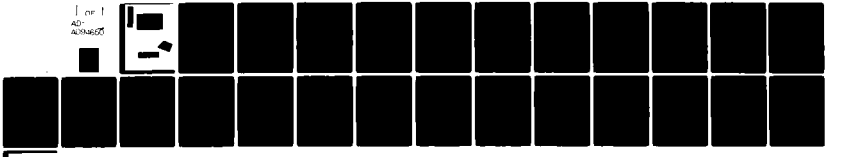
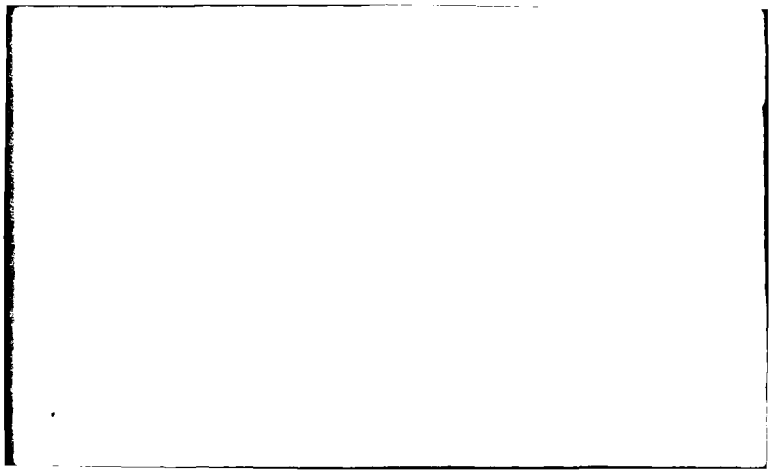
AD-A094 650 HARVARD UNIV CAMBRIDGE MASS AIKEN COMPUTATION LAB F/G 9/2
DISTRIBUTED ALGORITHMS FOR SYNCHRONIZING INTERPROCESS COMMUNICA--ETC(U)
1980 J H REIF, P SPIRAKIS N00014-80-C-0647
UNCLASSIFIED TR-23-80 NL



END
DATE FILMED
3-81
DTIC



DISTRIBUTED ALGORITHMS FOR SYNCHRONIZING INTERPROCESS COMMUNICATION WITHIN REAL TIME

BY

John H. Reif
Paul Spirakis

TR-23-80

Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time

by

John H. Reif and Paul Spirakis

Aiken Computation Lab., Harvard University, Cambridge, MA 02138

This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024 and the Office of Naval Research Grant N00014-80-C-0647.

Distributed Algorithms for Synchronizing Interprocess Communcation within Real Time

Abstract

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. With very few assumptions (the speeds of processes may vary in time within fixed arbitrary bounds, and the processes may be willing to communicate with a time varying set of processes (but bounded in number), and *no* probability assumptions about system behavior) we show our synchronization algorithms have *real time response:*

if at any time a pair of processes are mutually willing to communicate, they establish communication within a constant time interval, with high likelihood (for the worst case behavior of the system).

Our communication model and synchronization algorithms are quite robust and can be applied to solve a large class of resource synchronization problems, as well as implement Dijkstra's CSP in real time.

1. Introduction

Recently, [Rabin, 80], [Lehman and Rabin, 81], and [Francez,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 probabilistic choice) leads to considerably *simpler algorithms* and *shorter proofs*, perhaps because 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].

Previously [Schwartz, 80] proposed a deterministic synchronization algorithm for implementing CSP [Hoare, 78] on a fixed acyclic distributed network. Also [Lynch, 80] gave a algorithm for resource synchronization problems which may be adopted to communication synchronization. Both algorithms are considerably less time efficient than our proposed algorithm (for a specific comparison of time preference, see Section 2E).

[Francez and Rodeh, 80] also propose a probabilistic solution to synchronization of communication, but make no consideration to the time efficiency of these solutions.

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 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, irregardless of probabilistic choice. Section 5 gives a probabilistic analysis of the performance of our algorithms. Although our algorithms are quite simple we have taken considerable effort in their design to improve their expected time performance.

We have also included two Appendices: (i) the first appendix applies our synchronization algorithms to implement resource granting systems similar to those of [Lynch, 80] and we give an example application to "hasty dining philosophers." (ii) The other appendix gives more of the details of the probabilistic analysis of our algorithms.

2. Our Model for a Distributed Communication System (DCS), and Its Complexity Measures

2A. The Model

Let $\Pi = \{1,2,\ldots\}$ be a fixed, (possibly infinite) collection of *processes*. We assume a (global) *time* t, on the nonnegative real line $[0,\infty)$, whereby events of the system are totally ordered. The processes of Π are *asynchronous*; their speeds may vary over time and they have no access to any global clock giving the time. Each process $i \in \Pi$ consists of a constant size set of *synchronous subprocesses* (i.e., with the same speeds), of which we distinguish:

(1) The *primary subprocess* (the *director*) of i

(2) The *implementing subprocess* (the *poller*) of i.

Intuitively, the *director* of i wishes at various times to communicate with directors of various processes in $\Pi - \{i\}$, and the director of i has no means (i.e., shared variables, etc.) of interacting with the directors of $\Pi - \{i\}$, except via the communication

system. All communication by the director of i is implemented by the *poller* of i. Thus systemwide communication is implemented by a distributed scheduler, the pollers of Π.

Note. The formal model DCS (for Distributed Communication System) described below, has been designed so that

(1) we are not concerned with the *values of the messages* communicated between the directors, 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.)

(2) 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 assume a (possibly infinite) undirected graph H (the *connections graph*) with vertices Π, and undirected edges given by symmetric relation $\Leftrightarrow \subseteq (\Pi \times \Pi) - \{(i,i): i \in \Pi\}$. Then $i \Leftrightarrow j$ denotes $i \in \Pi$ *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 the director of $i \in \Pi$ 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 $i \Leftrightarrow j$ if $i \xrightarrow[t]{} j$ (i.e., the director of i is willing to communicate only with processes with which i is able to communicate with). Also, let $i \xleftrightarrow[t]{} j$ if both $i \xrightarrow[t]{} j$ and $j \xrightarrow[t]{} i$. (I.e., $i \xleftrightarrow[t]{} j$ denotes the directors of 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 $j \xrightarrow[\Delta]{} i$. ($\xrightarrow[\Delta]{}$ and $\xleftrightarrow[\Delta]{}$ denote 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, specified by the director of i and read only by the poller of i.

We assume the following:

(A1) 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 upper bounded by v.

(A2) We assume given fixed real constants r_{min}, r_{max} $(0 < r_{min} \leq r_{max})$ such that the *speed* (*steps* per real time unit) of each process $i \in \Pi$ at any time $t \geq 0$ is on the interval $[1/r_{max}, 1/r_{min}]$.

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

(A3) We assume that two-way communication between any two directors of processes $i,j \in \Pi$ requires only one step of i and j. (Thus, the directors of i,j are assumed to communicate in short "bursts." In practice, this assumption can be easily circumvented to allow long intervals of one-way communication between directors between the times that two-way communication between i,j is established.)

2B. Implementations of a DCS

An *implementation of a* DCS assigns a fixed program to each of the pollers of Π. The implementation is *symmetric* if the poller's 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 $\overset{t}{\longrightarrow} \subseteq \Pi \times \Pi$. Then $i \overset{t}{\longrightarrow} j$ denotes (the poller of) i *opens communication* with $j \in \Pi - \{i\}$ at time t.

Let $i \overset{t}{\longleftrightarrow} j$ if both $i \overset{t}{\longrightarrow} j$ and $j \overset{t}{\longrightarrow} i$. Then, $i \overset{t}{\longleftrightarrow} j$ denotes i,j *achieve mutual communication* at time t. (Also, we extend the notation to intervals Δ on $[0,\infty]$ as for G_t).

For each $i,j \in \Pi$ such that $i \leftrightarrow j$ we assume a *communication port* $\text{PORT}_{i,j}$ (controlled by the poller of i) which is *open* at time $t \geq 0$ if $i \overset{t}{\longrightarrow} j$ (i.e., the poller of i has opened communication with j) and *closed* otherwise. Thus, $i \overset{t}{\longleftrightarrow} j$ if and only if $\text{PORT}_{i,j}$ and $\text{PORT}_{j,i}$ are simultaneously *open* at time t. We assume 2-way communication between i,j is possible at any time $\text{PORT}_{i,j}$ and $\text{PORT}_{j,i}$ are simultaneously *open*, but we make no particular assumptions (beyond A3) about this communication.

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

(R1) $i \overset{t}{\longrightarrow} j$ only if $i \overset{t}{\longleftrightarrow} j$ (i.e., the poller of i *opens* communication with j only when both i and j are simultaneously willing to communicate).

(R2) We require that $\overset{t}{\longrightarrow}$ be a (partial) matching: if $i \overset{t}{\longrightarrow} j$ then neither $j' \overset{t}{\longrightarrow} i$ nor $i \overset{t}{\longrightarrow} j'$ for any $j' \in \Pi - \{i\}$ (i.e., i does not open communication with more than one process at a time).

(R3) $\forall i,j \in \Pi$ and $\forall$ time intervals Δ of maximal length $(\leq r_{max}^2/r_{min})$, if $i \overset{\Delta}{\longrightarrow} j$ then $i \overset{t}{\longleftrightarrow} j$ for some t in Δ.
(Thus the poller of i opens communication for the least possible time and furthermore it never opens communication with j without *achieving* communication with j.)

(R4) 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). Let this also hold for the subprocesses (in particular, the *director* and *poller*) of each process $i \in \Pi$.

(R5) Furthermore, the communication graph G_t is distributedly stored. We assume for each $i \in \Pi$, an integer variable D_i and integer array E_i of length v (written by the director of i, read only by the poller of i) such that for each $t \geq 0$.

(i) D_i is the outdegree of i in G_t (by A1, $D_i \leq v$).

(ii) $E_i(1),\ldots,E_i(D_i)$ are the targets of the edges of G_t departing from i. Also, we assume the directors of Π have no other flags which may be read by other processes.

1C. Global State of the DCS

For each $t \geq 0$, let R_t be a mapping $\Pi \to 2^{\mathbb{R}}$ giving the speed of each process of Π at time t. We assume the speed schedule $R = \{R_t \mid t \geq 0\}$ is chosen by an oracle $\mathcal{A}$ (our worst "enemy") at time $t = 0$. Also, we assume for each $t \geq 0$, $\mathcal{A}$ *chooses* (for the directors 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 pollers of Π, (which attempt a distributed synchronization of the DCS, depending on our given implementation).

In addition, we allow the pollers of Π to freely make probabilistic choices as in [Rabin, 80]. Let L_t = the probabilistic choices made by pollers 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 game, where the omnipotent oracle $\mathcal{A}$ plays against the team of pollers of Π. We wish *measures* of *the success* of the pollers of Π.

1D. Complexity Measures on DCS Implementations

We define here complexity measures for the performance of a DCS implementation.

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

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* $T(\varepsilon)$ (note: this is a function, not a random variable) be the upper bound on the inverse of the cumulative distribution function of $\tau_{\mathcal{A}}$. Thus, $\forall$ time interval $\Delta \geq T(\varepsilon)$, $\forall_{i,j} \in \Pi$, $\forall$ oracle $\mathcal{A}$ $i \overset{\Delta}{\longleftrightarrow} j$ implies $i \overset{t}{\longleftrightarrow} j$ for some $t \in \Delta$, with probability $\geq 1 - \varepsilon$.

The DCS implementation is *real time* if $\forall\varepsilon$, $0 \le \varepsilon < 1$, $T(\varepsilon)$ is a constant, independent of any parameters of H (except v, which we assumed to be a constant upper bound on the outdegree of vertices of G_t). Note that $\bar{\tau}$ is bounded by a fixed constant.

For all $\Pi_1, \Pi_2 \subseteq \Pi$ and time intervals Δ, let $\Pi_1 \xleftrightarrow[\Delta]{} \Pi_2$ if $\forall i \in \Pi_1$, $j \in \Pi_2$, $i \xleftrightarrow[\Delta]{} j$ (i.e., this implies all managers of Π_1 and Π_2 are mutually willing to communicate during Δ).

PROPOSITION 1: $\forall$ oracle $\mathscr{A}$, $\forall$ time interval Δ such that $|\Delta| > T(\varepsilon)$ $\forall \Pi_1, \Pi_2 \subseteq \Pi$, if $\Pi_1 \xleftrightarrow[\Delta]{} \Pi_2$ then $i \xleftrightarrow[\Delta]{\wedge\wedge\wedge} j$ with probability $\ge 1 - \varepsilon$, $\forall i \in \Pi_1, j \in \Pi_2$ (i.e., connection is established between all directors of Π_1 and Π_2 within the time interval Δ, with probability $\ge 1 - \varepsilon$.

Thus, real time response implies that the DCS implementation has a *very robust* type of *fairness*.

We also consider the cases where the director of 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, the director of i distinguishes the *first target of communication*, say $E_i(1)$. For each $t \ge 0$, $\xrightarrow[t]{}'$ is the relation on $\Pi \times \Pi$ sucn that $\forall i,j$ $i \xrightarrow[t]{}' j$ iff $E_i(1) = j$ at time t (and let $i \xrightarrow[\Delta]{}' j$ if $i \xrightarrow[t]{}' j$ $\forall t \in \Delta$).

Let the *insisting response time* of a DCS implementation be the random variable $\tau'_{\mathscr{A}}$, for each oracle $\mathscr{A}$, giving the length of a time interval required for the establishment of communication, with the *first target*. Let

$$\bar{\tau}' = \max\{\text{mean}(\tau'_{\mathscr{A}})/\text{all oracles } \mathscr{A}\} .$$

For each ε, $0 \le \varepsilon < 1$ let the ε-*error insisting response* $T'(\varepsilon)$ to be an upper bound on the inverse of the cumulative distribution function of $\tau_{\mathscr{A}}$. Thus, for every interval $\Delta \ge T'(\varepsilon)$, for every $i,j \in \Pi$, for every oracle $\mathscr{A}$, $(i \xrightarrow[\Delta]{}' j$ and $j \xrightarrow[\Delta]{} i)$ implies $i \xleftrightarrow[\Delta]{\wedge\wedge\wedge} j$ for some $t \in \Delta$ with probability $\ge 1 - \varepsilon$. The insisting DCS implementation is *real time* if $\forall \varepsilon$ $0 \le \varepsilon < 1$, $T'(\varepsilon)$ is a constant, independent of any parameter of H (except v). Obviously, $\bar{T}'$ is bounded by a fixed constant if we have real time insisting DCS implementation.

It is useful to observe, given $T'(\varepsilon)$, 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.

PROPOSITION 2: $\forall$ oracles $\mathscr{A}$, $\forall$ time intervals $\Delta \ge T'(\varepsilon)$ and $\forall i,j \in \Pi$, if $i \longrightarrow' j$ but there is no $t \in \Delta$ such that $i \xleftrightarrow[t]{\wedge\wedge\wedge} j$

then $j \not\xrightarrow[\Delta]{} i$ (i.e., j is not willing to communicate with i sometime during Δ) with probability $\ge 1 - \varepsilon$.

This proposition may be used for *timing out insisting* requests to communicate with a specific process.

2E. Results and Previous Work

The primary results for this paper are:

COROLLARY 1: There is a *real time implementation* of DCS such that

(1) the worst case mean response $\bar{T} = O(v^3)$

(2) the ε-error response $T(\varepsilon)$ is $T(\varepsilon) = O(v^3 \log(1/\varepsilon))$.

Furthermore, a more sophisticated probabilistic analysis of our implementation, not given in this paper, implies a worst case mean response time $O(v^2)$.

COROLLARY 2: There is a *real time insisting implementation* of DCS such that

(1) worst case mean insisting response $\bar{T}' = O(v)$

(2) the ε-error response $T'(\varepsilon)$ is $T'(\varepsilon) = O(v \log(1/\varepsilon))$.

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

Our implementations are *proper* (satisfy restrictions R1-R4), symmetric, and are completely independent of the connection graph H (H may be any finite or infinite graph with finite valence).

The best previous result is due to [Schwartz, 79] 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'. 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 symmetric*, since processes are required to know their color in H'.

Also [Lynch, 80] gives a solution to a distributed resource allocation problem, which may be adopted to yield a DCS implementation with response time $v^{\chi(H)}$.

In Appendix I we show that a class of generalized resource allocation problems related to those of [Lynch, 80] may be efficiently solved by our DCS implementation. Our innovation, which results in real time response, is to allow pollers to make probabilistic choices as in [Rabin, 80].

3. The Implementation of a DCS

To implement a DCS, we must give an algorithm for the poller of each process in Π. We present here two such implementations. Both satisfy restrictions R1-R4, required by proper implementations, and both are symmetric: (i.e., each poller has the same algorithm regardless of its position in the graph H).

Pollers 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.

3A. Informal Description of the Pollers' Algorithms

In both Algorithms, the poller repeatedly throws a fair coin and then executes a *phase*. Each phase is either *asking* or *answering* and is chosen by the coin throw with probability 1/2.

Informal Description of Algorithm 1 (for the noninsisting implementation)

```
WHILE TRUE DO
   BEGIN
      choose  b  from  {0,1}  with  prob 1/2
      if  b = 0  then
         BEGIN  ("answer" phase) randomly sample  v  of the pollers your director wants
            to communicate with.  "Answer" each of these pollers that "asked"
         END
      else
         BEGIN  ("ask" phase) choose at random a poller  j  your director wants to
            communicate with.  "Ask"  j  for a constant number of steps
         END
   END
OD
```

Informal Description of Algorithm 2 (the insisting implementation)

```
WHILE TRUE DO
   BEGIN
      choose  w  at random form  [0,cv]  (c  is a constant)
      wait for [w] steps
      choose  b ∈ {0,1}  with  prob 1/2
```

```
if b = 0 then
   BEGIN
     ("answer" phase) sample (at random) v of the pollers your director wants
     to communicate with. "Answer" them that "asked"
   END
else
   BEGIN
     ("ask" phase) "Ask" E_i(1) for a constant number of steps (when your
     director insists on communicating with E_i(1))
   END
END
OD
```

3B. Details of the Poller's Algorithms

Numerous important details are hidden from the above informal description of the poller's algorithms.

For each $i,j \in \Pi$ such that $i \Leftrightarrow j$, there are three *flags* (boolean variables) Q_{ij}, A_{ij}, B_{ij} which are written only by i and read only by (the poller of) 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. Also, at every time $t \geq 0$, $E_i(1),\ldots,E_i(D_i)$ is the list of targets of edges of G_t departing from $i \in \Pi$, and $D_i \leq v$. These variables must be locked by the poller of i during the "ask" or "answer" phase so that they will be unmodified by the director of i during that phase. (Note that since for each $i \in \Pi$ the director of i is synchronized with the poller of i, the director of i need *not* busy wait during these phases.)

The Algorithms 1 and 2 require the number of steps in each phase to be a constant v.c (where c is a constant).

A certain number of no-ops is executed to achieve this.

3C. The Algorithms 1 and 2

We now give Algorithms 1, 2 in full detail.

Algorithm 1 (noninsisting implementation)
Program for poller $i \in \Pi$

```
INITIALIZE_i( );
WHILE TRUE DO
   BEGIN
      L1:  LOCK  E_i, D_i
      L2:  CHOOSE  b ∈ {0,1}  with  prob 1/2
   IF  b = 0  THEN
            BEGIN (answer phase)
               L3:  FOR  x = 1  to  v  DO
                  BEGIN
                     choose a random  m ∈ {1,...,D_i}
                     RESPOND_i(E_i(m));
                  END
            END
   ELSE
            BEGIN (ask phase)
               L4:  choose at random  m ∈ {1,...,D_i}
                    ASK_i(E_i(m))
            END
            UNLOCK E_i,D_i
   END
OD
```

Algorithm 2 (the insisting implementation)
Program for poller $i \in \Pi$

```
INITIALIZE_i( )
WHILE TRUE DO
   BEGIN
      L1:  LOCK E_i, D_i
           choose  w  at random from  [0,cv]
           DO  w  no-ops
      L2:  choose  b ∈ {0,1}  with  prob 1/2
```

```
      IF  b = 0  THEN
         BEGIN  (answer phase)
         L3:  FOR  x = 1  to  v  DO
              BEGIN
                 choose random  m ∈ {1,...,D_i}
                 RESPOND_i(E_i(m))
              END
         END
      ELSE
         BEGIN  (ask phase)
         L4:  ASK_i(E_i(1))
         END
      UNLOCK E_i, D_i
   END
OD
```

The variables of the poller i are initialized as follows:

```
   INTIALIZE_i( );
      BEGIN
         for all  j ∈ Π  such that  i ⇔ j  do
            BEGIN
               Q_ij ← A_ij ← 0
               B_ij ← 1
               PORT_ij  set to closed
            END
      END
```

In the following two procedures, we assume a register CURSTEP which gives the current number of the steps executed by the poller of i. (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.)

Furthermore, we define the constants appearing in the procedures below: Let

$$c_0 = \frac{r_{max}}{r_{min}}$$

$$c_R = 20 + 6 \cdot c_0$$

$$c_A = v(1 + c_R)$$

$$c_1 = c_A - (8 + 7c_0)$$

```
PROCEDURE  ASK_i(target)

     BEGIN
     A1:  x_0 ← CURSTEP
     A2:  Q_i,target ← 1;
          a ← 0;
          B_i,target ← 0;
     A3:  WHILE  CURSTEP - x_0 < c_1  AND  a = 0  DO
             BEGIN  a ← A_target,i  END
          IF  CURSTEP - x_0 = c_1  AND  a = 0  THEN  B_i,target ← 1;
        IF a = 1 THEN
             BEGIN
                Q_i,target ← 0;
          A4: WHILE A_target,i = 1  DO  no-op
          A5: OPEN-COM_i(target)
             END
          Q_i,target ← 0
          B_i,target ← 1
          WHILE  CURSTEP - x_0 < c_A  DO  no-op
     END


PROCEDURE  RESPOND_i(asker)

     BEGIN
        x_0 ← CURSTEP
        L1:  IF  Q_asker,i = 1  THEN
               BEGIN
                  A_i,asker ← 1;
               L2: WHILE  Q_asker,i = 1  DO  no-op
               IF  B_asker,i = 1  THEN
                                  BEGIN
                                    A_i,asker ← 0
                                  END
               ELSE
                                  BEGIN
                                  L3:  A_i,asker ← 0
                                  L4:  OPEN-COM_i(asker)
                                  END
               END
          RG:  WHILE  CURSTEP < c_R + x_0  DO  no-op
     END
```

PROCEDURE OPEN-COM$_i$(j)
    BEGIN
        Set PORT$_{ij}$ to open
        Do c_0 no-ops
        Set PORT$_{ij}$ to closed
    END

(Note. During the c_0 no-ops, the director of i communicates with the director of j.)

4. Correctness Properties of the Poller's 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*, irregardless of probabilistic choice. We show restrictions R1-R4 are satisfied by our implementations, and thus they are proper. (Of course, we assume either all the pollers in Π execute Algorithm 1, or they all execute Algorithm 2.)

4A. Definitions and Terminology

In the following, for brevity, we allow a poller of a process $i \in \Pi$ to be identified with the process i. A poler is in the *asking mode* when it executes procedure ASK(j), and it is in the *answering 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 poller* 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 A2 with a = 1.

A *phase* of the poller algorithm consists of the steps between random choices of the variable $b \in \{0,1\}$. If $b = 0$ the poler is in *an answering phase* and else it is in an *asking phase*.

4B. Correctness Proofs

Using the above terminology, it is easy to prove three lemmas stating that restrictions R1, R2 and R3 are satisfied for both Algorithms.

The following is a key lemma whose proof is given in detail. It holds for both Algorithms.

LEMMA 4.1: If $i \in \Pi$ is in its watching window for j and is answered by j, then i,j *establish communication* within $\leq c_E$ steps of i, and $\leq c_E r_{max}$ time units, where $c_E = 8$. Furthermore, restriction R3 is satisfied.

Proof. If i exits the A3 loop at time t_0, then (since no process but j can wrtie in $A_{j,i}$) at the same time j must be executing RESPOND(j) at the L2 loop. Since i will arrive at A_4 within 4 of its steps, then by at most time $t_0 + 4 \cdot r_{max}$, i sets $Q_{i,j}$ to 0 and enters the A4 loop. At this time, j exits the L2 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 R4 and set $A_{j,i}$ to 0 at most time $t_0 + 7 \cdot r_{max}$. At the same time, i exits the A4 loop. Then, within a time interval of length $\leq r_{max}$, i opens $PORT_{ij}$ and j opens $PORT_{ji}$. Each of i,j keep their respective ports open for r_{max}/r_{min} of their steps (which are not synchronized), on time intervals Δ_1 and Δ_2, respectively. Thus $i \xrightarrow{\Delta_1} j$ and $j \xrightarrow{\Delta_2} i$ and $r_{min} \leq |\Delta_i| \leq r_{max}^2/r_{min}$ for $i = 1,2$. Hence, there is some time $t \leq t_0 + 8r_{max}$ such that $t \in \Delta_1 \cap \Delta_2$ and at time t i,j establish communication $(i \xleftarrow[t]{} j)$. QED

Thus we have

THEOREM 4.1: The Algorithms 1 and 2 each satisfy restrictions R1-R4 and thus are proper.

The following Lemma is useful in the probabilistic analysis of the next section.

LEMMA 4.2: If $i \in \Pi$ executes procedure ASK, then it requires precisely c_A steps of i. Also execution of RESPOND by i requires precisely c_R steps of i. Also, each phase of Algorithm 1 (Algorithm 2) requires exactly $c_1 v$ ($c_2 v$) steps, for fixed constants $c_1, c_2 > 0$.

Proof. By [illegible] process can ever be blocked in the busy-wait loops of A3 or L2. Lemma 4.2 then follows by counting steps. QED

5. Probabilistic Analysis of the Response Time of the Poller's Algorithm

The analysis done here holds for both Algorithms 1 and 2 (except that they differ in the parameters σ_{ij}^{max}, σ_{ij}^{min} defined below). We assume here the terminology of Section 4A.

Fix some time $t \geq 0$. Let Γ_t be the global system history up to t, derived from oracle $\mathcal{O}$ and luck "up to time t" as defined in Section 2C.

Note that $(\mathcal{O}, \Gamma_t)$ essentially specifies everything of the system's immediate future except the pollers' "luck" $L_{t'}$ for times $t' > t$. For all $i,j \in \Pi$ let $\sigma_{ij}(\mathcal{O}, \Gamma_t)$ 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.

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

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

In Appendix II we show

THEOREM II.A: For Algorithm 1, $\sigma_{ij}^{min} = \Omega(1/v)$, $\sigma_{ij}^{max} = O(1)$ satisfying (*).

THEOREM II.B: For Algorithm 2, $\sigma_{ij}^{min} = \Omega(1)$, $\sigma_{ij}^{max} = O(1)$, satisfying (*).

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

LEMMA 5.1:

$$\frac{1}{2}\sigma_{ij}^{min}\left(1 - \frac{1}{2}\sigma_{ij}^{max}\right)^{k-1} \leq P_{ij}(k/(\mathcal{A},\Gamma_t)) \leq \frac{1}{2}\sigma_{ij}^{max}\left(1 - \frac{1}{2}\sigma_{ij}^{min}\right)^{k-1} .$$

Proof. It suffices to observe that the process of i be answered by j is a geometric process, with success probability bounded by $[1/2\ \sigma_{ij}^{min},\ 1/2\ \sigma_{ij}^{max}]$. QED

Since by Theorems II.A and II.B, the σ_{ij}^{min} and σ_{ij}^{max} of interest here are independent of i,j, in the following we drop subscript i,j.

By using Lemma 5.1 and calculating moments, we get

LEMMA 5.2:

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

and

$$\sigma^{min}\ \frac{(4-\sigma^{max})^2}{(\sigma^{max})^3} \leq \text{mean}(k^2) \leq \sigma^{max}\ \frac{(4-\sigma^{min})^2}{(\sigma^{min})^3} .$$

By Lemma 5.2 and expressions for the tail of the geometric

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

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

Let vc be the number of steps required for each phase (of the poller algorithm considered), as given in Lemma 4.2. Then the maximum time required for each phase is $\leq v c r_{max}$.

Lemma 5.1 and 5.2 imply

THEOREM 5.1: If τ is the response of the system, $mean(\tau) \leq v c r_{max} \cdot 2 \sigma^{max}/(\sigma^{min})^2$ and if $T(\varepsilon)$ is the ε-error response, $T(\varepsilon) \leq v c r_{max} \cdot k_{max}(\varepsilon)$.

Finally, this theorem and Theorems II.A, and II.B imply the corollaries claimed in Section 2E.

Acknowledgments. The authors wish to thank Ed Clarke, who introduced us to the synchronization problems considered in this paper, and to Michael Rabin, whose previous work in probabilistic synchronization inspired this work.

References

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

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.

Dijkstra, E.W., "Hierarchical Ordering of Sequential Processes," *Acta Informatica*, Vol. 1, 1971, pp. 115-138.

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, Aug. 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*, Jan. 1981.

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

Lynch, N.A., "Fast Allocation of Nearby Resources in a Distributed System," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, 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, Oct. 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.

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 I

An Application: A Real Time Resource Granting System

To demonstrate the *robustness* of a DCS, with real time implementation, we show that we can use it to solve an interesting class of resource synchronization problems in real time. These are similar to the resource synchronization problems of [Lynch, 1979] and Dijkstra's "dining philosophers" problem) except that the resource synchronization problems we consider have the property that processes *are granting resources only for bounded intervals of time.*

The *resource granting system* (RGS) defined below will be assumed to be embedded within a distributed communication system (DCS) as defined in Section 2. (We will also assume the DCS has an implementation with real time measure, as provided by Sections 3 and 4.)

The (possibly infinite) set of asynchronous processes Π is assumed to be partitioned into a set Π^r of *requesting processes* and a set Π^g of *granting processes.* (Note: It is easy to superimpose each granting process into a requesting process so the total number of processes is $|\Pi^r|$, if we wish.)

We assume a (possibly infinite) set of resources ρ. Let each granting process j in Π^g have a distinct fixed *resource* $\rho(j) \in \rho$ which it controls. Also, each requesting process $i \in \Pi^r$ has a set resources$(i) \subseteq \rho$, fixed for all times $t \geq 0$.

We assume assumptions A1-A3 of Section 2 and also:

A1' We assume a fixed integer constant $v \geq 0$ such that

$$\forall j \text{ in } \Pi^g, \ |\text{resource}^{-1}(j)| \leq v .$$

As in DCS, each process $i \in \Pi$ consists of two synchronized subprocesses

(1) the *director* of i

(2) the *poller* of i.

If $i \in \Pi^r$, the director of i is a *requesting director* and if $i \in \Pi^g$ the director of i is a *granting director.*

We assume the programs of the pollers of Π are given by the implementation of the DCS. The programs of the *granting directors* are given by an implementation of the RGS (which we describe below). At each time $t \geq 0$ *actions of the requesting directors* are specified by an oracle $\mathcal{A}$, our "worst enemy." $\mathcal{A}$ also gives at time $t = 0$ the schedule of the speeds of the processes of Π for all times.

Intuitively, the directors of Π^r request, at various time intervals, resources of ρ from the appropriate granting directors of Π^g. We require that at no time the director of any $i \in \Pi^g$ grants the resource $\rho(i)$ to more than one requesting director. All

communication will be by the DCS system (see Definition in Section 2) as specified below.

We define the *connection graph* of the DCS to be a (possibly infinite) undirected bipartite graph H as follows:

$$\forall i \in \Pi^r,\ j \in \Pi^g$$

$$i \leftrightarrow j \quad \text{iff} \quad \rho(j) \in \text{resource}(i) \ .$$

Note that by A1', the vertices of Π^g have valence $\leq v$ in H; but we do not assume that the valence of vertices of Π^r be bounded by a fixed constant.

For each time $t \geq 0$, the *willingness* (to communicate) digraph G_t is defined:

$$\forall i \in \Pi^r,\ j \in \Pi^g$$

(1) let $i \xrightarrow[t]{} j$ only if $i \leftrightarrow j$ (i.e., the requesting director of i has resource(i) containing $\rho(j)$) and the director of i requests (or has been granted) resource $\rho(j)$ at time t.

(2) Let $j \xrightarrow[t]{} i$ be specified by the programs of the granting director of j (this is provided by an implementation of the RGS).

To satisfy assumption A1 of Section 2 for a DCS, we must assume $\forall t \geq 0$, $\forall i \in \Pi^r$, the director of i simultaneously requests $\leq v$ resources at time t. (Nevertheless, these requested resources may vary with time.)

In addition, we may assume by Section 3 we have an implementation of the above DCS system (satisfying restrictions R1-R4) with real time response T for establishment of communication between directors.

An *implementation of an RGS* specifies the programs of the granting directors of Π^g. These programs may be probabilistic as in [Rabin, 1980].

We assume, in addition to R1-R4 that:

R1' In communication between the granting director of any $i \in \Pi^g$ and the requesting director of any $j \in \Pi^r$ that once communication between i,j is established the granting director of i ignores the particular message values transmitted by j and the granting director of i either transmits "yes" (to indicate the resource $\rho(i)$ *is granted to* j) or "no" (to indicate the resource $\rho(i)$ *is not granted to* j, or $\rho(i)$ has been revoked from j).

R2' $\forall t \geq 0$, the granting director of any $i \in \Pi^g$ cannot grant resource $\rho(i)$ simultaneously to more than one process $j \in \Pi^r$, and furthermore we must have $i \leftrightarrow j$.

Fix a RGS implementation (which may be probabilistic). For each k, $0 \le k \le v$, and oracle $\mathcal{A}$ let the k-*grant response* be the random variable giving the length of the time interval Δ required for any director of $i \in \Pi^r$ to have k resource requests simultaneously granted, given the director of i requested these resources during the entire interval Δ.

Let the *mean* k-*grant response* be

$$\bar{\gamma}_k = \max\{\text{mean}[\gamma_{k,\mathcal{A}}]/\text{all oracles } \mathcal{A}\}$$

For each ε in $(0,1]$ let the ε-*error* k-*grant response* be the minimum function $\gamma_k(\varepsilon)$ such that $\forall$ oracle $\mathcal{A}$

$$\text{Prob}\{\gamma_{k,\mathcal{A}} < \gamma_k(\varepsilon)\} > 1 - \varepsilon \quad .$$

The RGS implementation is *real time* if for all $k \in \{1,\ldots,v\}$ and all $\varepsilon \in (0,1]$, $\gamma_k(\varepsilon) > 0$ and independent of any parameters of the connection graph H (except v, which is assumed constant by A1). Note that if the RGS implementation is real time, the $\bar{\gamma}_k$ is a constant, independent of H.

THEOREM I: There is an RGS implementation with real time k-grant response for any $k \le v$.

It has mean k-grant response

$$\bar{\gamma}_k = O(v^{k+2})$$

and ε-error k-grant response

$$\gamma_k(\varepsilon) = O\left(v^{2k+2} \log\left(\frac{1}{\varepsilon}\right)\right) \quad .$$

Proof. We only sketch the RGS implementation. We assume a DCS implementation with real time response Γ as in Section 3.

We assign each granting director of each $i \in \Pi^g$ to be "willing" at all time to connect to all $j \in \text{resource}^{-1}[i]$. (By A1', i is then "willing" to communicate with no more than v processes.) The grant director of i will repeat (forever) a *grant phase*.

Each grant phase will be of length precisely $2\bar{\Gamma}$ (as defined in Section 2C). There is a variable g such that at the start of each phase the grant director of i will have either given resource $\rho(i)$ to process $g \in \Pi$ or to no process (in which case $g = 0$). During the grant phase j the grant director will, with high likelihood, communicate at least once with all processes in $\text{resource}^{-1}[i]$. Suppose i communicates with $j_1,\ldots,j_\ell$ in this order, during the grant phase.

Thus, for each $s = 1,\ldots,\ell$:

(1) If $g = 0$ then the grant director of i sets g to j_s and says "yes" to j_s (to indicate j_s has been granted resource $\rho(i)$).

(2) Else if $g \neq 0$ then the grant director of i says "no" to j_s (to indicate the grant is denied).

In the case where $g = j_s$ the grant director also sets $g \leftarrow -1$. At the end of the granting phase, the grant director of i sets $g \leftarrow 0$.

A probabilistic analysis (deleted here) of this implementation shows the k-grant response as above. QED

For example, we consider an interesting RGS system, which we call *"hasty dining philosophers."*

Let the *requesting processes* Π^r be distinct distinguished integers $p_0,\ldots,p_n$ (these are the Gödel numbers of the distinguished philosophers) and the *granting processes* Π^g be $0,\ldots,n$ so that Π^g, Π^r are disjoint.

The *resources* ρ are *forks* $[\rho(0),\ldots,\rho(n)]$. Let $p_{n+1} = p_0$ and $\rho(n+1) = \rho(0)$.

Each *philosopher* $p_j \in \Pi^r$ has resources(p_j) consisting of the forks $\{\rho(p_j), \rho(p_{(j+1) \bmod n})\}$. Thus the graph H is a cycle of length $2n$.

COROLLARY I. The "hasty dining philosophers" (as described above) have a real time RGS implementation with mean

$$\text{2-grant response } \bar{\gamma}_2 = O(1)$$

and ε-error 2-grant response

$$, \gamma_2(\varepsilon) = O\left(\log\left(\frac{1}{\varepsilon}\right)\right) \quad .$$

Intuitively, the RGS implementation requires each philosopher p_j to be at any time granted both forks of resource(p_j) in expected constant time, but p_j must be "hasty" and relinquish these resources within constant time interval.

Note that for each $i \in \{0,\ldots,n\}$ the granting process i can be placed within process p_i, thus resulting in essentially only $n+1$ processes.

APPENDIX II

IIA. Probabilistic Analysis of the Noninsisting Algorithm 1

Algorithm 1 is *noninsisting*: 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. Also, if both (directors of) i,j are willing to communicate, both of the pollers of i,j will attempt to establish communication.

Because of this "symmetry" in the way pollers ask, we can show that the worst case $\sigma_{ij}(\mathcal{A},\Gamma_t)$ is when the oracle $\mathcal{A}$ sets the rates of i,j equal (but not necessarily constant) (recall that by our model of Section 2C, $\mathcal{A}$ determines the rates $\{R_t \mid t \geq 0\}$ at time 0) and A cannot influence the probabilistic choice of pollers).

Let cv be the fixed number of steps between phases, as given in Lemma 3E. Let x be the number of steps by which i executes each phase before j, where $0 \leq x \leq cv$.

Let $S(S')$ be the event: j answers i given that j is in its answering mode and i is in its asking mode after (before) j (and, of course, we assume i,j both willing to communicate).

Then we can show $\text{Prob}(S) \cong f(x)$ as $v >> 0$ where

$$f(x) = 1 - \left(1 - \frac{1}{v}\right)^{x/c} .$$

Then

$$\begin{aligned}
\sigma_{ij}(\mathcal{A},\Gamma_t) &= \frac{1}{v}\left(\frac{1}{4}\,\text{Prob}(S) + \left(1 - \frac{1}{4}\,\text{Prob}(S)\right)\cdot\frac{1}{4}\,\text{Prob}(S')\right) \\
&\cong \underset{0\leq x\leq cv}{\text{MIN}} \left\{\frac{1}{4v}\left(f(x) + \left(1 - \frac{1}{4}\,f(x)\right) f(v-x)\right)\right\} \\
&= \frac{1}{2v}\left(f\left(\frac{v}{2}\right) - \frac{1}{8}\,f^2\left(\frac{v}{2}\right)\right) \\
&= O\left(\frac{1}{v}\right) \qquad \text{for } v >> 0
\end{aligned}$$

since

$$f\left(\frac{v}{2}\right) \simeq 1 - e^{1/2c} \qquad \text{for } v >> 0 .$$

Thus we have $\sigma_{ij}^{\min} = O(1/v)$, proving Theorem II.A stated in Section 5.

IIB. Probabilistic Analysis of the "Insisting" Algorithm 2

Algorithm 2 is *insisting*: poller 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 speeds of the process of asking poller to be $1/r_{max}$ and the speed of the answering poller to be $1/r_{min}$.

Let $f(x)$ be as defined in IIA.

We can show:

$$\left(\frac{r_{max}}{r_{min}}\right)\sigma_{ij}^{min} \geq \frac{1}{v}\sum_{x=0}^{v} f(x)$$

$$\geq 1 - \frac{v}{v+1} f(v+1)$$

$$\geq 1 - (1-e^{-c}) = e^{-c} .$$

Thus $\sigma_{ij}^{min} = O(1)$.

Thus Theorem II.B of Section 5 follows.


DA
FILM