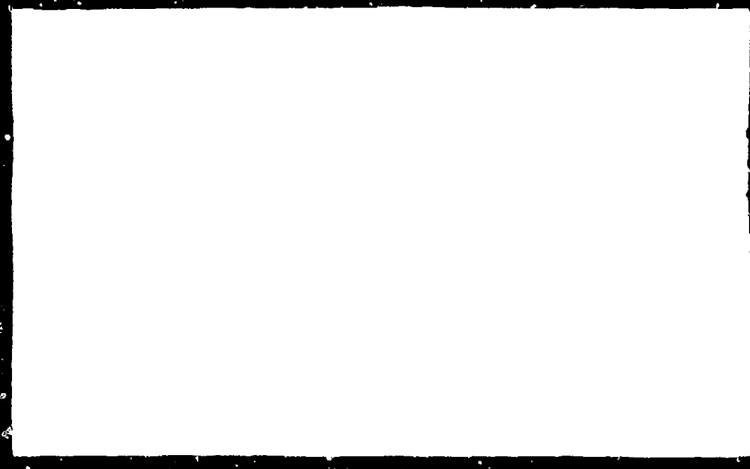
LEVEL II

AD A108825



DTIC
ELECTE
DEC 23 1981
S D
D

REAL-TIME SYNCHRONIZATION OF INTERPROCESS COMMUNICATION

John H. Reif
Paul G. Spirakis

TR-23-80

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A108825	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Real-Time Synchronization of Interprocess Communication	5. TYPE OF REPORT & PERIOD COVERED Technical Report
	6. PERFORMING ORG. REPORT NUMBER TR-23-80
7. AUTHOR(s) John H. Reif and Paul G. Spirakis	8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0674
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University Cambridge, MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 North Quincy Street Arlington, VA 22217.	12. REPORT DATE 1980
	13. NUMBER OF PAGES 38
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) same as above	15. SECURITY CLASS. (of this report)
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

real time, synchronization, parallel algorithm, distributed communication, CSP, multiprocessing

81 12 22 118

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper considers a fixed (possibly infinite) set of distributed asynchronous processes which at various times are willing to communicate with each other.

We describe probabilistic algorithms for synchronizing this communication with boolean "flag" variables, each of which can be written by only one process and read by at most one other process.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20.

The use of flag variables seems as to require the fewest assumptions possible without considering specific systems.† A process is considered to be *tame* over a time interval (Δ) if its speed varies within certain arbitrarily fixed nonzero bounds.

We show our synchronization algorithms have *real time response:*

If a pair of processes are mutually willing to communicate within a time interval (Δ) and the pair are tame on (Δ), then they establish communication within (Δ) with high likelihood (for the worst case behavior of the system).

We have very few assumptions: (1) Tameness is required of a process only during the interval it is willing to communicate (if the tameness property is violated during that interval, then there may be lower probability of successful communicatipn); 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 which are only number. There are *no* probability assumptions about system behavior.

Our communication model and synchronization algorithms are quite robust. They are applied to solve a large class of real time resource synchronization problems, as well as real time implementation of the synchronization primitives of Hoare's multiprocessing language CSP.

† 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.

REAL-TIME SYNCHRONIZATION OF INTERPROCESS COMMUNICATION *

by

John H. Reif and Paul G. Spirakis
Harvard University
Aiken Computation Laboratory
Cambridge, MA 02138

*This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024 and the Office of Naval Research Contract N00014-80-C-0674.

SUMMARY

This paper considers a fixed (possibly infinite) set Π of distributed asynchronous processes which at various times are willing to communicate with each other.

We describe probabilistic algorithms for synchronizing this communication with 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 as to require the fewest assumptions possible without considering specific systems.† A process is considered to be *tame* over a time interval Δ if its speed varies within certain arbitrarily fixed nonzero bounds.

We show our synchronization algorithms have *real time response:*

If a pair of processes are mutually willing to communicate within a time interval Δ and the pair are tame on Δ, then they establish communication within Δ with high likelihood (for the worst case behavior of the system).

We have very few assumptions: (1) Tameness is required of a process only during the interval 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 which are only bounded in number. There are *no* probability assumptions about system behavior.

Our communication model and synchronization algorithms are quite robust. They are applied to solve a large class of real time resource synchronization

† 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.

problems, as well as real time implementation of the synchronization primitives of Hoare's multiprocessing language CSP.

1 INTRODUCTION

Recently, [Rabin, 80], [Lehman and Rabin, 81], and [Francez and Rodeh, 80] have proposed probabilistic algorithms for a number of synchronization problems. This *probabilistic approach* (where we make no probabilistic assumptions about the system behavior, but allow our algorithms to make probabilistic choices) leads to considerably *simpler algorithms* (perhaps because of the locality of their decisions) and *shorter proofs* (perhaps because the proofs of the corresponding deterministic algorithms had to consider complex situations which would have very low probability, if probabilistic choices were taken). The probabilistic approach may also lead to improvement in the efficiency of synchronization algorithms. An improvement in space efficiency is seen in [Rabin, 80]. 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 [Tonag, 80] and [Valiant, 80]. Furthermore, we consider *handshake* communication (as in Hoare's CSP), rather than *buffered* communication (which is very easy to implement by message queues).

Previously [Schwartz, 80] proposed a deterministic synchronization algorithm for implementing CSP [Hoare, 78] on a fixed acyclic distributed network. Also [Lynch, 80] gave a related algorithm for resource synchronization problems. Both algorithms are considerably less time efficient than our proposed algorithm (for a specific comparison of time performance, see

Section 2.E). [Francez and Rodeh, 80] also propose a probabilistic solution to synchronization of communication, but make no consideration of the time efficiency of their solution.

Our paper is organized as follows: As is usual in the study of combinatorial problems, we state the problem before giving our proposed solution. We present in Section 2 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 synchronization problem of interest. This model also allows a precise definition of the relevant complexity measures of synchronization algorithms, such as response time. Section 3 presents our synchronization algorithms, and in Section 4 we prove various properties of the synchronization algorithms which must hold with certainty, regardless of probabilistic choice. Sections 5 and 6 give a probabilistic analysis of the performance of our algorithms. We have taken 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 CSP.

[Reif and Spirakis, 81C] presents a further application: a real time resource granting system. We feel these applications demonstrate the broad applicability of our synchronization algorithms.

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

Let $\Pi=\{1,2,\ldots\}$ be a fixed, (possibly infinite) collection of *processes*. We assume a (global) *time* t, on the nonegative 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. (In contrast, [Arjomandi, Fischer, Lynch, 81] consider synchronization problems with access to a global clock.)

We also assume a global oracle $\mathcal{A}$ which directs the willingness of processes to communicate with each other. (Note that, in applications of DCS occuring in practice, no such oracle $\mathcal{A}$ exists, but instead each process is running some program which 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 where 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 a global centralized synchronization mechanism. Thus system-wide communication is implemented by a distributed scheduler, the processes of Π.

The formal model *DCS* (for *D*istributed *C*ommunication *S*ystem) described below, has been designed with as few assumptions as possible and 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 which may vary for any given application. Also, we are concerned only with *direct* (two way) *communication* between processes; we are not concerned with packet switching, as in [Valiant, 80] and [Tonag, 80].

We now introduce some graphs to precisely describe the DCS model. These 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.

We assume a possibly infinite undirected graph H, the *connections graph* with vertices Π, and undirected edges given by symmetric antireflexive relation $\longleftrightarrow \subseteq (\Pi\times\Pi)-\{(i,i) : i\in\Pi\}$. Then $i\longleftrightarrow j$ denotes that $i\in\Pi$ *is physically able to communicate* with $j\in\Pi-\{i\}$. H is fixed for all time and can be considered to be essentially the hardware connections between processes of Π. We assume H has finite valence (i.e., only a finite number of processes are connected to any given process $i\in\Pi$).

For each time $t\geq 0$, we assume a possibly infinite directed graph G_t, the *willingness digraph* with vertices Π and directed edges given by relation $\xrightarrow{t} \subseteq \Pi\times\Pi$. Then $i \xrightarrow[t]{} j$ denotes that $i\in\Pi$ is *willing to communicate* with $j\in\Pi-\{i\}$ at time t. (In that sense we say i is the *source* and j is the *target*). We require that $i\longleftrightarrow j$ is $i \xrightarrow[t]{} j$ so i is willing to communicate only with processes which i is able to communicate with.) Also, let $i \xleftrightarrow[t]{} j$ if both $i \xrightarrow[t]{} j$ and $j \xrightarrow[t]{} i$. Thus, $i \xleftrightarrow[t]{} j$ denotes that i,j are both willing to communicate at time t. For each time interval Δ on $(0,\infty)$, let $i \xrightarrow[\Delta]{} j$ if $i \xrightarrow[t]{} j$ for all $t\in\Delta$, and let $i \xleftrightarrow[\Delta]{} j$ if both $i \xrightarrow[\Delta]{} j$ and $j \xrightarrow[\Delta]{} i$. Thus $\xrightarrow[\Delta]{}$ and $\xleftrightarrow[\Delta]{}$ denote that the willingness to communicate holds over time intervals. The edges of G_t departing from $i\in\Pi$ are assumed to be stored locally at i, in the form of a set E_i whose elements are the names of the targets of i at time t. The set E_i is specified by the oracle and read only by i. □

Assumption A1 We assume that 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.A. 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 $t \geq 0$, we assume a (possibly infinite) directed graph M_t with vertices Π and directed edges given by relation $\overset{t}{\rightsquigarrow} \subseteq \Pi \times \Pi$. Then $i \overset{t}{\rightsquigarrow} j$ denotes i *opens communication* with $j \in \Pi - \{i\}$ at time t. Let $i \overset{t}{\leftrightsquigarrow} j$ if both $i \overset{t}{\rightsquigarrow} j$ and $j \overset{t}{\rightsquigarrow} i$. Then, $i \overset{t}{\leftrightsquigarrow} j$ denotes 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 We also assume that if $i \xrightarrow[t_1]{} j$ and not $i \xrightarrow[t_2]{} j$, $t_2 > t_1$, then $i \overset{t}{\leftrightsquigarrow} j$ for some t, $t_1 \leq t < t_2$; i.e., the oracle $\mathcal{A}$ can withdraw willingness to communicate only after communication has been established.

For each $i,j \in \Pi$ such that $i \neq j$ we assume a *communication port flag* $\text{PORT}_{i,j}$ (controlled by process i) which is 1 at time $t \geq 0$ if $i \overset{t}{\rightsquigarrow} j$ (i.e., process i has *opened* its port for communication with j) and 0 otherwise (indicating the communication port from i to j is *closed*). Thus, $i \overset{t}{\leftrightsquigarrow} j$ if and only if $\text{PORT}_{i,j}$ and $\text{PORT}_{j,i}$ are simultaneously 1 at time t. We assume 2-way communication between i,j is possible at any time both $\text{PORT}_{i,j}$ and $\text{PORT}_{j,i}$ are simultaneously 1, but we make no particular assumptions (beyond assumptions A1, A2) about this communication.

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

R1 $i \overset{t}{\rightsquigarrow} j$ only if $i \overset{t}{\leftrightarrow} j$

R2 $\overset{t}{\leftrightsquigarrow}$ is a (partial) matching; if $i \overset{t}{\leftrightsquigarrow} j$ then not $i \overset{t}{\leftrightsquigarrow} j'$ for any $j' \in \Pi - \{j\}$.

Note that R1 implies that the poller of 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 [Reif, Spirakis, 81B].

2.B. Global State of the DCS

For each $t \geq 0$, let R_t be a mapping from Π to the reals giving the speed of each process of Π at time t. We assume the speed schedule $R = \{R_t | t \geq 0\}$ is chosen by an oracle $\mathcal{A}$ (our scheduler's worst "enemy") at time $t = 0$. Also, we assume 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

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

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.C. Complexity Measures on DCS Implementations

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 $\{j \mid i \xrightarrow[t]{} j\}$) is bounded above by v.

Let process i be *tame* on the interval Δ, if its *speed* (steps per real time unit) of i at times t on Δ is on the interval $[\frac{1}{r_{max}}, \frac{1}{r_{min}}]$ 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). A *step* consists of either an assignment of a variable, a test, a logical or arithmetic operator, or a *no-op*.

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

Let processes i,j have *successful communication* at interval Δ if $i \overset{\Delta}{\rightsquigarrow} j$ 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) $i \overset{\Delta}{\longleftrightarrow} j$,

(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 since Δ is maximal, either i,j were not mutually willing to communicate immediately before Δ, or Δ begins at time 0. Also note that

an oracle can make a response interval infinite if i,j do not ever have successful communication.)

Let the *response time* of a DCS implementation, for any oracle $\mathcal{A}$, be the random variable $\tau_{\mathcal{A}}$ giving the length of a response interval.

Let $\bar{\tau} = \max\{\text{mean}(\tau_{\mathcal{A}})/\text{all oracles } \mathcal{A}\}$. For each ε, $0 \leq \varepsilon \leq 1$, let the ε-*error response* $\tau(\varepsilon)$ (note: this is a function, not a random variable) be the least upper bound on the inverse of the cumulative distribution function of $\tau_{\mathcal{A}}$. Thus, if we have a finite interval Δ, $|\Delta| \geq \tau(\varepsilon)$ and any two processes i,j which are tame on Δ, for all oracles $\mathcal{A}$, $i \overset{\Delta}{\longleftrightarrow} j$ implies i,j have successful communications sometime within Δ, with probability $\geq 1-\varepsilon$. Note that time response as defined above for pairs of processes also holds for communication between sets of processes. Suppose we have finite sets of processes $\Pi_1, \Pi_2 \subseteq \Pi$ such that for all i in Π_1 and all j in Π_2, $i \overset{\Delta}{\longleftrightarrow} j$ for the same finite interval Δ of length $\geq \tau(\varepsilon)$. Then, for all oracles $\mathcal{A}$ and all $i \in \Pi_1$ and $j \in \Pi_2$, i and j have successful communication sometime within Δ with probability $\geq 1-\varepsilon$ for each pair of processes.

This implies a very robust type of fairness. Each pair of $\Pi_1 \times \Pi_2$ are guaranteed $\geq 1-\varepsilon$ probability of successful communication within Δ, independent of the success of communication by other processes.

The DCS implementation is *real time* if for all ε, $0 < \varepsilon \leq 1$, $\tau(\varepsilon)$ is a constant dependent only on v, assumed to be a constant upper bound on the outdegree of vertices of G_t) *and* $\bar{\tau}$ is bounded above by a fixed constant. (Note that $\bar{\tau} \leq \int_0^1 \tau(\varepsilon) \cdot \varepsilon \, d\varepsilon$).

2.D. Insisting DCS Implementations

We also consider the cases where any given process $i \in \Pi$ may assign a *priority* to the processes $j \in \Pi - \{i\}$ which i wishes to communicate with.

In the simplest case, i distinguishes the *first target* of communication, say $E_i(1)$, which i *insists* on communicating with (process i may eventually communicate with the other processes of E_i, but i insists on communicating with $E_i(1)$ with highest priority). For each $t \geq 0$, $\xrightarrow{t}'$ is the relation on $\Pi \times \Pi$ such that $\forall i,j \in \Pi$ $i \xrightarrow{t}' j$ iff $E_i(1) = j$ at time t (also let $i \xrightarrow{\Delta}' j$ if $i \xrightarrow{t}' j$ $\forall t \in \Delta$).

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

(1) $i \xrightarrow{\Delta}' j$ and $j \xrightarrow{\Delta} i$

(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.

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

Let the *insisting response time* of a DCS implementation be the random variable $\tau'_{\mathcal{A}}$, for each oracle $\mathcal{A}$, giving the length of an insisting response interval. Let $\bar{\tau}' = \max\{\text{mean}(\tau')/\text{all oracles } \mathcal{A}\}$. For each ε, $0 < \varepsilon \leq 1$ let the ε-*error insisting response* $\tau'(\varepsilon)$ be the least upper bound on the inverse of the cumulative distribution function of $\tau_{\mathcal{A}}$.

Thus, if we have a finite interval Δ, $|\Delta| \geq \tau'(\varepsilon)$ and any two processes i,j which are tame on Δ, for every oracle $\mathcal{A}$, ($i \xrightarrow{\Delta}' j$ and $j \xrightarrow{\Delta} i$) implies i,j have successful communication sometime within Δ, with probability $\geq 1-\varepsilon$.

The DCS implementation has *real time insisting response* if for all ε, $0 < \varepsilon \leq 1$, $\tau'(\varepsilon)$ is a constant, independent of any parameter of H (execpt v) *and* $\bar{\tau}'$ is bounded above by a fixed constant.

It is useful to observe that, given $\tau'(\varepsilon)$, any given process $i \in \Pi$ may determine (with any given probability) whether any other process $j \in \Pi - \{i\}$ is

willing to communicate with i over a given time interval in which both i,j are tame.

PROPOSITION 2.1. Let $\mathcal{A}$ be any oracle and Δ be any time interval of finite length $\geq \tau'(\varepsilon)$. Suppose i,j are tame on Δ. If $i \xrightarrow[\Delta]{}' j$ but there is no $t \in \Delta$ such that $i \overset{t}{\rightsquigarrow} j$, then j is not willing to communicate with i sometime during Δ, with probability $\geq 1-\varepsilon$. This proposition may be used for *timing out insisting* 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 $\Delta \geq \tau'(\varepsilon)$ such that for all $i \in \Pi_1$, $i \xrightarrow[\Delta]{}' j$ and $j \xrightarrow[\Delta]{} i$. Assume also, 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 have successful communication sometime with Δ, with probability $\geq 1-\varepsilon$.)

2.E. Results and Previous Work

The primary results for this paper are:

There is a *real time implementation* of DCS such that

(1) the worst case mean response $\bar{\tau}$ is $O(v^2)$;

(2) the ε-error response $\tau(\varepsilon)$ is $O(v^2 \log(1/\varepsilon))$.

Also, there is a *real time implementation* of DCS such that

(1) worst case mean *insisting* response $\bar{\tau}'$ is $O(v)$;

(2) the ε-error *insisting* response $\tau'(\varepsilon)$ is $O(v \log(1/\varepsilon))$.

These results follow from a single general theorem of Section 4.

Our implementations are proper, symmetric, and are completely independent of the connection graph H (H may be any finite or infinite graph with finite valence). Our innovation, which results in real time response, is to allow processes to make probabilistic choices.

The best previous result is due to [Schwartz, 80] and is restricted to the case H is finite and 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 [Schwartz, 80] 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 ofprocesses (of length $\chi(H)$ can be formed in which each process waits for the next to reply. So the deterministic DCS implementation of [Schwartz, 80] has insisting 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 order $|\Pi|$. Also, his DCS implementation is *not by metric*, since processes are required to know their color in H'.

Also [Lynch, 80] gives a solution to a distributed resource allocation problem, which in [Reif, Spirakis, 81C] is adopted to yield a DCS implementation with response time $v^{\chi(H)}$. (In [Reif, Spirakis, 81C] we show that a class of generalized resource allocation problems related to those of [Lynch, 80] may be efficiently solved by our DCS implementation.)

[Francez, Rodeh, 80] proposed a probabilistic synchronization algorithm which can be considered to be a DCS implementation. An important difference between our implementation and theirs is that in the responding phase, in our algorithms, each process responds to all processes to which it is willing to communicate, while in [Francez, Rodeh, 80] only one process is considered at a time. Although [Francez, Rodeh, 80] make no explicit timing assumptions, they do assume that setting and resetting of shared variables takes only a negligible time compared to 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 were essentially not considered in any previous papers on synchronization. Also, we utilize probabilistic choice in new ways than those utilized in [Francez, Rodeh, 80]. In particular, we utilize *random waits* to ensure that the oracle cannot make the behavior of waiting processes depend on choices of partners made by other processes (see the analysis of the insisting algorithm).

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,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 "noninsisting" implementation, and Algorithm 2 in our "insisting" 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).

Our following description of the DCS implementations will be 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 which are irrelevant to the superfluous understanding of our algorithms.) Also, before giving the formal specifications of any algorithm or procedure, we give in English an informal

description of its actions. The actual formal algorithms have been written carefully 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 a probability 1/2.

Algorithm 1 (Non-insisting)

In the responding phase, process i repeats a loop v times. On each iteration of the loop, process i chooses at random a process j from the processes i is willing to communicate with, and executes procedure $\text{RESPOND}_i(j)$. This procedure takes constant number of steps (namely c_R) and during it process i tests a flag to determine if j is currently willing to talk to i and then sets a flag to determine if j pays attention to i (these tests have the form of a *handshake*). If so, processes i and j synchronize their steps and then both open communication to each other. In either case, i repeats the loop until the responding phase finishes.

In the asking phase, process i chooses *only once* at random a process j to which i is willing to communicate with, and the i executes procedure $\text{ASK}_i(j)$. This procedure takes $c_A = c_R \cdot v$ steps (so that both phases take exactly the same number of steps. As a consequence, process i is in each phase half of the time on the average. This is important to the analysis). During procedure $\text{ASK}_i(j)$, process i raises a flag to show to j that it is willing to communicate currently with j, and then pays attention to j for a limited number of steps to test if j responds to the attempt and wants to proceed in communication. If so, 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 clearing its flags.

Algorithm 2 (Insisting)

Each process executes forever the following loop: It begins by choosing a random integer w from $\{0,1,2,\ldots,c_A\}$. It then waits for w steps and then it chooses with probability 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. After executing one or the other of the phases, process i then waits for $c_A - w$ steps. (This guarantees that, at any time t, a process is *not* waiting with probability 1/2, and that a process is asking (given it is not waiting) with probability 1/2).

We now give Algorithms 1 and 2 in full detail.

Algorithm 1 (noninsisting implementation)

Program for process $i \in \Pi$

```
INITIALIZE_i( );
WHILE TRUE DO
  BEGIN

    L2:  CHOOSE a random  b∈{0,1}
  IF  b = 0  THEN
      BEGIN
      COMMENT:  respond phase
        L3:  FOR  x=1  to  v  DO
          BEGIN
            CHOOSE at random  j ∈ E_i
            RESPOND_i (j);
          END
      END
  ELSE
      BEGIN
      COMMENT:  ask phase
        L4:  CHOOSE at random  j ∈ E_i
             ASK_i (j)
      END

  END

OD
```

Algorithm 2 (the insisting implementation)

Program for process $i \in \Pi$

```
INITIALIZE_i( )
WHILE TRUE DO
  BEGIN

    L1: CHOOSE w at random from {0,1,...,c_A}
        DO w no-ops
    L2: CHOOSE a random b∈{0,1}
      IF b = 0 THEN
        BEGIN
        COMMENT: respond phase
        L3: FOR x=1 to v DO
            BEGIN
              CHOOSE a random j ∈ E_i
              RESPOND_i(j)
            END
        END
      ELSE
        BEGIN
        COMMENT: ask phase
        L4: ASK_i(E_i(1))
        END

      DO c_A - w no-ops
  END
OD
```

3.A. 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 $i \neq j$, 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 answer 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 0 by i only during the *"watching window"* which is the interval when i is in the asking phase and is watching for an answer ($A_{ji} = 1$) from j. At all other times, B_{ij} is set to 1 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, $i \overset{t}{\rightsquigarrow} j$ 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(target) then it first sets a flag variable $Q_{i,target}$ to 1 (to indicate to j that it asks) and sets another flag $B_{i,target}$ to 0 (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 $A_{target,i}$ (the answer flag of target). If the interval finishes with no answer from target, then i *first* sets $B_{i,target}$ to 1 (to show that it stops paying attention to target) and *then* it sets $Q_{i,target}$ to 0 to drop the question. This order of actions guarantees that process target will interpret correctly what it sees from the flags of i.

If i gets an answer from j (that is, if $A_{target,i}$ is set to 1) during the (previously discussed) c_B steps, then i *first* sets $Q_{i,target}$ to 0 (but keeps $B_{i,target}$ to its current value to indicate that it continues to pay attention to j). Process i waits until target also zeros its flag $A_{target,i}$ and then process i calls $OPEN\text{-}COM_i$ (target) immediately. As the analysis shows, the events leading to this call guarantee that communication is achieved between i and target during the execution of OPEN-COM, assuming i and target are tame (we do not use a handshake protocol within

OPEN-COM since unfortunately our timing constraints would be violated if we were to require that communication between i and target be successful irregardless of whether they are tame). At the end of OPEN-COM, i sets $B_{i,target}$ to 1 (showing that it stops paying attention to target) and exits procedure ASK_i.

If process i executes procedure $RESPOND_i$ (asker), then it first examines if $Q_{asker,i}$ is 1 (i.e., if asker is interested in communicating with i). If so, then i sets $A_{i,asker}$ to 1 and waits until process asker zeros its question flag (this is the "handshake" technique). When this happens, then i tests $B_{asker,i}$ to see if process asker still pays attention to i. If not, then i zeros its answer flag $A_{i,asker}$ and exits. Else, i knows that asker waits for step synchronization and communication. So, i zeros its flag $A_{i,asker}$ and calls $OPEN\text{-}COM_i$ (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(j), and it is in the *responding mode* when it executes the procedure RESPOND. If i is executing ASK(j) and $B_{ij}=0$ 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 with a=1. A *phase* of the algorithms consists of the steps between random choices of the variable $b \in \{0,1\}$. If b=0 the process is in *an answering phase* and 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 ∈ Π  such that  i≠j  do
      BEGIN
        Q_ij ← 0
        A_ij ← 0
        B_ij ← 1
        PORT_ij ← 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 of the execution of procedures RESPOND, ASK are each always exactly the same. (This is crucial to our proof of real time response.)

We define the constants appearing in the procedures:

Let $c_R = 6 + 20 \frac{r_{max}}{r_{min}}$; this will be precisely the number of steps always required by procedure RESPOND.

Let $c_A = c_R v$; this will be the number of steps required by procedure ASK.

Let $c_B = c_A - (6 + 20 \frac{r_{max}}{r_{min}})$; this is the number of steps required for a watching window.

Let $c_P = 2 + 3 \frac{r_{max}}{r_{min}}$; this is the number of steps required in procedure OPEN-COM.

Let $c_D = c_A - c_P - 2$; this constant is used in our algorithms.

```
PROCEDURE  ASK_i(target)
local a

BEGIN

A1:  CURSTEP←0

A2:  Q_{i,target}←1
     a←0.
     B_{i,target}←0
     COMMENT:  Begin watching window for target

A3:  WHILE  CURSTEP < c_B  AND  a=0  DO  a←A_{target,i}
     IF  CURSTEP ≥ c_B  AND  a=0  THEN  B_{i,target}←1
     Q_{i,target}←0
     IF  a=1  THEN
     BEGIN

     A4:  WHILE (A_{target,i}=0  AND  CURSTEP < c_D  AND  a=1) DO  a←A_{target,i}

     A5:  IF  a=0  AND  CURSTEP ≤ c_D  THEN  OPEN-COM_i(target)

     END

     COMMENT:  End watching window for target
     B_{i,target}←1
     WHILE  CURSTEP < c_A  DO  no-op

END


PROCEDURE  RESPOND_i(asker)
local q

BEGIN  CURSTEP←0
       q←Q_{asker,i}

B1:  IF  q=1  THEN
     BEGIN
     A_{i,asker} ← 1

     B2:  WHILE (CURSTEP < c_D  AND  q=1) DO  q←Q_{asker,i}
          q← (q  OR  B_{i,asker} = 1  OR  CURSTEP ≥ c_D)

     B3:  A_{i,asker}←0
          IF ¬q  THEN  B4: OPEN-COM_i(asker)


B5:  WHILE  CURSTEP < c_R  DO  no-op

END
```

```
PROCEDURE  OPEN-COM_i(j)
BEGIN
  PORT_ij ← 1
  DO  c_p - 2  no ops
  PORT_ij ← 0
END
```

4.A. Correctness Properties of the Algorithms which 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 which 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 all the processes in Π execute Algorithm 1, or they all execute Algorithm 2.)

LEMMA 4.1. For both algorithms,

$$i \overset{t}{\rightsquigarrow} j \text{ only if } i \overset{t}{\longleftrightarrow} j.$$

Proof. Process i calls $\text{OPEN-COM}_i(j)$ 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 $B_{j,i}=0$. In both cases, i was willing to communicate with j in the start of the execution of its phase, since i asks (or responds) only to processes it is willing to communicate with. So, $i \overset{t'}{\longrightarrow} j$ where t' was the time of start of i's phase. By assumption (A2) then, $i \overset{t}{\longrightarrow} j$.

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

$j \xrightarrow[t]{} i$.

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

In both cases, $i \overset{t}{\rightsquigarrow} j \Rightarrow i \overset{t}{\longleftrightarrow} j$. □

LEMMA 4.2. For both algorithms,

$\overset{t}{\rightsquigarrow}$ 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 $\overset{t}{\rightsquigarrow}$ is one to one. Hence $\overset{t}{\rightsquigarrow}$ cannot be more than a matching. □

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

Proof. By the Lemmas 4.1, 4.2 both R1,R2 hold.

4.B. Timing Lemmas Which Hold With Certainty

LEMMA 4.3. For both algorithms, if i is in its watching window for j and is answered by j, then i,j have successful communication, within 20 steps of the slowest of i,j, given both i,j are tame.

Proof. If i exits the A3 loop, then (since no process but j can assign to $A_{j,i}$) at the same time j must be executing $\text{RESPOND}_j(i)$ at the B2 loop. Process i will arive at A4 within 7 of its steps and will have, by then, set Q_{ij} to 0. Within $7\, r_{max}/r_{min}$ steps of j, j will have exited the B2 loop. Also at this time, the assumption that i exits the loop A3 with $a=1$ implies that $B_{ij}=0$. So, j will arrive at B3 and set $A_{j,i}$ to 0 in at most 7 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 its steps i will call $\text{OPEN-COM}_i(j)$ and within one

of j's steps j will call $\text{OPEN-COM}_j(i)$. Note that both i,j will set their respective port flags PORT_{ij}, PORT_{ji} to 1 within one step of the slowest process (or, within at most r_{max}/r_{min} steps of the fastest). 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 $2\, r_{max}/r_{min} \cdot r_{min} = 2\, r_{max}$ time, guaranteeing at least 1 step overlap of both processes. Thus, i,j have successful communication. Note that OPEN-COM takes c_p steps. The Lemma follows by counting steps. □

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

Proof. $i \overset{\Delta}{\rightsquigarrow} j$ implies $i \overset{\Delta}{\rightsquigarrow} j$. The only sequence of events leading to that 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. Also, by Lemma 4.3, $|\Delta| \leq 20\, r_{max}^2/r_{min}$ □

LEMMA 4.5*. For both algorithms, if i, j are tame on Δ and $i \overset{\Delta}{\rightsquigarrow} j$ for a maximal interval Δ, then $i \overset{\Delta'}{\rightsquigarrow} j$ for some $\Delta' \subseteq \Delta$. Furthermore i,j have successful communication during Δ'.

Proof. The only sequence of events leading to $i \overset{\Delta}{\rightsquigarrow} j$ 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'. □

*Note: Lemma 4.5 means that a tame process never opens its channel to another tame process without communicating with it.

In the following lemma, we do not necessarily assume 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 Algorithm 1 requires exactly $c_A + 2$ steps and each phase of Algorithm 2 requires $2c_A + 2$ steps.

Proof. By observation of timeouts within the procedures ASK and RESPOND.

Corollary 4.2. Let cv be the number of steps required for each phase ($c = c_R + 2$ for Algorithm 1 and $c = 2c_R + 2$ for Algorithm 2, as implied by Lemma 4.6). Then the maximum time required for each phase is $< cvr_{max}$.

5. PROBABILISTIC ANALYSIS OF THE RESPONSE TIME OF THE ALGORITHMS.

The analysis done here holds for both Algorithms 1 and 2 (except that they differ in the parameter σ^{min} defined below). We assume here the terminology of section 4.

Intuitively, in both algorithms, the ASK or RESPOND phrases take O(v) time each. In the worst case of the non-insisting algorithm, it requires O(v) expected executions of the ASK phrase to choose any given willing neighbour, if the set of willing neighbours is O(v). Given that a given neighbour is chosen and he is willing, communication will be achieved with probability bounded below by a constant. Hence, we expect the average time of response of the non-insisting algorithm to be $O(v^2)$.

On the other hand, in the asking phase of the insisting algorithm we ask a specific neighbour 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 insisting algorithm will be O(v) in the worst case.

A formal analysis follows.

Let I be a time interval of length $\geq 2cvr_{max}$ (i.e., at least two phases), such that $i \overset{}{\underset{I}{\leftrightarrow}} j$ and i,j both tame on I. Fix some time $t \geq 0$. Let Γ_t be the global system history up to t, derived from oracle $\mathcal{A}$ and "luck up to time t" as defined in Section 2B. Note that $(\mathcal{A},\Gamma_t)$ essentially specifies everything of the system's immediate future except the pollers' "luck" $L_{t'}$ for time $t' > t$. For all $i,j \in \Pi$ let $\sigma_{ij}(\mathcal{A},\Gamma_t)$ be the probability that the poller of i is answered by j some time within Δ, *given* i *is in a watching window for* j during time interval Δ starting at time t, and assuming both i,j are tame on Δ, where $\Delta \subseteq I$

In the following analysis, we assume constants σ^{min}, σ^{max} such that

(*) $0 < \sigma^{min} \leq \sigma_{ij}(\mathcal{A},\Gamma_t) \leq \sigma^{max} \leq 1$ for all $i,j, \in \Pi$, for all $t \geq 0$, all oracles $\mathcal{A}$, and any global system history Γ_t, consistent with $\mathcal{A}$.

In the next section we show

THEOREM 5.1: For Algorithm 1, we may let

$$\sigma^{min} = \frac{1}{4} .$$

THEOREM 5.2: For Algorithm 2,

$$\sigma^{min} = \frac{1}{4} \frac{r_{min}}{r_{max}} c' \quad ,$$

where

$$c' = 1 - \frac{1}{c_R} (1-e^{-c_R}) \quad .$$

Also, for simplicity we let $\sigma^{max} = 1$ for both Algorithms 1, 2. Let $\lambda_{ij}(\mathcal{A},\Gamma_t)$ be the probability that the poller of i is answered by j some time within $\Delta \subseteq I$ and i is in a watching window for j during

time interval Δ starting at t, where $\Delta \subseteq I$.

From the definitions of $\sigma_{ij}(\mathcal{A},\Gamma_t)$ and $\lambda_{ij}(\mathcal{A},\Gamma_t)$ it immediately follows that

$$\lambda_{ij}(\mathcal{A},\Gamma_t) = \frac{1}{2} \cdot \sigma_{ij}(\mathcal{A},\Gamma_t) \text{ for Algorithm 2}$$

$$\lambda_{ij}(\mathcal{A},\Gamma_t) = \frac{1}{2} \cdot \frac{1}{d_{t'}} \cdot \sigma_{ij}(\mathcal{A},\Gamma_t) \text{ for Algorithm 1}$$

where $d_{t'} = D_i$ (the outdegree of i) at the time instant $t' < t$ at which i selects a neighbour to ask.

Since by A1, $1 \leq d_{t'} \leq v$, it follows that

$$\frac{1}{2v}\ \sigma^{min} \leq \lambda_{ij}(\mathcal{A},\Gamma_t) \leq \frac{1}{2}\ \sigma^{max} \text{ for Algorithm 1}$$

and

$$\frac{1}{2}\ \sigma^{min} \leq \lambda_{ij}(\mathcal{A},\Gamma_t) \leq \frac{1}{2}\ \sigma^{max} \text{ for Algorithm 2} .$$

Let $\mathcal{C}$ 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. □

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

For each oracle $\mathcal{A} \in \mathcal{C}$ with Algorithm 1,

$$\frac{1}{2v}\ \sigma^{min} \leq \lambda_{ij}(\mathcal{A},\Gamma_t) \leq \frac{1}{2v}\ \sigma^{max} .$$

Let us define

$$\lambda^{min} = \frac{1}{2v}\ \sigma^{min} \qquad \text{(for Algorithm 1)}$$

and let

$$\lambda^{min} = \frac{1}{2}\ \sigma^{min} \qquad \text{(for Algorithm 2)}\ .$$

Also let

$$\lambda^{max} = \frac{1}{2v}\ \sigma^{max} \qquad \text{(for Algorithm 1 and oracles } \mathscr{C}\text{)}$$

$$= \frac{1}{2}\ \sigma^{max} \qquad \text{(for Algorithm 1 with all oracles but those of } \mathscr{C} \text{ and Algorithm 2 for all oracles)}$$

For all $i,j \in \Pi$ let $P_{ij}(k/(\mathscr{A},\Gamma_t))$ be the probability it *takes exactly* k *phases for process* i *to be answered by* j, given that $i \overset{\Delta}{\leftrightarrow} j$ for a time interval Δ starting at t, such that $\Delta \subseteq I$.

Lemma 5.1: For any oracle $\mathscr{A}$,

$$\lambda^{min}(1-\lambda^{max})^{k-1} \leq P_{ij}(k/\mathscr{A},\Gamma_t)) \leq \lambda^{max}(1-\lambda^{min})^{k-1}$$

Proof. It suffices to observe that the process of i be answered by j is a geometric stochastic process, with success probability bounded by $[\lambda^{min},\lambda^{max}]$. □

By using Lemma 5.1 and calculating the mean, we get

Lemma 5.2:

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

By Lemma 5.2 and known expressions for the tail of the geometric,

Lemma 5.3: $\forall \varepsilon\ 0 < \varepsilon < 1$, $\text{Prob}\{k > k_{max}(\varepsilon)\} \leq \varepsilon$,

where

$$k_{max}(\varepsilon) = \frac{\log(\lambda^{min} \cdot \varepsilon) - \log \lambda^{max}}{\log\ (1-\lambda^{min})}\ .$$

Let cv be the number of steps required for each phase. Lemmas 5.1 and 5.2 imply

Theorem 5.3: If τ is the response of the system, $\text{mean}(\tau) \leq cv\, r_{max}\, \lambda^{max}/(\lambda^{min})^2$ and if $\tau(\varepsilon)$ is the ε-error response, $\tau(\varepsilon) \leq cv\, r_{max} \cdot k_{max}(\varepsilon)$. Finally, Proposition (5.1) and the above theorem, together with theorems 5.1 and 5.2, imply the following corollaries, as claimed in Section 2E.

By using the value of λ^{max} for Algorithm 1 and oracles $\mathcal{O}$ and the value of λ^{min} for Algorithm 1, we get

$$\text{mean}(\tau) \leq 2cv^2 r_{max} \frac{\sigma^{max}}{(\sigma^{min})^2} \quad .$$

By using $\sigma^{max} = 1$, $\sigma^{min} = 1/4$ we have

Corollary 5.1: $\text{mean}(\tau) \leq 32cr_{max}v^2 = 0(v^2)$ for Algorithm 1. Since

$$k_{max}(\varepsilon) = \frac{\log\left(\frac{\varepsilon}{2v}\sigma^{min}\right) - \log\frac{1}{2v}\sigma^{max}}{\log\left(1 - \frac{1}{2v}\sigma^{min}\right)} \to 8v \log\frac{4}{\varepsilon}$$

for $v >> 0$.

Corollary 5.2: $\tau(\varepsilon) \leq 8c\, r_{max} v^2 \log\frac{4}{\varepsilon} = 0\left(v^2 \log\frac{1}{\varepsilon}\right)$

for Algorithm 1.

Also, by using the derived λ^{min}, λ^{max} for Algorithm 2,

Corollary 5.3: For Algorithm 2, ,

$$\text{mean}(\tau) \leq 16 \frac{r_{max}^3}{r_{min}} \frac{c}{(c')^2} \cdot v = 0(v) \quad .$$

Also, we get for Algorithm 2,

$$k_{max}(\varepsilon) = \frac{\log\left(\frac{\varepsilon}{2}\sigma^{min} - \log\frac{1}{2}\right)}{\left(\log 1 - \frac{1}{2}\sigma^{min}\right)}$$

$$= \frac{\log\left(\frac{\epsilon}{4}\ \frac{r_{min}}{r_{max}}\ c'\right)}{\log\left(1 - \frac{1}{8}\frac{r_{min}}{r_{max}}\ c'\right)}$$

Corollary 5.4: $\tau(\epsilon) \leq c r_{max} k_{max}(\epsilon) v$

$$= O\left(v \log \frac{1}{\epsilon}\right)$$

6. ESTIMATION OF σ^{min}

6.1 Analysis of the Non-insisting Algorithm 1

Algorithm 1 is *non-insisting*: if $i \in \Pi$ is in its asking mode, it chooses to ask a random j from the set of pollers to which i is willing to communicate. Observe that the probability of i,j communicating monotonically increases with the rate of the responding process and decreases with the rate of the asking process. However, if both i,j are willing to communicate, both of the i,j will attempt to establish communications. Because of this symmetry in the way processes ask, it follows that the worst case $\sigma_{ij}(\mathcal{A}, \Gamma_t)$ is when the oracle $\mathcal{A}$ sets the rates i,j equal, but not necessarily constant (recall that by our model of Section 2B, $\mathcal{A}$ determines the rates $\{R_t | t \geq 0\}$ at time 0 and $\mathcal{A}$ cannot influence the probabilistic choice of processes). Let $c_R v + 2$ be the fixed number of steps between phases, as given in Lemma 4.2. Let x be the number of steps by which i executes each phase before j, where $0 \leq x \leq c_R v$. Let S(S') be the event: j answers i is in its asking mode after (before) j assuming i,j both willing to communicate.

Then

$$\sigma_{ij}(\mathcal{A},\Gamma_t) = \frac{1}{2}\,\text{Prob}(s) + (1 - \frac{1}{2}\,\text{Prob}\,(S))$$

$$\cdot \frac{1}{2}\,\text{Prob}\,(S')$$

since Prob(j is in answer mode) = 1/2 and j may answer i either in j's current phase (in the phase where j got the question) or, at most in the next phase of j. Note that the length of the watching window of i does not allow for more than two phases of j (see Figure 1).

Let

$$f(x,y) = 1 - \left(1 - \frac{1}{y}\right)^{x/c_R} .$$

We have Prob(s) = $f(x,d_{t'})$ where $d_{t'}$ is the outdegree of j at time t' when j selected the v neighbours to answer. Let

$$g(x,y) = \frac{1}{2}\,f(x,y) + (1 - \frac{1}{2}\,f(x,y)) \cdot \frac{1}{2}\,f(c_R v - x - 7, y).$$

Then, since g is monotonically decreasing with y and x is bimodal,

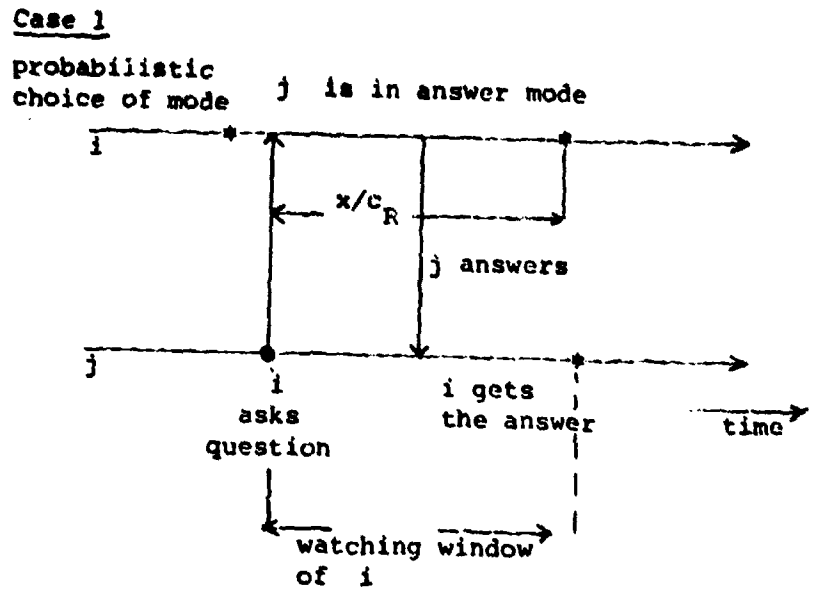
$$\sigma_{ij}(\mathcal{A},\Gamma_t) \geq \min_{\substack{0 \leq x \leq c_R v \\ 0 \leq y \leq v}} \{g(x,y)\}$$

$$\geq \frac{1}{4}$$

6.2 Analysis of the Insisting Algorithm 2

Algorithm 2 is *insisting*: poler i always asks that j which is the target of the first edge of G_t departing from i. The worst case σ_{ij} is where the oracle $\mathcal{A}$ sets the speed of the asking process to be $1/r_{min}$, and the speed of the answering process to be $1/r_{max}$. Let S and x and f(x,y) be as in 6.1. Because of the random wait in the beginning of each phase, x now can be any step in $[0,c_R v]$ with equal probability, and so

FIGURE 1


Case 1
probabilistic choice of mode
j is in answer mode
i
x/c_R
j answers
j
i asks question
i gets the answer
time
watching window of i
Case 2
j is in answer mode
x/c_R
j answers
i asks question
i gets the answer
time
watching window of i
NOTE:
a phase

Prob{j is in answering mode}

$$= \text{Prob\{j is not waiting\}} \cdot \text{Prob\{j is answering/j not waiting\}}$$

$$\geq \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \quad .$$

We have (by considering only case 1 of Figure 1)

$$\sigma_{ij}(\mathcal{A}, \Gamma_t) \geq \frac{1}{4} \text{ Prob (S)} \cdot \text{Prob(i's window was large enough}$$

to get the answer). However, due to the uniformly distributed random choice of x

$$\text{Prob(S)} \geq \sum_{Y=0}^{c_R v} f(Y, d_{t,}) \cdot \text{Prob}(x=Y)$$

$$\geq \sum_{Y=0}^{c_R v} f(Y,v) \cdot \frac{1}{c_R v}$$

$$\geq \frac{c_R v + 1}{c_R v} - \frac{1}{c_R}\left(1 - \left(1 - \frac{1}{v}\right)^{c_R v + 1}\right)$$

$$\geq c' \text{ for } v >> 0$$

where

$$c' = 1 - \frac{1}{c_R}\left[1 - e^{-c_R}\right] \quad .$$

Also, Prob(i's window is large enough to get the answer)

$$\geq \int_0^{c_R r_{min}} \frac{dt}{c_R v \, r_{max}}$$

$$= \frac{r_{min}}{r_{max}} \quad \text{for } v >> 0,$$

hence

$$\sigma_{ij}(\mathcal{A}, \Gamma_t) \geq \frac{1}{2} \frac{r_{min}}{r_{max}} c' \quad .$$

7. CONCLUSION

We have provided two real time implementations for the DCS system. A key assumption on 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 [Reif, Spirakis, 1981A], where we required processes to be tame at all times.

A referee has suggested a modification of our algorithms which 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 is 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 Ei can only be read by process i), so the idea requires additional flag variables. The modified algorithms will have worst case performance identical to those given in our paper.

In a further paper, [Reif, Spirakis, 1981B], we have relaxed our assumption of tameness. In that paper we require only bounds on the relative acceleration of ratios of speeds of neighbour processes. We propose there synchronization algorithms which have *relative* real time response, where communication is established with high probability between any pair of processes within constant number of steps of the *slowest* process. However, these algorithms are less efficient than those given in this paper. Also, we are applying our synchronization techniques to ADA for a relative real time implementation.

Acknowledgments

The authors 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

Angluin, D., "Local and Global Properties in Networks of Processors," *12th Annual Symposium on Theory of Computing*, Los Angeles, Cal., April 1980, pp. 82-93.

Arjomandi, E., M. Fischer, and N. Lynch, "A Difference in Efficiency between Synchronous and Asynchronous Systems," *13th Annual Symposium on Theory of Computing*, April 1981.

Bernstein, A.J.., "Output Guards and Nondeterminism in Communicating Sequential Processes," *ACM Trans. on Prog. Lang. and Systems*, Vol. 2, No. 2, April 1980, pp. 234-238.

Francez, N. and Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 373-379.

Hoare, C.A.R., "Communicating Sequential Processes," *Com. of ACM*, Vol. 21, No. 8, August 1978, pp. 666-677.

Lehmann, D. and M. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem," to appear in *8th ACM Symposium on Principles of Program Languages*, January 1981.

Lipton, R. and F.G. Sayward, "Response Time of Parallel Programs," Research Report #108, Dept. of Computer Science, Yale University, June 1977.

Lynch, N.A., "Fast Allocation of Nearby Resources in a Distributed System," *12th Annual Symposium on Theory of Computing*, Los Angeles, Calif., April 1980, pp. 70-81.

Rabin, M., "N-Process Synchronization by a 4 $\log_2$N-valued Shared Variable," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, October 1980, pp. 407-410.

Rabin, M., "The Choice Coordination Problem," Mem. No. UCB/ERL M80/38, Electronics Research Lab., Univ. of California, Berkeley, Aug. 1980.

Reif, J.H. and P.G. Spirakis, "Distributed Algorithms for Synchronizing Interprocess Communication in Real Time", Thirteenth Annual ACM Symposium on Theory of Computing, Milwaukee, WI, May 1981.

Reif, J.H. and P.G. Spirakis, "Unbounded Speed Variability in distributed systems", Techn. Rep. 14-81, Aiken Comp. Lab., Harvard University, Aug. 1981. Also to appear in the 9th ACM Symp. on Principles of Programming Languages, Albuquerque, NM, Jan. 1982.

Reif, J.H. and P.G. Spirakis, "A Real Time Resource Granting System", to appear. (Also appearing in preliminary form as Appendix II of "Distributed Algorithms..." referenced above.)

Schwartz, J., "Distributed Synchronization of Communicating Sequential Processes," DAI Research Report No. 56, Univ. of Edinburg, 1980.

Tonag, S., "Deadlock and Livelock-Free Packet Switching Networks," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Valiant, L.G., "A Scheme for Fast Parallel Communication," Technical Report, Computer Science Dept., Edinburg Univ., Edinburg, Scotland, July 1980.

APPENDIX
A REAL-TIME IMPLEMENTATION OF CSP

[Hoare, 1978] 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, 1980] describes an extension of CSP which allows both input command and output commands as guards. Here we briefly describe CSP with Bernstein's extension and present a real-time implementation of the synchronization constructs.

CSP Synchronization Constructs

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

$$[P_1 \| P_2 \| \ldots \| P_n] .$$

Each process has its own set of variables which are inaccessible to all other processes. The communication primitives are the *output command* $P_j!u$ that 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*

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

contains guards $G_1,\ldots,G_k$ and command lists $C_1,\ldots,C_k$. Each guard consists of a list of elements which 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 which is satisfied (to test that, 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*

$$*[G_1 \rightarrow C_1 \,\square\, \ldots \,\square\, G_k \rightarrow C_k]$$

results in the repeated execution of the alternative statement $[G_1 \rightarrow C_1 \,\square\, \ldots \,\square\, G_k \rightarrow C_k]$, 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_i with output commands $P_i!u$ by process P_j 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 system-wide constant, which may be fixed to any arbitrarily small constant on the interval $(0,1)$. We assume a *real time* DCS implementation with ε-error response time $\tau(\varepsilon)$. 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.

Our CSP implementation is *real-time* in the sense that there exists a positive integer ℓ (which is independent of the number of processes n) such that if in some alternative or repetitive statement S some guard G

is continuously satisfied for a time interval Δ of length $\geq \ell$ and if the processes of G and the process executing the statement are tame on Δ, then the command list associated with some satisfied guard is immediately executed with probability $\geq 1-\varepsilon$ and otherwise, a *failure exit* is always made immediately after a time interval of length ℓ. Therefore, we allow a failure exit with probability $<\varepsilon$, even though some guard may be satisfied.

To attempt to execute an output command $P_j!u$ in process P_i, P_i sets $P_i \rightarrow P_j$, indicating P_i is willing to communicate with P_j. Also, to attempt to execute an input comand $P_i?x$ 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 $G_1,\ldots,G_s$ where $s \leq v$. To execute the statement S, P_i first executes the booleans appearing in each guard. 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 $P_j \in R$ for a time interval of length $\ell = \tau(\varepsilon)$. At the first time that an appropriate communication is established between P_i and some willing process $P_j \in R$, P_i must immediately set $P_i \not\rightarrow P_j$, for all $P_{j'} \in R$ and then P_i must execute the command list associated with the now satisfied guard in the statement S. Otherwise, if no appropriate communication is established within time $\tau(\varepsilon)$, P_i must then exit the statement S with failure. Note that the probability of an incorrect failure exit is $<\varepsilon$.