $a = 0$  and CURSTEP  $\leq c_D$  then OPEN-COMi(target)
              end
              comment: End watching window for target
               $B_{i,\text{target}} \leftarrow 0$ 
              while CURSTEP  $< c_A$  do a nonoperative step.
        end
      end
  end

PROCEDURE RESPONDi(asker)
local  $q$ 
begin CURSTEP  $\leftarrow 0$ 
   $q \leftarrow Q_{\text{asker},i}$ 
  B1: if  $q = 1$  then
    begin
      Ai,asker  $\leftarrow 1$ 
      B2: while (CURSTEP  $< c_E$  and  $q = 1$ ) do  $q \leftarrow Q_{\text{asker},i}$ 
           $q \leftarrow (q \text{ or } B_{i,\text{asker}} = 1 \text{ or } \text{CURSTEP} > c_E)$ 
      B3:  $A_{i,\text{asker}} \leftarrow 0$ 
          if  $\neg q$  then B4: OPEN-COMi(asker)
      B5: while CURSTEP  $< c_R$  do a nonoperative step
    end
  end

PROCEDURE OPEN-COMi(j)
begin
  PORTij  $\leftarrow 1$ 
  do  $c_P - 2$  nonoperative steps
  PORTij  $\leftarrow 0$ 
end

```

4. PROPERTIES OF THE ALGORITHMS THAT HOLD WITH CERTAINTY

4.1 Correctness Properties of the Algorithms That Hold with Certainty

Our algorithms are probabilistic and therefore some of their properties (such as response time) only hold with a *certain probability*, and not with certainty. A probabilistic analysis of these properties is given in the next sections. However, in this section we prove properties of the algorithms that hold with *certainty*, regardless of probabilistic choice. We show restrictions R1, R2 are satisfied by our implementations, and thus they are proper. (Of course, we assume either that all the processes in Π execute Algorithm 1, or that they all execute Algorithm 2.)

LEMMA 4.1. *For both algorithms,*

$$i \rightsquigarrow_t j \quad \text{only if} \quad i \leftrightarrow_t j.$$

PROOF. Process i calls OPEN-COM $_i(j)$ and opens its channel to j only if either (a) i was executing an asking phase and exited the loop A3 with $a = 1$ or (b) i was executing a respond phase and exited the busy wait B2 with $B_{j,i} = 1$. In both cases, i was willing to communicate with j at the start of the execution of its phase, since i only asks (or responds to) processes it is willing to communicate with. So, $i \rightarrow_{t'} j$ where t' was the time of start of i 's phase. By assumption (A2) then, $i \rightarrow_t j$.

In case (a), $a = 1$ means that j responded by setting $A_{j,i}$ to 1 to i 's question. So $j \rightarrow_{t''} i$ for some $t'' < t$ and by assumption (A2), $j \rightarrow_t i$.

In case (b), j was the process setting $Q_{j,i}$ to 1 at the beginning of i 's phase. Hence $j \rightarrow_{t'} i$ and, by (A2), $j \leftrightarrow_t i$.

In both cases, $i \rightsquigarrow_{t'} j$ implies $i \leftrightarrow_t j$. \square

LEMMA 4.2. *For both algorithms, $\rightsquigarrow_{\Delta}$ is a partial matching.*

PROOF. Since each process opens communication to at most one process each time (this is so since the programs in both algorithms are sequential and each neighbor is asked or responded to separately), the relation $\rightsquigarrow_{\Delta}$ is one to one. Hence $\rightsquigarrow_{\Delta}$ cannot be more than a matching. \square

COROLLARY 4.1. *Both algorithms give a proper implementation of DCS.*

4.2 Timing Lemmas That Hold with Certainty

Timing is an important aspect of our algorithms. The following lemmas are essential, but somewhat tedious to prove.

LEMMA 4.3. *Assume i, j are tame. For both algorithms, if i is answered by j , then i, j have successful communication, within $(12 + 4(r_{\max}/r_{\min}))$ steps of the slower of i, j , from the time i exits loop A3.*

PROOF. If i exits the A3 loop with $a = 1$, then (since no process but j can assign to $A_{j,i}$) at the same time j must be executing RESPOND $_j(i)$ at the B2 loop. Process i will arrive at A4 within 4 of its steps and will have by then set Q_{ij} to 0. These four steps of i correspond to at most four r_{\max}/r_{\min} steps of j , during which j will have exited the B2 loop. Also at this time, the assumption that i exits the loop A3 with $a = 1$ implies that $B_{ij} = 1$. So j will arrive at B3 and set $A_{j,i}$ to 0 in at most four of its steps from the time it exited the B2 loop. Within r_{\max}/r_{\min} steps of i , process i exits the A4 loop. Then, within two of i 's steps i will call OPEN-COM $_i(j)$ and within one of j 's steps j will call OPEN-COM $_j(i)$. Note that both i, j will set their respective port flags PORT $_{ij}$, PORT $_{ji}$ to 1 within one step of the slower process (or, within at most r_{\max}/r_{\min} steps of the faster). They keep their ports open for $c_P - 2 = 3(r_{\max}/r_{\min})$ steps each. This implies that both processes will overlap for at least $2(r_{\max}/r_{\min}) \cdot r_{\min} = 2r_{\max}$ time, guaranteeing at least one step overlap of both processes. Thus i, j have successful communication. Note that OPEN-COM takes c_P steps. Counting steps of i plus those of j in nonoverlapping time intervals, we have a total of $4 + 4 + r_{\max}/r_{\min} + 2 + c_P = 12 + 4(r_{\max}/r_{\min})$, which is certainly an upper bound to the steps of the slowest of the two processes. \square

LEMMA 4.4. *For both algorithms, if i, j are tame on Δ' and $i \rightsquigarrow_{\Delta'} j$ for a maximal interval Δ' , then Δ' contains at least a step of both i and j and $|\Delta'| = O(1)$. (This ensures that Δ' is just long enough for i, j to communicate.)*

PROOF. The only sequence of events leading to this is the sequence in which one of i, j is in its watching window for the other and is answered by the other. By Lemma 4.3, then, Δ' contains a step of both i, j . Since Δ' is not greater than c_P steps of either process, $|\Delta'| \leq c_P r_{\max} = 2r_{\max} + 3(r_{\max}^2/r_{\min})$. \square

LEMMA 4.5. *For both algorithms, if i, j are tame on Δ and $i \rightsquigarrow_{\Delta} j$ for a maximal interval Δ , then $i \rightsquigarrow_{\Delta'} j$ for some $\Delta' \subseteq \Delta$. Furthermore, i, j have successful communication during Δ' . (I.e., a tame process never opens its channel to another tame process without communicating with it.)*

PROOF. The only sequence of events leading to $i \rightsquigarrow_{\Delta} j$ is the sequence in which one of i, j was in its watching window for the other and is answered by the other. By Lemma 4.3, $\exists \Delta' \subseteq \Delta$ such that i, j have successful communication during Δ' . \square

In the following lemma, we need not necessarily assume that i is tame.

LEMMA 4.6. *If $i \in \Pi$ executes procedure ASK, then precisely c_A steps of i are required for the execution of this procedure. Execution of RESPOND by i requires precisely c_R steps of i . Also, each phase of either Algorithm 1 or Algorithm 2 requires exactly $c_A + 2$ steps.*

PROOF. By observation of time-outs within the procedures ASK and RESPOND and by the definition of $c_A = m \cdot c_R$. \square

Let $c = m/v \cdot c_R$. Then $c \cdot v$ is the number of steps required for each phase.

COROLLARY 4.2. *The time required for each phase is upper bounded by $c v r_{\max}$.*

5. PROBABILISTIC ANALYSIS OF THE RESPONSE TIME OF THE ALGORITHMS

Intuitively, in both algorithms, the ASK or Respond phase takes $O(v)$ time each. In the worst case of the nonpreferential algorithm, it requires $O(v)$ expected executions of the ASK phase to choose any given willing neighbor, if the set of willing neighbors is $O(v)$. Given that a given neighbor is chosen and is willing, communication will be achieved with probability bounded below a constant. Hence, we expect the average time of response of the nonpreferential algorithm to be $O(v^2)$.

On the other hand, in the asking phase of the preferential algorithm we ask a specific neighbor and we have a constant probability to communicate with him, if he is willing. Thus, the expected total number of phases will be $O(1)$ and so the expected response time of the preferential algorithm will be $O(v)$ in the worst case.

A formal analysis follows:

By Corollary 4.2, cv is the total number of steps of the asking or responding phase. Fix throughout this section I to be an interval, starting at time t_0 , of length at least four phases, so $|I| \geq 4cv r_{\max}$. Let Γ_{t_0} be the global system history up to t_0 and let \mathcal{A} be a fixed oracle. Note that (\mathcal{A}, Γ_t) essentially specifies everything of the system's immediate future except "luck" L_t for $t' > t_0$. Consider

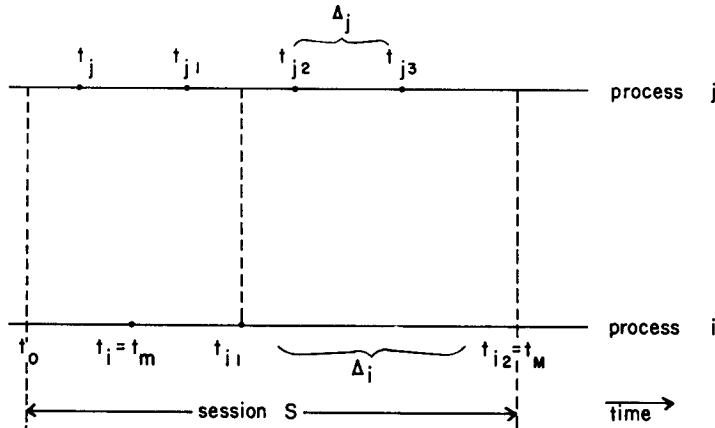


Fig. 2. A session S of i, j and one of the possible orderings of events: * indicates the start of a phase; Δ_j is a phase of j ; Δ_i is a phase of i .

two processes i, j such that $\{i, j\}$ is an edge of H and $i \leftrightarrow_I j$, i, j tame on I . Let

$$t_j = \min\{t > t_0 \mid j \text{ does a phase selection at } t\}$$

and

$$t_i = \min\{t > t_0 \mid i \text{ does a phase selection at } t\}.$$

Let $t_m = \max(t_i, t_j)$ and let t_{i1}, t_{i2} (and t_{j1}, t_{j2}) be the next two phase selections of i (and j , respectively) after t_m such that

$$t_m < t_{i1} < t_{i2} \quad \text{and} \quad t_m < t_{j1} < t_{j2}.$$

Let

$$t_M = \max(t_{i2}, t_{j2}).$$

The interval $(t_0, t_M]$ is called a *session* S of processes i, j (see Fig. 2). Note that a session has three or fewer phases of one of the processes i, j and hence its length is $\leq 3c\tau_{\max}$.

Let $\sigma_{ij}(\mathcal{A}, \Gamma_{t_0})$ be the probability that i, j will establish communication during session $S = (t_0, t_M]$ given $(\mathcal{A}, \Gamma_{t_0})$.

Let \mathcal{L} be the class of oracles \mathcal{A} for which the outdegree d_t is set equal to v for all nodes i in G_t and for all instances t .

PROPOSITION 5.1. *The response time of Algorithm 1 increases with increased requests to communication.*

PROOF. The probability that a specific process is chosen in the ASK or RESPOND phases decreases monotonically with the number of processes to which the process executing ASK or RESPOND is willing to communicate. \square

By Proposition 5.1, the class of oracles \mathcal{L} gives an upper bound in the response time of the system, since adding requests to communicate cannot decrease the response time.

COROLLARY 5.1. *For oracles $\mathcal{A} \in \mathcal{L}$, $\sigma_{ij}(\mathcal{A}, \Gamma_{t_0}) \leq 1/v$ for the nonpreferential Algorithm 1.*

Consider the event $\mathcal{E} = \{\text{process } i \text{ is in the responding phase in the interval } \Delta_i = [t_{i1}, t_{i2}) \text{ and process } j \text{ is in the same asking phase and waiting in a watching window for } i, \text{ for at least } (v/2) \cdot c_R \text{ steps of process } i \text{ during } \Delta_i\}$. Let $x_{ij}(\mathcal{A}, \Gamma_{t_0}, \mathcal{E})$ be the probability that i, j will establish communication during session $S = (t_0, t_M]$, given $(\mathcal{A}, \Gamma_{t_0})$ and event \mathcal{E} .

PROPOSITION 5.2

$$x_{ij}(\mathcal{A}, \Gamma_{t_0}, \mathcal{E}) \cdot \Pr(\mathcal{E} \text{ given } (\mathcal{A}, \Gamma_{t_0})) \leq \sigma_{ij}(\mathcal{A}, \Gamma_{t_0}).$$

PROOF. easy. \square

Note that for both algorithms Proposition 5.3 holds:

PROPOSITION 5.3

$$x_{ij}(\mathcal{A}, \Gamma_{t_0}, \mathcal{E}) \geq 1 - \left(1 - \frac{1}{v}\right)^{v/2} \geq 1 - e^{-1/2},$$

since given \mathcal{E} it is enough for i to select j as the process to answer for at least one of the at least $v/2$ consecutive answering intervals that overlap with the watching window of j .

THEOREM 5.1. *For Algorithm 1, we have*

$$\Pr(\mathcal{E} \text{ given } (\mathcal{A}, \Gamma_{t_0})) \geq \frac{1}{4v}.$$

PROOF. The length of a watching window of j is $c_B = c_R \cdot (m - 1)$ steps of j , which is at least $c_R \cdot (m - 1) \cdot r_{\min}$ time, which is at least

$$c_R \cdot (m - 1) r_{\min} \cdot \frac{1}{r_{\max}} \geq c_R (v + 3)$$

steps of i .

Let $\Delta_i = [t_{i1}, t_{i2})$. The interval Δ_i either contains at least half of a phase Δ_j of j , or half of it is contained in a phase Δ_j of j (see Figure 3). In either case, given that j is asking i during Δ_j and that i is responding during Δ_i , there is an overlap of the watching window of phase Δ_j of j and phase Δ_i of i which contains at least $c_R \cdot v/2$ steps of i . Since both phases Δ_j and Δ_i were selected in the session S after t_0 , the probability of j asking i during Δ_j is $\geq 1/2v$ and the probability of i answering during Δ_i is $\frac{1}{2}$. Our theorem follows by multiplying these probabilities. \square

THEOREM 5.2. *For Algorithm 2, we have*

$$\Pr(\mathcal{E} \text{ given } (\mathcal{A}, \Gamma_{t_0})) \geq \frac{1}{4}.$$

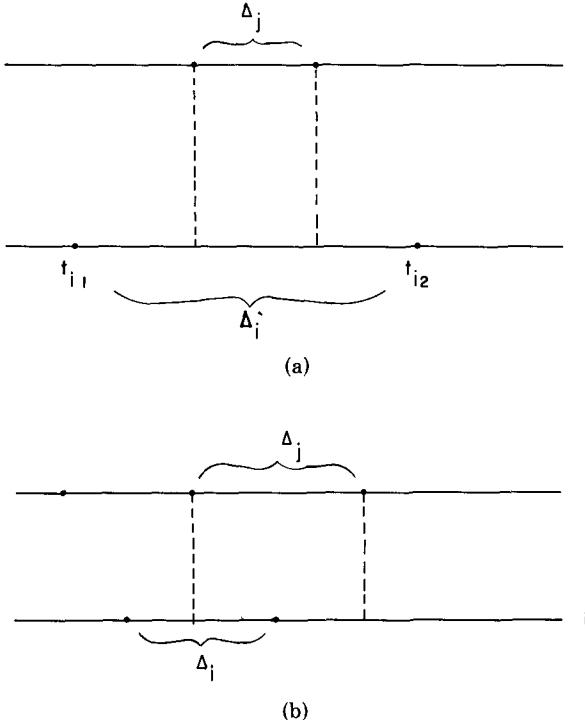


Fig. 3. (a) A case of phase overlap where phase Δ_i of i contains at least half of phase Δ_j of j . (b) Phase Δ_j of j contains at least half of phase Δ_i of i .

PROOF. Same as in 5.1. The only difference is that now j insists on asking i and hence $\Pr(j \text{ asking } i \text{ given } j \text{ in asking phase and preferring } i) = 1$. \square

Let

$$\sigma_{\min} = \frac{1}{4v} (1 - e^{-1/2}), \quad \sigma_{\max} = \frac{1}{v} \quad \text{for Algorithm 1,}$$

$$\sigma_{\min} = \frac{1}{4} (1 - e^{-1/2}), \quad \sigma_{\max} = 1 \quad \text{for Algorithm 2.}$$

THEOREM 5.3

$$0 < \sigma_{\min} \leq \sigma_{ij}(\mathcal{A}, \Gamma_{t_0}) \leq \sigma_{\max} \leq 1.$$

PROOF. By Theorems 5.1 and 5.2, Propositions 5.2 and 5.3, and Corollary 5.1. \square

Note that our lower bounds on $x_{ij}(\mathcal{A}, \Gamma_{t_0}, \mathcal{E})$ and $\Pr(\mathcal{E} \text{ given } (\mathcal{A}, \Gamma_{t_0}))$ do not depend on \mathcal{A} or Γ_{t_0} . This is so because all inequalities hold for any possible speed ratio of processes i, j and because the results of choices of phases for times $t \geq t_m$ in session S do not depend on \mathcal{A} or Γ_{t_0} .

Let $P_{ij}(k | (\mathcal{A}, \Gamma_{t_0}))$ be the probability that it takes exactly k sessions for processes i, j to succeed in establishing communication, given that $i \leftrightarrow_\Delta j$ (or $i \rightarrow'_\Delta j$ and $j \rightarrow_\Delta i$) for a time interval Δ starting at t_0 , such that $\Delta \subseteq I$.

Let $S_0 = t_0, S_1, \dots, S_{k-1}$ be the starting times of these sessions. Then, by Baye's formula,

$$\begin{aligned} P_{ij}(k | (\mathcal{A}, \Gamma_{t_0})) &= (1 - \sigma_{ij}(\mathcal{A}, \Gamma_{S_0})) \cdot (1 - \sigma_{ij}(\mathcal{A}, \Gamma_{S_1})) \cdots \\ &\quad (1 - \sigma_{ij}(\mathcal{A}, \Gamma_{S_{k-2}})) \cdot \sigma_{ij}(\mathcal{A}, \Gamma_{S_{k-1}}). \end{aligned}$$

Since for all \mathcal{A} , all Γ_t ,

$$\sigma_{\min} \leq \sigma_{ij}(\mathcal{A}, \Gamma_t) \leq \sigma_{\max},$$

we have

$$P_{ij}(k | (\mathcal{A}, \Gamma_{t_0})) \leq \sigma_{\max} (1 - \sigma_{\min})^{k-1}$$

and

$$P_{ij}(k | (\mathcal{A}, \Gamma_{t_0})) \geq \sigma_{\min} (1 - \sigma_{\max})^{k-1}.$$

By using the above inequalities and calculating the mean, we get

LEMMA 5.1

$$\frac{\sigma_{\min}}{(\sigma_{\max})^2} \leq \text{mean}(k) \leq \frac{\sigma_{\max}}{(\sigma_{\min})^2}.$$

By known expressions about tails of geometrics, we get

LEMMA 5.2. $\forall \epsilon, 0 \leq \epsilon < 1$,

$$\Pr\{k > k_{\max}(\epsilon)\} < \epsilon$$

where

$$k_{\max}(\epsilon) = \frac{\log((\sigma_{\min} \cdot \epsilon)/\sigma_{\max})}{\log(1 - \sigma_{\min})}.$$

Recall $3cvr_{\max}$ is an upper bound on a session length. Lemmas 5.1 and 5.2 imply

THEOREM 5.4. *If τ is the response of the system, then*

$$\text{mean}(\tau) \leq 3cvr_{\max} \cdot \frac{\sigma_{\max}}{(\sigma_{\min})^2}$$

and if $\tau(\epsilon)$ is the ϵ -response, then

$$\tau(\epsilon) \leq 3cvr_{\max} \cdot k_{\max}(\epsilon).$$

By using $c = (1 + 3/v)(r_{\max}/r_{\min}) \cdot c_R$ and the $\sigma_{\max}, \sigma_{\min}$ of Theorems 5.1 and 5.2, we get

COROLLARY 5.1. *For Algorithm 1*

$$\text{mean}(\tau) \leq 48(1 - e^{-1/2})^{-2} \cdot (r_{\max}^2/r_{\min}) c_R \left(1 + \frac{3}{v}\right) \cdot v^2$$

or

$$\text{mean}(\tau) = O(v^2)$$

and

$$k_{\max}(\epsilon) = \frac{\log((\epsilon/4)(1 - e^{-1/2}))}{\log(1 - (1/4v)(1 - e^{-1/2}))} = O\left(v \log\left(\frac{4}{\epsilon}\right)\right)$$

implying

$$\tau(\epsilon) \leq 3cvr_{\max} \cdot k_{\max}(\epsilon) = O\left(v^2 \log\left(\frac{4}{\epsilon}\right)\right).$$

Also, by using the derived σ_{\min} , σ_{\max} for Algorithm 2 we get

COROLLARY 5.2. *For Algorithm 2,*

$$\text{mean}(\tau) \leq 48(1 - e^{-1/2})^{-2} \cdot \left(\frac{r_{\max}^2}{r_{\min}}\right) \cdot c_R \cdot \left(1 + \frac{3}{v}\right) \cdot v = O(v)$$

and

$$k_{\max}(\epsilon) = \frac{\log((\epsilon/4)(1 - e^{-1/2}))}{\log(1 - \frac{1}{4}(1 - e^{-1/2}))} = O\left(\log\left(\frac{4}{\epsilon}\right)\right),$$

implying

$$\tau(\epsilon) = O\left(v \log\left(\frac{4}{\epsilon}\right)\right).$$

6. CONCLUSION

We have provided two real-time implementations for the DCS system. A key assumption of our time analysis is that processes have to be tame during attempts to communicate, but at other times processes need not be tame. This improves a previous version of this paper [11], where we required processes to be tame at all times.

A referee has suggested a modification of our algorithms that may be of practical use in speeding up the expected time response in some practical cases. The modification presumes that the connections graph has fixed valence (otherwise, an infinite number of variables per process are required). The idea is to allow each process to have additional flag variables, which indicate to other processes its willingness to communicate with them (we had presumed that the set E_i could only be read by process i). The modified algorithms will have worst case performance identical to those given in our paper.

In a further paper [12], we have relaxed our assumption of tameness. In that paper we require only bounds on the relative acceleration of ratios of speeds of neighbor processes. There we propose synchronization algorithms that have *relative* real-time response, where communication is established with high probability between any pair of processes within a constant number of steps of the *slowest* process. However, these algorithms are less efficient than those given in this paper. Also, in [12, Appendix], we apply our synchronization techniques to ADA for a relative real-time implementation.

APPENDIX. A REAL-TIME IMPLEMENTATION OF CSP

Hoare [5] introduced a concurrent programming language CSP (for Communicating Sequential Processes). The CSP language is notable for the elegance of its synchronization constructs: They are powerful and yet simple. Bernstein [3] describes an extension of CSP that allows both input and output commands as guards. Here we briefly describe CSP with Bernstein's extension and present a real-time implementation of the synchronization constructs.

A1. CSP Synchronization Constructs

The relevant aspects of CSP concern its process structure and communication mechanisms. Concurrent execution of processes P_1, P_2, \dots, P_n is denoted

$$[P_1 \parallel P_2 \parallel \dots \parallel P_n].$$

Each process has its own set of variables, which are inaccessible to all other processes. The communication primitives are the *output command* $P_j ! u$, which requests that P_j receive the value of u and *input command* $P_i ? x$, which requests that P_i send a value which is then assigned to x .

There are two relevant compound statements. The *alternative statement*

$$[G_1 \rightarrow C_1 \square G_2 \rightarrow C_2 \square \dots \square G_k \rightarrow C_k]$$

contains guards G_1, \dots, G_k and command lists C_1, \dots, C_k . Each guard consists of a list of elements that may be a sequence of Booleans, followed by at most one input command or (in Bernstein's extension of CSP) an output command.

The execution nondeterminately chooses a guard G_i that is satisfied (to test the satisfiability, it executes each element of G_i from left to right) and then executes the corresponding command list C_i . If no guard is satisfied, the alternative statement *fails*. The *repetitive statement*

$$*[G_1 \rightarrow C_1 \square \dots \square G_k \rightarrow C_k]$$

results in the repeated execution of the alternative statement $[G_1 \rightarrow C_1 \square \dots \square G_k \rightarrow C_k]$, until no guards are satisfied.

Note that the crucial problem in implementing CSP is to synchronize executions of input commands $P_j ? x$ by process P_j with output commands $P_i ! u$ by process P_i so that the value u is transmitted to x .

It is very easy to implement CSP by DCS. (In fact, this was the original motivation for our work on DCS.) Let ϵ be a systemwide constant that may be fixed to any arbitrarily small constant on the interval $(0, 1)$. We assume a *real-time* DCS implementation with ϵ -response $\tau(\epsilon)$. Let v be the maximum number of guards appearing in any alternative or repetitive statement; we assume that v is constant relative to the total number n of processes. We also assume that the length of the guard lists is bounded by a small fixed constant. We also assume all processes reliably execute their programs and satisfy assumptions A1 and A2.

Let J be the event: For a given alternative statement, the execution either determines a satisfied guard and executes the corresponding command list, or determines that no guard is satisfied and makes a failure exit from the statement.

Our CSP implementation is real time in the sense that J takes at most $\tau(\epsilon)$ time with probability at least $1 - \epsilon$. Furthermore, J takes finite time with

probability 1. (In fact, a conclusion of Theorem 5.3 is that the time of execution of J is upper bounded by a geometric distribution.)

To attempt to execute an output command $P_j!u$ in process P_i , P_i sets $P_i \rightarrow P_j$, indicating that P_i is willing to communicate with P_j . Also, to attempt to execute an input command $P_i?x$ in process P_j , P_j sets $P_j \rightarrow P_i$. If successful communication is established by P_i and P_j , the process P_j immediately transmits value u to variable x in P_i ; and immediately thereafter P_i sets $P_i \not\rightarrow P_j$ and P_j sets $P_j \not\rightarrow P_i$.

An alternative or repetitive statement S may contain the execution of one of several guarded input commands and output commands, say G_1, \dots, G_s where $s \leq v$. To execute the statement S , P_i first executes the Booleans appearing in each guard. If no guard is satisfied, P_i must then exit the statement S with failure. Otherwise, let R be the set of processes appearing in those guards of S , all of whose Booleans evaluate to true. P_i must set $P_i \rightarrow P_j$ for each $P_j \in R$. At the first time that an appropriate communication is established between P_i and some willing process $P_j \in R$, P_i must execute the command list associated with the now satisfied guard in the statement S . Note that P_i can ignore the communications established later between i and the processes of $R - \{j\}$.

ACKNOWLEDGMENTS

We wish to thank Ed Clarke, who introduced us to the synchronization problems considered in this paper, and Michael Rabin, whose previous work in probabilistic synchronization inspired this work. Stavros Macrakis is thanked for helpful comments on our real-time CSP implementation. Also, the referees made many very useful comments.

REFERENCES

(Note. References 1, 2, and 7 are not cited in text.)

1. ANGLUIN, D. Local and global properties in networks of processors. In *Proceedings of 12th Annual ACM Symposium on Theory of Computing* (Los Angeles, Calif., Apr. 28–30) ACM, New York, 1980, pp. 82–93.
2. ARJOMANDI, E., FISCHER, M.J., AND LYNCH, N.A. A difference in efficiency between synchronous and asynchronous systems. In *Proceedings of 13th Annual ACM Symposium on Theory of Computing* (Milwaukee, Wis., May 11–13), ACM, New York, 1981, pp. 128–132.
3. BERNSTEIN, A.J. Output guards and nondeterminism in “communicating sequential processes.” *ACM Trans. on Program. Lang. Syst.* 2, 2 (Apr. 1980), 234–238.
4. FRANCEZ, N., AND RODEH. A distributed data type implemented by a probabilistic communication scheme. In *Proceedings of 21st Annual Symposium on Foundations of Computer Science* (Syracuse, N.Y., Oct.) 1980, pp. 373–379.
5. HOARE, C.A.R. Communicating sequential processes. *Commun. ACM* 21, 8 (Aug. 1978), 666–677.
6. LEHMANN, D., AND RABIN, M.O. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers’ problem. In *Conference Record of 8th ACM Symposium on Principles of Programming Languages* (Williamsburg, Va., Jan. 26–28), ACM, New York, 1981, pp. 133–138.
7. LIPTON, R., AND SAYWARD, F.G. Response time of parallel programs. Res. Rep. 108. Dept. of Computer Science, Yale Univ., New Haven, Conn., June 1977.
8. LYNCH, N.A. Fast allocation of nearby resources in a distributed system. In *Proceedings of 12th ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 2, April 1984.

- Annual ACM Symposium on Theory of Computing* (Los Angeles, Calif., Apr. 28–30) ACM, New York, 1980, pp. 70–81.
- 9. RABIN, M. The choice coordination problem. Mem. UCB/ERL M80/38. Electronics Research Lab., Univ. of California, Berkeley, Aug. 1980.
 - 10. RABIN, M. N -Process synchronization by a $4 \log_2 N$ -valued shared variable. In *Proceedings of 21st Annual Symposium on Foundations of Computer Science* (Syracuse, N.Y., Oct.) 1980, pp. 407–410.
 - 11. REIF, J.H., AND SPIRAKIS, P.G. Distributed algorithms for synchronizing interprocess communication within real time. In *Proceedings of 13th Annual ACM Symposium on Theory of Computing* (Milwaukee, Wis., May 11–13), ACM, New York, 1981, pp. 133–145.
 - 12. REIF, J.H., AND SPIRAKIS, P.G. Unbounded speed variability in distributed communication systems. In *Conference Record of 9th ACM Symposium on Principles of Programming Languages* (Albuquerque, New Mex., Jan. 25–27) ACM, New York, 1982, pp. 46–56.
 - 13. REIF, J.H., AND SPIRAKIS, P.G. Real time resource allocation in distributed systems. In *Proceedings of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (Ottawa, Canada, Aug. 18–20), ACM, New York, 1982, pp. 84–94.
 - 14. SCHWARZ, J. Distributed synchronization of communicating sequential processes. DAI Res. Rep. 56, Univ. of Edinburgh, Scotland, 1980.
 - 15. VALIANT, L.G. A scheme for fast parallel communication. Tech. Rep., Computer Science Dept., Univ. of Edinburgh, Scotland, July 1980.

Received February 1981; revised October 1981, June 1982, July 1983; accepted July 1983