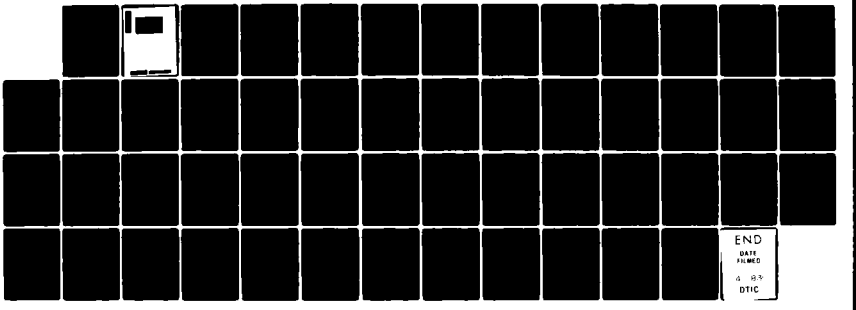
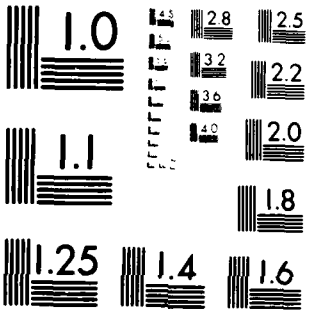
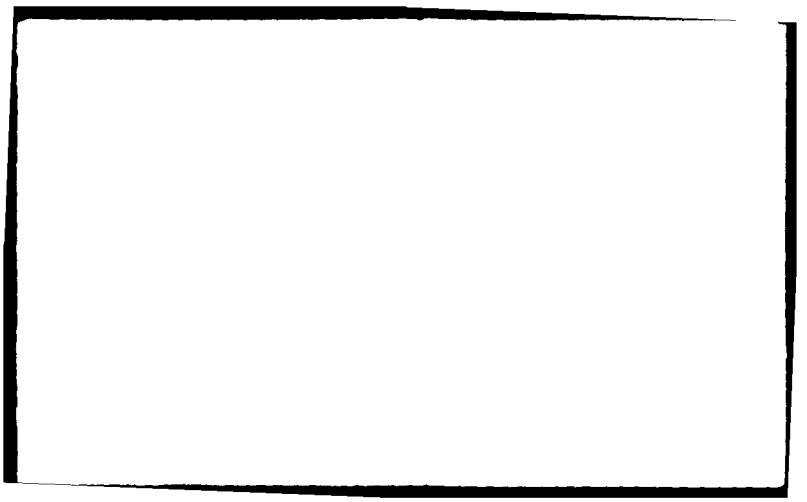
AD-A122 541 REAL-TIME SYNCHRONIZATION OF INTERPROCESS 1/1
COMMUNICATIONS(U) HARVARD UNIV CAMBRIDGE MA AIKEN
COMPUTATION LAB J H REIF ET AL. JUN 82 TR-25-82
UNCLASSIFIED N000014-80-C-0674 F/G 9/2 NL







REAL-TIME SYNCHRONIZATION OF INTERPROCESS COMMUNICATIONS

by

John H. Reif
and
Paul G. Spirakis

TR-25-82

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A122541	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Real-Time Synchronization of Interprocess Communications		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-25-82
7. AUTHOR(s) John Reif Paul Spirakis		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0674
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University Cambridge, MA 02138 Center for Research in Computing Technology Aiken Computation Lab.		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 June 1982 13. NUMBER OF PAGES 43
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

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)
distributed communication, handshake, synchronization, real time response.

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

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE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

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SUMMARY

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 neighbour 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 1 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 which does not guarantee the handshake property if used in an unsophisticated way.

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 as to require the fewest assumptions possible without considering specific systems.† A process is considered to be *tame* over a time interval Δ

(Continued on next page)

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 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. We feel the term *real time* is merited, for 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 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, in [Reif, Spirakis, 82B] and in our Appendix, 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.

REAL-TIME SYNCHRONIZATION OF INTERPROCESS COMMUNICATIONS*

by

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

June 1982

*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 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 neighbour 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 1 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 which does not guarantee the handshake property if used in an unsophisticated way.

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 as to require the fewest assumptions possible without considering specific systems.† A process is considered to be *tame* over a time interval Δ

† 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.

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 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. We feel the term *real time* is merited, for 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 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, in [Reif, Spirakis, 82B] and in our Appendix, 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.

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, whereas, in proofs of probabilistic algorithms, we need only consider those simple situations which 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 [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 [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 [Schwarz, 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 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: 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 communication synchronization problem. 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, Spirakis, 82B] 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.

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 occurring in practice, no such oracle 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.

We now introduce some graphs to describe precisely the DCS model. 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 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* $G_t = (\Pi, \vec{t})$ is a possibly infinite digraph with vertices Π and directed edges given by relation $\xrightarrow[t]{} \subseteq \Pi \times \Pi$ (See Figure 1B). 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 $i \xrightarrow[t]{} j$ implies $\{i,j\} \in E$ so i is willing to communicate only with processes which i is able to communicate with. Also, let $i \xleftrightarrow[t]{} j$ iff both $i \xrightarrow[t]{} j$ and $j \xrightarrow[t]{} i$. We use

$\xrightarrow[\Delta]{}$ and $\xleftrightarrow[\Delta]{}$ to denote that the willingness to communicate holds over time intervals. 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 $j \xrightarrow[\Delta]{} i$. 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 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 $\{j \mid i \xrightarrow[t]{} j\}$) 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.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 $i,j\in\Pi$ such that $\{i,j\}\in E$ we have a *communication port flag* $PORT_{i,j}$ (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 the communication port from i to j is closed at t). We assume 2-way communication between i,j is possible at any time that both $PORT_{i,j}$ and $PORT_{j,i}$ are simultaneously 1, but we make no particular assumptions (beyond A1 and A2 below) about this communication.

Let $i \xrightarrow[t]{\sim} j$ denote that $PORT_{i,j} = 1$ 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 $\overset{\sim}{\underset{t}{\longrightarrow}} \subseteq \Pi\times\Pi$. Let $i \overset{\sim}{\underset{t}{\longleftrightarrow}} j$ if both $i \overset{\sim}{\underset{t}{\longrightarrow}} j$ and $j \overset{\sim}{\underset{t}{\longrightarrow}} i$. If $i \overset{\sim}{\underset{t}{\longrightarrow}} j$ then we say i has *opened communication* with $j\in\Pi-\{i\}$ at t. If $i \overset{\sim}{\underset{t}{\longleftrightarrow}} j$ 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 $i \underset{t_1}{\longrightarrow} j$ and not $i \underset{t_2}{\longrightarrow} j$, $t_2 > t_1$, then $i \overset{\sim}{\underset{\Delta}{\longleftrightarrow}} j$ 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 willingness to communicate only after communication has been established and completed.

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

R1 $i \overset{\sim}{\underset{t}{\longrightarrow}} j$ only if $i \underset{t}{\longleftrightarrow} j$

R2 $\overset{\sim}{\underset{t}{\longrightarrow}}$ is a (partial) matching; if $i \overset{\sim}{\underset{t}{\longrightarrow}} j$ then not $i \overset{\sim}{\underset{t}{\longrightarrow}} j'$ 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 [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 nonnegative 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 adverse oracle $\mathcal{A}$ (possibly our scheduler's worst "enemy") *a priori* (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, to 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

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

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. 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, then $|\Delta'| \in [r_{min}, r_{max}]$, 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 $i \overset{\Delta}{\leftrightsquigarrow} j$ 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) $i \overset{\Delta}{\leftrightarrow} j$,

(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. Also, note that since 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 $\bar{\tau} = \max\{\text{mean}\{\tau_{\mathcal{A},R}\}$ for each oracle $\mathcal{A}$ and communication request $R\}$.

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 set of the values of the inverse functions of the cumulative distribution functions of $\tau_{\mathcal{A},R}$ at $1-\varepsilon$, for all $\mathcal{A}$ and R. 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 $\mathcal{A}$, $i \overset{\Delta}{\leftrightarrow} j$ 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 or processes Π_1, $\Pi_2 \subseteq \Pi$ such that $|\Pi_1|, |\Pi_2| \leq v$ and for the same interval Δ of length $\geq \tau(\varepsilon)$ and for all i in Π_1 and all j in Π_2, $i \overset{\Delta}{\leftrightarrow} j$. Then, each process i of Π_1 is guaranteed at least $(1-\varepsilon)^{|\Pi_2|}$ 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, $0 < \varepsilon \leq 1$, $\tau(\varepsilon)$ is a constant dependent only on v (assumed to be a constant upper bound on the outdegree of vertices of G_t). Note then that $\bar{\tau}$ is also bounded above by a fixed constant dependent only on v.

2.D. Preferential 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, which we only consider here, i distinguishes the *first target* of communication, $E_i(1)$, which i *prefers to* communicate with. (Process i may communicate with the other processes of E_i, but i prefers to communicate with $E_i(1)$.)

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

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

(1) $i \overset{'}{\underset{\Delta}{\rightarrow}} j$ and $j \overset{}{\underset{\Delta}{\rightarrow}} i$

(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, i.e. if $i \overset{}{\underset{\Delta'}{\leftrightsquigarrow}} j$ 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 $\bar{\tau}' = \max\{\text{mean}\{\tau'_{\mathcal{A},R}\}$ for all oracles $\mathcal{A}$ and communication requests $R\}$.

For each ε, $0 < \varepsilon \leq 1$, let the *ε-error preferential response* $\tau'(\varepsilon)$ 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 $1-\varepsilon$, for all $\mathcal{A}$ and R.

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 $\mathcal{A}$, ($i \overset{\Delta}{\rightarrow}' j$ and $j \overset{\Delta}{\rightarrow}' i$) implies i,j have successful communication sometimes within Δ, with probability $\geq 1-\varepsilon$.

The DCS implementation has *real time preferential response* if for all ε, $0 < \varepsilon \leq 1$, $\tau'(\varepsilon)$ 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 $\tau'(\varepsilon)$, any given proces $i \in \Pi$ 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 $\tau(\varepsilon)$ is given instead of $\tau'(\varepsilon)$.

PROPOSITION 2.1. *Let $\mathcal{A}$ be any oracle and Δ be any time interval of finite length $\geq \tau(\varepsilon)$ ($\tau'(\varepsilon)$ 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 $i \overset{t}{\rightsquigarrow} j$, then j is not willing to communicate with i sometime within Δ, with probability $\geq 1-\varepsilon$.*

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 $\Delta \geq \tau'(\varepsilon)$ such that for all $i \in \Pi_1$, $i \overset{\Delta}{\rightarrow}' j$ and

$j \overset{\Delta}{\rightarrow} i$. 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 $\geq 1-\varepsilon$. Furthermore, if $|\Pi_1| \leq v$, then, for all oracles $\mathcal{A}$, j will have successful communcation with all $i \in \Pi_1$ within Δ, with probability $\geq (1-\varepsilon)^{|\Pi_1|}$.

2.E. Results and Previous Work

The primary results of this paper are:

There is a *proper 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 $O(v^2 \log(\frac{1}{\varepsilon}))$.

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

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

(2) the ε-error preferential response $\tau'(\varepsilon)$ is $O(v \log(\frac{1}{\varepsilon}))$.

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). We allow processes to make probabilistic choices and show that our algorithms have real time response.

The best previous result is due to [Schwarz, 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 [Schwarz, 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 of processes (of length $\chi(H)$) can be formed in which each process waits for the next to reply. So the deterministic DCS implementation of [Schwarz, 80] has preferential response

time τ' lower bounded by $v^{*}\chi(H)$. 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 of 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, 80] gives a solution to a distributed resource allocation problem which in [Reif, Spirakis, 82B] is adopted to yield a DCS implementation with response time $v^{*}\chi(H)$. In [Reif, Spirakis, 82B] we show that a class of generalized resource allocation problems related to those of [Lynch, 80] may be solved in real time by our DCS implementation (with vanishingly small probability of violation of the real time property).

[Francez, Rodeh, 80] proposed a probabilistic synchronization algorithm which can be considered to be DCS implementation. An important difference between our implementation and theirs is that in the responding phase, in our algorithms, each process reponds 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.

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 "non-preferential" 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).

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 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 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 probability 1/2. This is used to ensure each process is in either phase half of the time or the average.

Informal Description of the Non-preferential Algorithm 1

In a responding phase, process i repeats a loop m times, where $m = (v+3) \cdot \frac{r_{max}}{r_{min}} + 1$. 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 $RESPOND_i(j)$. This procedure takes constant c_R number of steps. During these steps process i reads a flag to determine if j has recently been willing to talk to i and then sets a flag so as 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 to 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 $ASK_i(j)$. 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 half of the time on the average. This is important to the analysis). During procedure $ASK_i(j)$, 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 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 setting its flags to 0.

Informal Description for the Preferential Algorithm 2

Each process i executes forever the following loop:

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.

Formal Definitions of Algorithms 1 and 2

We now give Algorithms 1 and 2 in full detail.

Algorithm 1 (non-preferential 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 m 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 preferential implementation)

Program for process $i \in \Pi$

```
INITIALIZE_i( )
WHILE TRUE DO
  BEGIN
L1: CHOOSE a random b ∈ {0,1}
    IF b = 0 THEN
      BEGIN
      COMMENT: respond phase
      L3: FOR x = 1 to m 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
  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 $\{i,j\} \in H$ 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, $i \overset{t}{\rightsquigarrow} j$ 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 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* $B_{i,j}$ 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 (that is, 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 $OPEN\text{-}COM_i(j)$ 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 (namely, if i is tame but j is 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 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_i and it is in the *responding mode* when it executes the procedure RESPOND. If i is executing 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 lopp A3 of procedure ASK_i 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 *a responding 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} ∈ H  do
      BEGIN
        Q_ij ← 0
        A_ij ← 0
        B_ij ← 0
        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 parameters appearing in the procedures:

Let $c_R = 7 + (12 + 4\frac{r_{max}}{r_{min}})\frac{r_{max}}{r_{min}}$; this will be precisely the number of steps always required by procedure RESPOND (see justification in Lemma 4.3). Let $m = (v+3)\frac{r_{max}}{r_{min}} + 1$. 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 $c_P = 2 + 3\frac{r_{max}}{r_{min}}$; 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  ASK_i(target)
local a

BEGIN
   A1:  CURSTEP ← 0
   A2:  Q_{i,target} ← 1
        a ← 0
        B_{i,target} ← 1
        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} ← 0
        Q_{i,target} ← 0
        IF  a = 1  THEN
        BEGIN
   A4:  WHILE(A_{target,i} = 1  AND  CURSTEP < c_D)  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} ← 0
        WHILE  CURSTEP < c_A  DO a nonoperative step.
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_E  AND  q = 1)  DO  q ← Q_{asker,i}
          q ← (q  OR  B_{i,asker} = 1  OR  CURSTEP > c_E)
     B3:  A_{i,asker} ← 0
          IF  ¬q  THEN  B4:  OPEN-COM_i(asker)
B5:  WHILE  CURSTEP < c_R  DO a nonoperative step
END
```

```
PROCEDURE  OPEN-COM_i(j)
BEGIN
  PORT_ij ← 1
  DO  c_p - 2  nonoperative steps
  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,*

$$i \overset{t}{\rightsquigarrow} j \quad \textit{only if} \quad i \overset{t}{\leftrightarrow} j \ .$$

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 $B_{j,i} = 1$. 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, $i \overset{t'}{\rightarrow} j$ where

t' was the time of start of i's phase. By assumption (A2) then, $i \vec{t} j$.

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

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

In both cases, $i \overset{\sim\sim\rightarrow}{t} j$ implies $i \overset{\leftrightarrow}{t} j$. □

LEMMA 4.2. *For both algorithms,*

$$\overset{\leftrightsquigarrow}{t} \text{ is a partial matching.}$$

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

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

4.B. Timing Lemmas Which 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* $(12 + 4\frac{r_{max}}{r_{min}})$ *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 $\text{RESPOND}_j(i)$ 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 4 steps of i correspond to at most $4\, r_{max}/r_{min}$ 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 4 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 $\text{OPEN-COM}_i(j)$ 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 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 $2(r_{max}/r_{min}) \cdot r_{min} = 2\, r_{max}$ time, guaranteeing at least 1 step overlap of both processes. Thus, i,j have successful communication. Note that OPEN-COM takes c_p steps. 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. □

LEMMA 4.4. *For both algorithms, if* i,j *are tame on* Δ' *and* $i \overset{\Delta'}{\leftrightsquigarrow} j$ *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, then $|\Delta'| \leq c_p r_{max} = 2r_{max} + 3\ r_{max}^2/r_{min}$. □

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* $i \overset{\Delta'}{\leftrightsquigarrow} j$ *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 $i \overset{\Delta}{\leftrightsquigarrow} j$ 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'. □

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 timeouts within the procedures ASK and RESPOND and by the definition of $c_A = m \cdot c_R$. □

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* cvr_{max}.

5. PROBABILISTIC ANALYSIS OF THE RESPONSE TIME OF THE ALGORITHMS

Intuitively, in both algorithms, the ASK or Respond phases take $O(v)$ time each. In the worst case of the non-preferential 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 he is willing, communication will be achieved with probability bounded below a constant. Hence, we expect the average time of response of teh non-preferential 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 and fix throughout this section I to be an interval, starting at time t_0, of length at least 4 phases (i.e., $|I| \geq 4cvr_{max}$). Let Γ_{t_0} be the global system history up to t_0 and let $\mathcal{A}$ be a fixed oracle. Note that $(\mathcal{A},\Gamma_t)$ essentially specifies everything of the system's immediate future except "luck" $L_{t'}$, for $t' > t_0$. Consider two processes i,j such that $\{i,j\} \in H$ and $i \overset{I}{\leftrightarrow} j$ and i,j tame on I. Let $t_j = \min\{t > t_0 | j$ does a phase selection at $t\}$ and

$t_i = \min\{t > t_0 | i$ 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

$$t_m < t_{i1} < t_{i2}$$

and

$$t_m < t_{j1} < t_{j2} \ .$$

Let

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

The interval $(t_0, t_M]$ is called a *session* S of processes i,j. (See Figure 2). Note that a session has ≤ 3 phases of one of the processes i,j and hence its length is $\leq 3cvr_{max}$.

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 $\mathcal{C}$ 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.

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

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

PROPOSITION 5.2.

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

(Proof easy).

Note that for both algorithms the following holds:

PROPOSITION 5.3.

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

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

THEOREM 5.1. *For Algorithm 1, we have*

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

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 the phase Δ_j of j and of phase Δ_i of i which contains at least $c_R \cdot \frac{v}{2}$ steps of i. Since both phases Δ_j and Δ_i were selected in the session S after t_0, during probability of j asking i during Δ_j is $\geq 1/2v$ and the probability of i answering during Δ_i is $1/2$. Our theorem follows by multiplying these probabilities. □

THEOREM 5.2. *For Algorithm 2, we have*

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

Proof. Same as in 5.1. The only difference is that now j insists on asking i and hence prob(j asking i given j in asking phase and preferring i) = 1. □

Let

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

and

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

THEOREM 5.3.

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

Proof. By Theorems 5.1, 5.2, Propositions 5.2 and 5.3 and Corollary 5.1. □

Note that our lower bounds on $x_{ij}(\mathcal{A},\Gamma_{t_0},E)$ and prob(E 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 it takes exactly k sessions for processes i,j to succeed in establishing communication, given that $i \overset{\Delta}{\leftrightarrow} j$ (or $i \overset{\Delta}{\rightarrow}' j$ and $j \overset{\Delta}{\rightarrow} i$) for a time interval Δ starting at t_0, such that $\Delta \subseteq I$.

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

$$P_{ij}(k|(\mathcal{A},\Gamma_{t_0})) = (1-\sigma_{ij}(\mathcal{A},\Gamma_{S_0}))\cdot(1-\sigma_{ij}(\mathcal{A},\Gamma_{S_1}))$$

$$\ldots (1-\sigma_{ij}(\mathcal{A},\Gamma_{S_{k-2}}))\cdot\sigma_{ij}(\mathcal{A},\Gamma_{S_{k-1}}) .$$

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.

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

and, by known expressions about tails of geometrics, we get

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

$$\text{Prob}\{k > k_{max}(\varepsilon)\} < \varepsilon$$

where

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

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 $\tau(\varepsilon)$ *is the* ε*-error response, then*

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

By using $c = \left(1 + \frac{3}{v}\right) \frac{r_{max}}{r_{min}} \cdot c_R$ and the σ_{max}, σ_{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

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

implying

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

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

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

implying

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

6. 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 E_i can only be read by process i), so the idea requires additional flag variables. Those modified algorithms will have worst case performance identical to those given in our paper.

In a further paper, [Reif, Spirakis, 1982], 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. 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, in the Appendix of [Reif, Spirakis, 1982], 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; 9th ACM Symp. on Principles of Programming Languages, Albuquerque, NM, Jan. 1982.

Reif, J.H. and P.G. Spirakis, "Real Time Resource Allocation in a Distributed System", in ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Canada, August 1982(B).

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

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 $T(\varepsilon)$. 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 k (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 $<\varepsilon$.

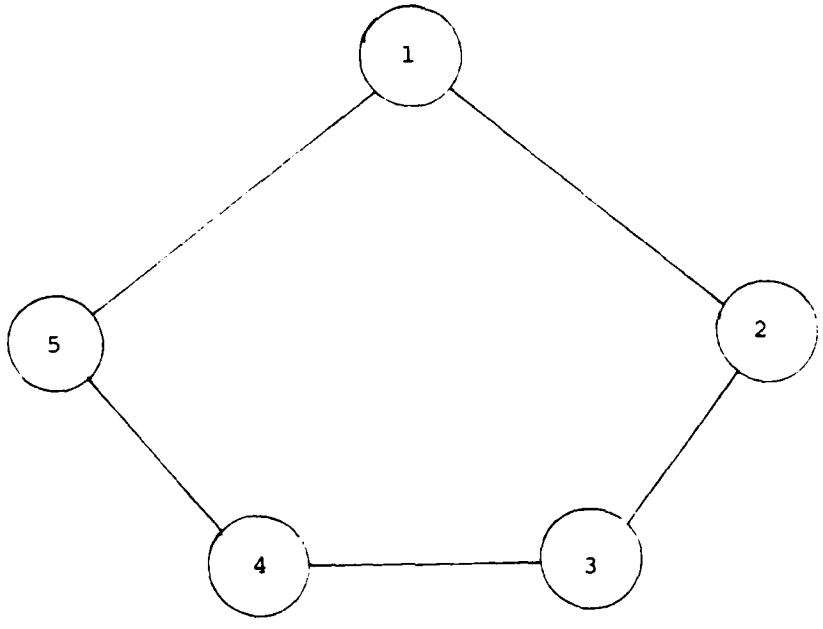


Figure 1A. The connections graph H in the case of a ring network of five processes.

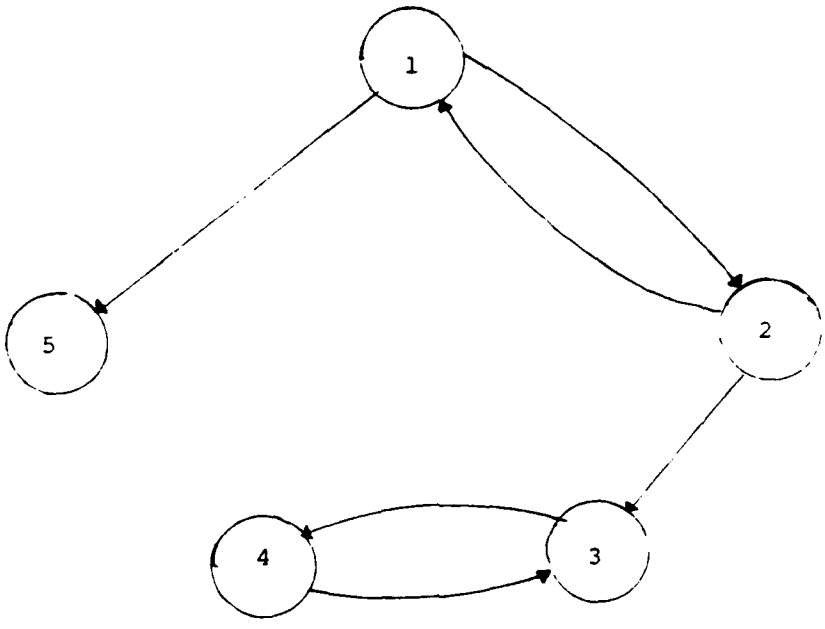


Figure 1B. The willingness digraph G_t

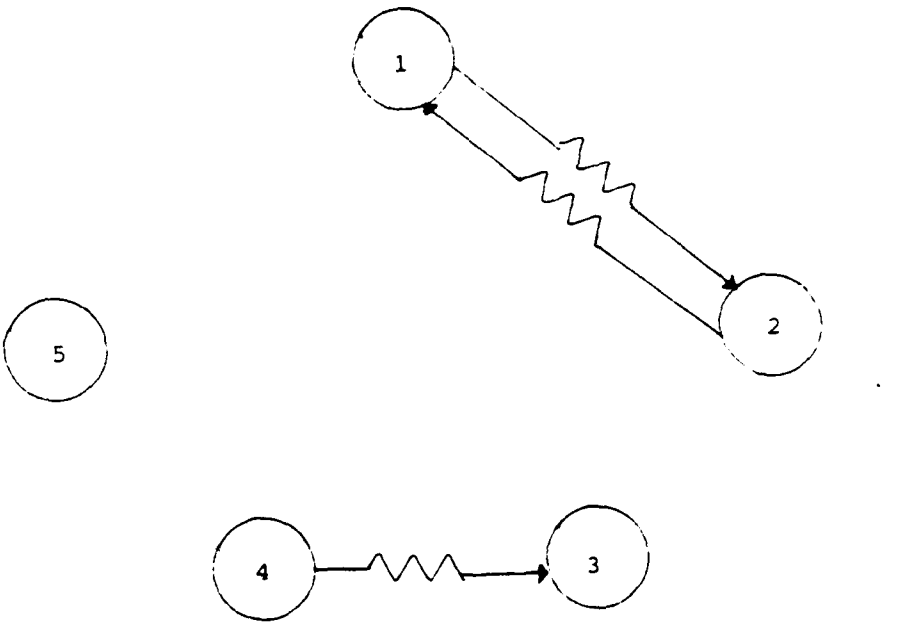


Figure 1C. The open-channel graph M_t.

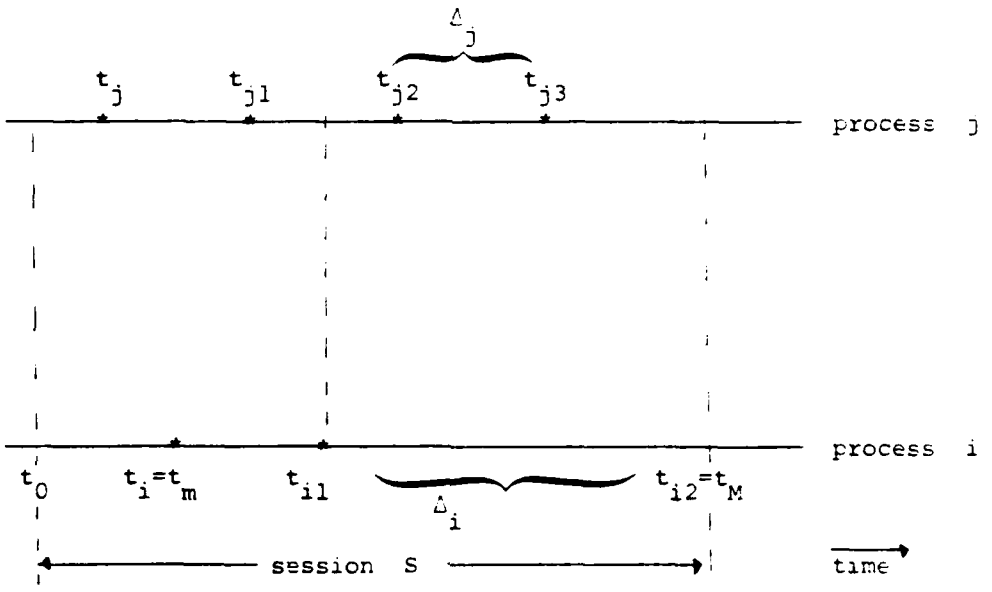


Figure 2. A session S of i,j and one of the possible orderings of events.

*: indicates the start of a phase

Δ_j is a phase of j

Δ_i is a phase of i

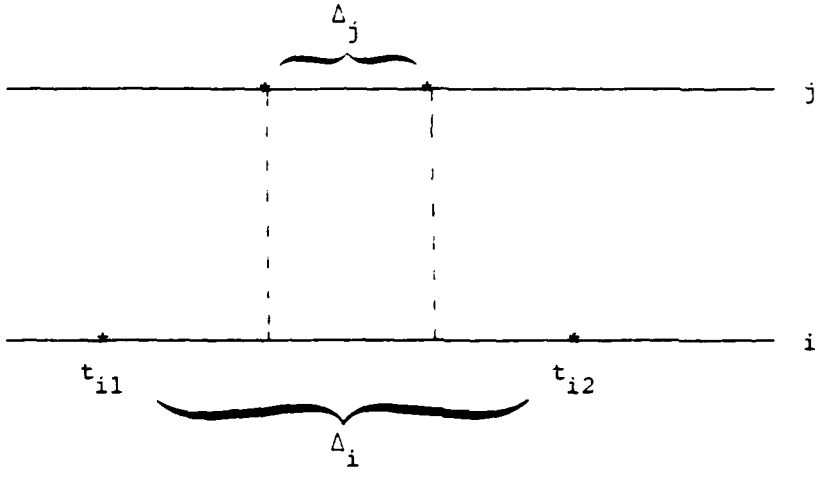


Figure 3A. A case of phase overlap where phase Δ_i of i contains at least half of the phase Δ_j of j.

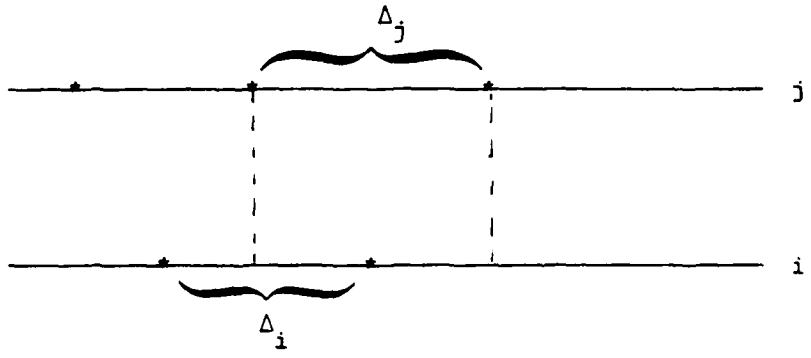


Figure 3B. Phase Δ_j of j contains at least half of phase Δ_i of i.


DATE
ILME