Distributed Infimum Approximation

G. Tel

RUU-CS-86-12
August 1986
Revised August 1987

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

Distributed Infimum Approximation

Gerard Tel

Department of Computer Science, University of Utrecht,
P.O. Box 80.012, 3508 TA Utrecht, The Netherlands.

Abstract: A "distributed infimum" is the infimum of a set of values that are distributed over the sites and channels in a distributed system. The values in the set are assumed to change over time only according to certain rules. Distributed Infimum Approximation (DIA) is defined as an abstraction of several interesting problems in the field of distributed computing. Global Virtual Time approximation as well as Distributed Termination Detection are special cases of DIA, as will be argued. Solutions to the DIA problem are given for different communication models (synchronous, bidirectional, FIFO, bounded delay) that can be implemented elegantly on various subtopologies (ring, tree, star).

Keywords and phrases: Distributed algorithms, wave algorithms, Distributed Termination, Global Virtual Time.

1 Introduction

A property P of the states of a distributed system is called *stable* if $P(S_1) \Rightarrow P(S_2)$ for all system states S_1, S_2 with $S_1 \rightarrow^* S_2$. The detection problem for P asks for a distributed algorithm to determine whether P holds in the "current state" of the system [CM85]. Examples of stable properties are termination of the system [Fr80], deadlock [BT84], loss of tokens in a token ring. A stable property can be characterized as a *monotonic boolean function* of the system state. The function is boolean because the property either holds or does not hold, and monotonic because once it holds, it holds forever. Attention has concentrated on properties that are *locally indicative*, which means that the (global) property is the conjunction of local properties x_p for each process p. (Strictly spoken this is not the case for deadlock.) A great number of algorithms for termination detection is known, see [BMR85]. The problem of termination detection is "generic" for detection of locally indicative stable properties.

This work was supported by the Foundation for Computer Science (SION) of the Netherlands Organization for Pure Research (ZWO). Author's UUCP address: mcvax!ruuinfvax!gerard.

In this paper we will generalize the notion of a stable property to more general *monotonic functions* of the system state. The value of a monotonic function is an element of a partially ordered set X. A function F is *monotonic* if for any system state S_2 reachable from S_1 we have $F(S_2) \geq F(S_1)$. The notion of being locally indicative is generalized to the assumption that F is defined as the infimum of values, local to processes. The values over which the infimum is taken may change in time. However, such a change must always result in increasing the value, except after communication with another process. A more precise formulation can be found in the next section. This limitation of the possible changes results in F being monotone, as will also be argued in the next section. The approximation problem asks for an algorithm to compute the value of F of the "current state" of the system. The DIA algorithms can be applied to several, seemingly unrelated problems that are well known from the literature.

(1) **Termination Detection:** When one takes $X = \{active, passive\}$, ordered such that $active \leq passive$, then the infimum F of all local values equals *passive* iff all local values are *passive*. Hence distributed termination detection according to [Fr80] is equivalent to computing the infimum of current X-values in the system. The reader may verify that the behavior of processes, as sketched in [Fr80] or [TL86] is in essence the same as described in this paper.

(2) **Global Virtual Time determination:** Jefferson [Je85] describes a method to simulate synchronous systems (i.e., systems in which the processes can be assumed to have access to a global clock) on asynchronous systems. Each process maintains a local clock, and "global time" is defined as the infimum of the local clock values and "time stamps" of messages in transit. This problem is known also as the **Global Cutoff** problem [SL87].

(3) **Garbage Collection:** Hughes [Hu85] presents a garbage collecting system in which DIA plays an essential role. Hughes gives a DIA algorithm based on the termination detection algorithm by Rana [Ra83].

The rest of this paper is organized as follows. In section 2 we will recall some theory about partially ordered sets and give a formal description of the behavior of the distributed systems we consider. Further we introduce the DIA problem formally. In section 3 we introduce an important "building block" for our DIA algorithms, namely wave algorithms. In the subsequent sections (4 to 8) we point out how wave algorithms can be used to build DIA algorithms under several assumptions about the basic interprocess communication. In section 4 we do this for synchronous communication, in section 5 we assume that channels are bidirectional and use acknowledgements, and in section 6 we assume that channels work in a FIFO fashion. In section 7 we assume that message delay is bounded by a constant, and finally in section 8 we will consider the case in which no assumption is made about the communication channels at all (except the usual one that the network is strongly connected). The algorithm in section 8 will be the most straight-forward application of the definition of a distributed infimum, but also will

be the least elegant to implement. Section 9 contains some final remarks and conclusions.

2 Preliminaries

First we recall some elementary notions from lattice theory. We say that a structure $(X, \leq)$ is a *poset* (partially ordered set) if it satisfies the following axioms:

A1: $x \leq y$ **and** $y \leq z \Rightarrow x \leq z$ (transitivity)

A2: $x \leq y$ **and** $y \leq x \Rightarrow x = y$ (antisymmetry)

A3: $x \leq x$ (reflexivity)

We will assume that X contains a greatest (*top*, $\top$) and a least (*bottom*, $\bot$) element:

A4: $x \leq \top,\ x \geq \bot$

The assumption imposes no real restrictions, as any poset not containing a greatest and a least element, can be extended with two new elements $\top$ and $\bot$ which satisfy A4 by definition. The only reason to introduce $\top$ and $\bot$ is to simplify the algorithms. Next we postulate the existence of a binary operator *infimum* (denoted $\wedge$) on X, satisfying

A5: $x \wedge y \leq x,\ x \wedge y \leq y$

A6: $\forall z\ (z \leq x$ **and** $z \leq y) \Rightarrow z \leq x \wedge y$

It is easy to verify the following facts:

A7: $x \wedge y = y \wedge x$ (commutativity)

A8: $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ (associativity)

A9: $x \wedge x = x$ (idempotency)

A10: $x \wedge \bot = \bot,\ x \wedge \top = x$ (absorption)

The infimum operator is extended to finite subsets of X as follows: for $A = \{a_1, a_2, \ldots, a_k\} \subseteq X$, let $\inf(A) = a_1 \wedge a_2 \wedge \ldots \wedge a_k$. The operator satisfies the following properties:

A11: $\forall a \in A\ \ \inf(A) \leq a$

A12: $(\forall a \in A\ z \leq a) \Rightarrow z \leq \inf(A)$

A13: $A_1 \subseteq A_2 \Rightarrow \inf(A_2) \leq \inf(A_1)$

A14: $\inf(A_1 \cup A_2) = \inf(A_1) \wedge \inf(A_2)$

We will assume that the nodes of a distributed system all maintain some value from a poset $(X, \leq)$ according to certain rules that we will specify.

A distributed system consists of a set of *processes* and a set of directed communication *channels*. Each channel is directed from one process to another. Processes send messages over outgoing channels and receive messages over incoming channels. By an *event* we denote the *sending* of a message, the *receipt* of a message or an *internal* computation in a process. The event of sending a message will eventually be followed by the receipt of that message. The following assumptions on communication in a distributed system can be made:

(1) We say communication is *synchronous* if the send and the corresponding receive events always happen at the same time. In this case the two events are regarded as one "communication" event.

(2) We say the communication is *FIFO* (*First-In-First-Out*) if, whenever a process sends two messages over the same channel, the messages will be received (and processed) in the same order at the other end.

(3) We say the communication is *bidirectional* (in contrast to *unidirectional*) if, for every channel from process p to process q, there is also a channel from q to p.

(4) We say that the communication has Δ–*bounded delay* if whenever a message is sent, the corresponding receive event will take place within time Δ after the send event, measured on some global clock.

At any time each process is in one of a set of *process states* and each channel is in some *channel state*; the state of a non-FIFO channel is the set of messages in transit over this channel, the state of a FIFO channel is the sequence of messages in transit. In the *initial* state of the system all channels are empty. Let S_1, S_2 be system states. We say S_2 is *reachable* from state S_1 (notation $S_1 \rightarrow^* S_2$) if there is a sequence of legal state transitions (events) leading from S_1 to S_2.

We will now define a function F on system states. Suppose each process p owns a register x_p of type X. That is, the value of x_p is a member of the poset $(X, \leq)$. Further every message that is exchanged also carries a value from this poset, its X–*stamp*. As a result of send, receive or internal events the value of x_p-registers and X-stamps can be set as follows:

send: p sends a message $\langle\!\langle M, x_p \rangle\!\rangle$ to q. (No change of x_p.)

receive: p receives $\langle\!\langle M, x \rangle\!\rangle$. p sets x_p to a value x', where $x' \geq x_p \wedge x$.

internal: p sets x_p to a value $x' \geq x_p$.

For each system state S, consider the multiset of x-values that are in the system in state S (by *expression*$^{(S)}$ we mean the value of *expression* in system state S):

Definition 2.1:

$\mathrm{Val}(S) := \{x_p^{(S)} : p \text{ is a process}\} \cup \{x : \langle\!\langle M, x \rangle\!\rangle \text{ is in transit in } S\}$.

$F(S) := \inf(\mathrm{Val}(S))$.

We now have the following theorem:

Theorem 2.1: (Monotonicity) For system states S_1, S_2 we have $S_1 \rightarrow^* S_2 \Rightarrow F(S_1) \leq F(S_2)$.

Proof: By induction on the sequence of steps leading from S_1 to S_2. Verify, that F can only grow as a result of any event. □

The problem of distributed infimum approximation can now be defined as follows. Given a distributed system and a monotonic function F as defined, suppose the system goes through a sequence of system states S_0, S_1, S_2, ... We call this sequence the *basic computation*, and

the messages belonging to this basic computation will be referred to as the *basic messages*. We wish to superimpose an algorithm on the basic computation to update in each process p a local approximation f_p of F. The approximation f_p must satisfy the following two conditions:

(1) *Safety*: At any time $f_p \leq F(S)$ for all p (i.e., the approximation must be a lower bound for F).

(2) *Progress*: If at some time $F(S) \geq k$ (for some $k \in X$), then within finite time $f_p \geq k$ for all p.

Remark: Whenever we use words as "time", "later" etc, we refer to the actual sequence of system states. A "point in time" is just a system state, by "later" we mean a later state, etc. For states t_1 and t_2, by $t_1 \leq t_2$ we mean that t_2 is a later state then t_1. Note this implies $t_1 \rightarrow^* t_2$ and hence $F(t_1) \leq F(t_2)$.

3 Wave algorithms

An important paradigm for our DIA algorithms will be the concept of a *wave*. Suppose all processes have some local *constant* value r, and we want to compute the infimum of these values. It is then necessary to "collect" all values r. A *wave* algorithm is a distributed algorithm that collects the local values and ensures that all local values (*reports*) are concentrated in one process. This process knows that it has received all reports. Next it computes the infimum and broadcasts the result to all other processes. The receipt of this broadcast may trigger the start of a next wave, when repeated evaluation of an infimum is wanted. Note, that this mechanism ensures, that waves are *time disjoint*: a new wave can start only after the previous one has completed. When p reports its r_p to the wave, we say that p *passes the wave* or the wave *visits* p.

From now on we will use wave algorithms for the following three purposes:

(1) *Knowledge acquisition*: the reported values are concentrated in one process.

(2) *Broadcasting*: a result from the previous wave can be made public.

(3) "*Synchronization*": a process may take some actions the moment it is visited by the wave.

We give a simple wave algorithm, suitable for networks that contain a Hamiltonian circuit (ring), and a more complex one that is suitable for networks that contain a spanning tree of bidirectional channels.

First, assume that the network contains a Hamiltonian circuit as a subtopology. That is, each process p knows of a process $Suc(p)$ (the *successor* of p), such that a path, starting at a process p and stepping from successor to successor, passes through every process exactly once

and returns to p. We add a new process *leader* to the ring, which does not take part in the basic computation. Its only task is to coordinate the wave algorithm. For this system a wave algorithm can be given as follows:

leader:
send «REPORT, ∅» to *Suc* (*leader*);
wait until «REPORT, A» arrives;
result := inf (A) ;
send «BROADCAST, *result*» to *Suc* (*leader*);
wait until «BROADCAST, *result*» arrives.

$p \neq$ *leader:*
wait until «REPORT, A» arrives ;
send «REPORT, $A \cup \{r_p\}$» to *Suc* (p);
wait until «BROADCAST, *result*» arrives ;
res_p := *result* ;
send «BROADCAST, *result*» to *Suc* (p).

Let N be the number of basic processes. Obviously a wave takes $2N+2$ messages and $O(N)$ time. When it is necessary to repeat the wave algorithm many times it is elegant and efficient to combine the broadcast phase of one wave with the report phase of the next wave. Only $N+1$ messages are used per wave.

leader:
send «TOKEN, ∅, *nil*» to *Suc* (*leader*);
repeat
wait until «TOKEN, A, x» arrives ;
send «TOKEN, ∅, inf *(A)*» to *Suc* (*leader*)
until ... (* some termination condition *).

$p \neq$ *leader:*
repeat
wait until «TOKEN, A, x» arrives ;
$res_p := x$;
send «TOKEN, $A \cup \{r_p\}$, x» to *Suc* (p)
until ... (* some termination condition *).

Of course it is also possible to select one of the basic processes and make this process the leader. This process then runs a program that is a combination of the two programs above.

The message complexity is then $2N$ (or N) because no extra process is added.

Next assume the network contains a spanning tree T of bidirectional communication channels. Each process p knows which of its channels are T-edges. The algorithm goes as follows: suppose p has degree d in T, i.e., d of p's channels are T-edges. When p has received reports «REPORT, A_1», «REPORT, A_2», ... «REPORT, A_{d-1}» over $d-1$ of its T-edges, it sends «REPORT, $A_1 \cup ... \cup A_{d-1} \cup \{r_p\}$» over the d^{th} edge. Note that if p is a leaf, it sends «REPORT, $\{r_p\}$» immediately. It is now easy to see that a report sent from p to its T-neighbor q contains the values of all processes that belong to the subtree of T under p (as seen from q). All processes will eventually be in the situation of having received $d-1$ reports. Two cases may arise:

Case 1 (Single leader case): One process (say p) has received d reports before it has sent a report itself. Now p computes $\inf(A_1 \cup ... \cup A_d \cup \{r_p\})$ and broadcasts this result.

Case 2 (Double leader case): On one edge (say pq) two reports cross each other. This implies that these two reports together contain all reported values. p and q detect this situation (they receive a report over an edge, over which they just sent a report) and compute the infimum from the two reports. They broadcast the result over the T-edges except pq.

Which of the two cases can arise depends on properties of communication and on implementation details. Note, that the single leader case can always be avoided: a process that has received d reports before it sent a report itself can ignore the last report for a while, send a report and finally treat the last report as mentioned under case 2.

For each wave two messages go over each T-edge. So, the number of messages for one complete wave is $2(N-1)$. The time to complete is $O(D)$, where D is the diameter of the spanning tree.

It is possible for the participating processes to "precompute" the reports: in stead of sending A, they can just send $\inf(A)$. Only two wave algorithms are given here, in fact any "total algorithm" [TT87] can be used.

4 Distributed Infimum Approximation in systems with synchronous message communication

In this section we will consider distributed systems in which message communication is synchronous. This implies that in no system state are there messages in transit and thus $\text{Val}(S) = \{x_p^{(S)} : p \text{ is a process}\}$. We use time diagrams, cf. figure 1, to depict the behavior of the basic computation and our algorithms. Each horizontal line represents one process. Gray arrows represent (basic) messages. Visits of a wave will be denoted by circles, and visits belonging to the same wave will be connected by fat black lines. Where necessary numbers in

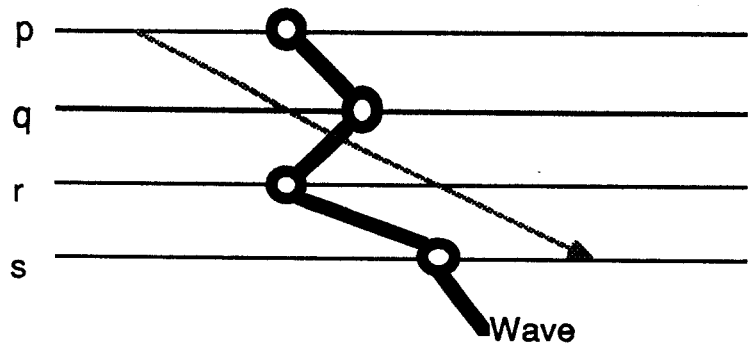

Figure 1

the pictures indicate x values.

Suppose the wave algorithm was executed "in a flash", i.e., all processes were visited in the same system state. Then we could use the following algorithm:

Algorithm A: Send a wave through the system in system state (say) t_1. Let process p report $x_p^{(t_1)}$ to this wave. The infimum of the reported values is broadcast, and this broadcast triggers the next wave, which takes place in system state t_2, etc. This is repeated as long as the basic computation goes on.

Theorem 4.1: Algorithm A satisfies safety and progress.

Proof: Let f_i be the value, computed as the result of the i^{th} wave. Now $f_i = \inf(\{x_p^{(t_i)} : p \text{ is a process}\}) = \inf(\text{Val}(t_i)) = F(t_i)$. Safety follows from the fact that, by the time t at which f_i is broadcast and a process p sets f_p to f_i, we have $t \geq t_i$ and hence $F(t) \geq f_i$.

Suppose $F(t) \geq k$ $(k \in X)$ at some time t. Let $t_i \geq t$. Then for all p $x_p^{(t_i)} \geq k$. Hence $f_i \geq k$ and $f_p \geq k$ at all times $t \geq t_{i+1}$ because by time t_{i+1} the broadcast of f_i is finished. So, within two waves, $f_p \geq k$. Progress follows. □

Unfortunately, simultaneity cannot be achieved in distributed systems (unless a global clock is available). In general it can not be avoided that the wave visits different processes during different system states. This makes algorithm A unsafe, as can be shown by the following scenario with two processes, see figure 2. Initially $x_p = 0$, $x_q = 5$. The wave visits q and q reports 5. Then p sends a message to q, which causes q to set x_q to 0. Next, p increases x_p

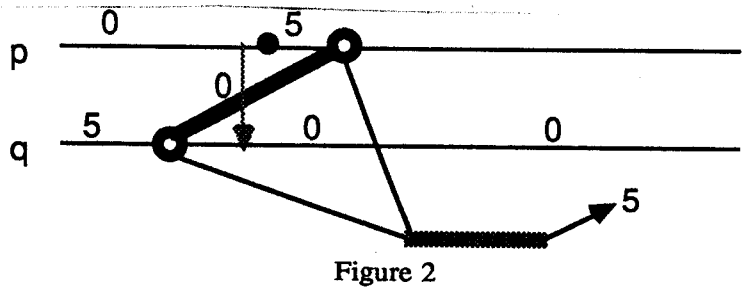

Figure 2

to 5 and is visited by the wave. So, p also reports 5 and the result of the wave is 5, which is bigger than the current value of x_q (0). The problem can be solved (as in [CM85]) by observing each process during a certain *period* rather than just at a point in time, in such a way that the periods have a nonempty intersection.

Definition 4.1: For each process,the i^{th} *observation period* is the time between the visits of the $(i-1)^{th}$ and the i^{th} wave to this process.

In the next and the following theorems we will assume that the waves are time disjoint, so for every i the i^{th} observation periods of all processes will indeed have a nonempty intersection. Let r_p be the infimum of all the values x_p had during p's i^{th} observation period.

Theorem 4.2: $\inf(\{r_p : p \text{ is a process}\})$ is a safe lower bound for $F(e)$, where e is the time the i^{th} wave completes (see figure 3).

Proof: Denote by f_i the infimum of the r_p, and by $start_p$ and end_p the time the $(i-1)^{th}$ and i^{th} wave passes p, respectively. The waves are time disjoint and hence there is a time t such that $start_p \le t \le end_p$ for all p. So $r_p \le x_p^{(t)}$ and hence $f_i \le \inf(\{x_p^{(t)} : p \text{ is a process}\}) = F(t) \le F(e)$. □

Theorem 4.2 suggests the following algorithm:

> **Algorithm B:** Repeatedly send time disjoint waves through the network. A process p reports to the i^{th} wave the infimum of all values x_p had during the i^{th} observation period. The infimum of these values is broadcast during the $i+1^{th}$ wave.

Theorem 4.3: Algorithm B satisfies safety and progress.

Proof: The safety follows by theorem 4.2. Again let f_i be the infimum, computed after the i^{th} wave. To prove progress, assume that $F(t) \ge k$ for some time t, some $k \in X$. So for all p and $t' \ge t$ we have $x_p^{(t')} \ge k$. Let i be the number of the first wave that starts after t, see

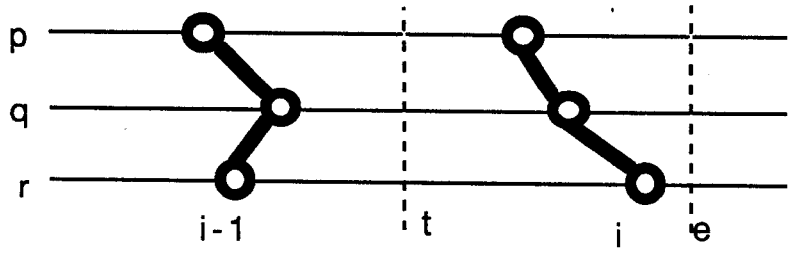

Figure 3

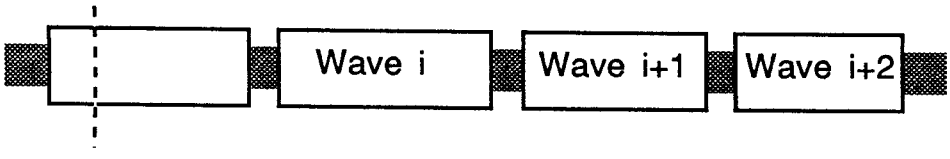

Figure 4

figure 4. For all processes p, $x_p \geq k$ at any time after the visit of the i^{th} wave. So p's report r_p to the $(i+1)^{th}$ wave satisfies $r_p \geq k$. All reports do so, and it follows that $f_{i+1} \geq k$. This result is broadcast during the $(i+2)^{th}$ wave. It follows that within four waves after t all p have $f_p \geq k$. □

The wave algorithms of section 3 can be used to implement this algorithm. For example, on a ring the program would look like this:

leader:

send «TOKEN, ⊤, ⊥» to *Suc*(*leader*);
repeat
    wait until «TOKEN, a, f» arrives ;
    send «TOKEN, ⊤, a» to *Suc*(*leader*)
until the basic computation terminates.

$p \neq$ *leader:*

$r_p := x_p$; $f_p := \bot$;
repeat
    wait until «TOKEN, a, f» arrives ;
    $f_p := f$;
    send «TOKEN, $a \wedge r_p, f$» to *Suc*(p);
    $r_p := x_p$
until the basic computation terminates.

A process $p \neq$ *leader* can receive a basic message during its waits. Such a message is handled according to the basic program, immediately after which $r_p := r_p \wedge x_p$ is executed:

upon receipt of «M, x» **do**
    handle message according to basic program ;
    $r_p := r_p \wedge x_p$.

5 DIA in distributed systems using communication with acknowledgements

When message communication is not synchronous messages may take some time to reach their destination. The definition of the function F includes the X-stamps of these messages in transit. Any DIA algorithm *ALG* that does not deal with these messages is not correct, as a simple scenario shows (see figure 5). Suppose initially $x_p = x_q = 0$. p sends a message

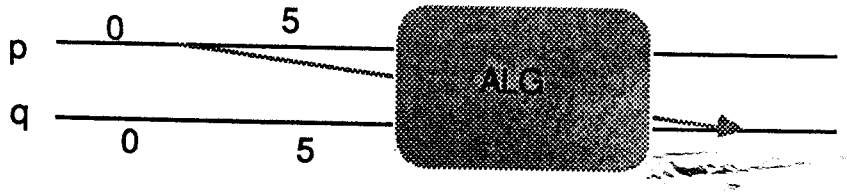

Figure 5

«M,0» to q. Now both processes increase their x to 5. Note that now $\inf(\{x_p : p \text{ is a process}\}) = 5$ and hence, by the progress property, within finite time *ALG* ensures $f_p = f_q = 5$. (Remember *ALG* ignores messages in transit and as far as *ALG* is concerned, the message could as well not have been sent at all.) Next q receives «M,0» and finds out that $f_q \geq F$, which is a conflict.

We will present a solution to the DIA problem in which it is ensured that p's report is a lower bound for messages, sent by p, that are still in transit. Eventually p will learn which of its messages are no longer in transit, namely, when it receives acknowledgements for them. So, p maintains a multiset variable $UNACK_p$, in which the X-stamp of every message that p sends is inserted, and an X-stamp is deleted again when an acknowledgement for the message of this X-stamp is received. The derived algorithm will again use time disjoint waves. Again by the i^{th} observation interval we mean the time between the i^{th} and the $i+1^{\text{th}}$ wave. As an analog to theorem 4.2. we have:

Theorem 5.1: Let r_p be the infimum of all the values which x_p had or $UNACK_p$ contained during p's i^{th} observation period. Then $\inf(\{r_p : p \text{ is a process}\}) \leq F(e)$, where e is the time the i^{th} wave completes.

Proof: Let $f_i = \inf(\{r_p : p \text{ is a process}\})$. Choose t as in theorem 4.2. Let «M,x» be in transit at time t, and p be its sender. Then $x \in UNACK_p^{(t)}$ and hence $r_p \leq x$ and $f_i \leq r_p \leq x$. Furthermore, for all p we have $r_p \leq x_p^{(t)}$ and hence $f_i \leq x_p^{(t)}$. So $f_i \leq v$ for all $v \in \text{Val}(t)$ and hence $f_i \leq F(t) \leq F(e)$. □

We can now formulate the following DIA algorithm:

> **Algorithm C:** During normal execution, processes acknowledge basic messages and maintain a set *UNACK* as described. Time disjoint waves are sent through the network. To each wave a process reports the infimum of the values its x register had and its *UNACK* contained since the previous wave passed. The infimum of the reported values is broadcast as the new approximation of F.

In the following it is assumed that not only basic messages, but also acknowledgements are received in finite time.

Theorem 5.2: Algorithm C satisfies safety and progress.

Proof: Safety follows from theorem 5.1. Assume that $F(t) \geq k$ for some time t, some $k \in X$. So for all $t' \geq t$, all p, $x_p^{(t')} \geq k$ and for all messages «M, x» in transit at t' we have $x \geq k$. Within finite time after t, say at t'', all acknowledgements for messages sent before t will have arrived. So from then on all sets *UNACK* contain only values $\geq k$. Let i be the number of the first wave that starts after t'', For all processes p, $x_p \geq k$ and $UNACK_p$ contains only values $\geq k$ at any time after the visit of the i^{th} wave. So p's report r_p to the $(i+1)^{th}$ wave satisfies $r_p \geq k$. All reports do so, and it follows that $f_{i+1} \geq k$. This result is broadcast during the $(i+2)^{th}$ wave. It follows that within four waves after t'' all p have $f_p \geq k$. □

Many systems, even those where communication channels are unidirectional to the user's point of view, use acknowledgements on a lower level to ensure that messages will eventually arrive. In such systems the acknowledgements needed in this algorithm would not increase the message complexity, as the lower levels in the hierarchy could inform the higher level about the receipt of the acknowledgements.

We sketch a sample implementation on a tree. We use the tree wave algorithm from section 3, and assume that only the double leader case arises (remember that it can always be "simulated"). The basic events are extended with some overhead as follows:

SEND:

p sends «M, x_p» to q ;
insert ($UNACK_p$, x_p).

RECEIVE:

p receives «M, x» from q ;
send «ACK, x» to q ;
handle message according to basic program ;
$r_p := r_p \wedge x_p$.

INTERNAL:

$x_p := x'$; (* $x' \geq x_p$ *).

A new type of message, «ACK, x» is introduced. Its receipt by p triggers:

delete ($UNACK_p$, x).

Concurrently the following code is run (d is the degree of p in T):

$r_p := x_p$; $f_p := \perp$;
repeat
  wait until «REPORT, v_i» has been received over $d-1$ channels ;
  $rep := \inf(\{v_1, .., v_{d-1}, r_p\})$;
  send «REPORT, rep» over the d^{th} channel ;
  $r_p := \inf(UNACK_p) \wedge x_p$;
  wait until a message arrives over this last channel ;
    (* this is either a «REPORT, v» or a «BROADCAST, f» *)
  if it is «REPORT, v» **then** $f := rep \wedge v$;
  $f_p := f$;
  send «BROADCAST, f» over the other $d-1$ channels
until basic computation ready.

The data structure used to represent $UNACK_p$ must support insertions, deletions and inf-evaluations. One can use for example a balanced tree, in which every internal node contains the infimum of all the values in the subtree under that node. It is easy to maintain this information, even in case of rotations, in $O(1)$ time per node. It follows that insertions and deletions cost $O(\log n)$ time, where n is the size of the multiset. Inf-evaluations cost constant time, because the infimum of the set can be found in the root.

6 DIA in distributed systems with communication over FIFO channels

We say an event happens before (after) a wave if it happens before (after) this wave visits the process in which the event takes place. In this section we will prove that algorithm B is safe if one can ensure that any message, sent before wave $i-1$, is received (and handled) before wave i. Or, equivalently, a message sent in observation period $i-1$ is received at the latest during observation period i. Further we will see how we can obtain this property of communication in a FIFO environment. The following is an analogue of theorem 4.2:

Theorem 6.1: Suppose each message sent before wave $i-1$ is received before wave i. Let r_p be the infimum of the values x_p had between the $(i-1)^{\text{th}}$ and the i^{th} wave and the X-stamps of messages p received between the $(i-1)^{\text{th}}$ and the i^{th} wave. Let $f_i := \inf(\{r_p : p \text{ is a process}\})$. Then $f_i \leq F(e)$, where e is the time the i^{th} wave completes.

Proof: Choose t as in the proof of theorem 4.2. Let at time t a message «M, x» be on its way from p to q. If it was sent after wave $i-1$ then $r_p \leq x$ because x_p was equal to x at the time of the sending of the message, and this time is within p's i^{th} observation period. If it was sent before the $(i-1)^{\text{th}}$ wave it will by assumption be received before the i^{th} wave and

hence $r_q \leq x$ by definition of r_q. In both cases we get $f_i \leq x$. Also $x_p^{(t)} \leq r_p \leq f_i$. So it follows $f_i \leq F(t) \leq F(e)$. □

Theorem 6.2: Suppose each message sent before wave $i-1$ is received before wave i. Let r_p be the infimum of the values x_p had between the $(i-1)^{\text{th}}$ and the i^{th} wave. Let $f_i := \inf(\{r_p : p \text{ is a process}\})$. Then $f_i \leq F(e)$, where e is the time the i^{th} wave completes.

Proof: To make the reader realize that this theorem is more tricky than theorem 6.1 we first show that not necessarily $f_i \leq F(t)$, see figure 6. Start with $x_p = x_q = 0$. p sends «M, 0» to q. Both p and q increase x to 1 by an internal event. Wave $i-1$ visits p and q (and yields 0). Now wave i visits p, note that $F = 0$ because the message is still in transit. Next q receives «M, 0» but does not decrease x_q, and passes the wave. Here, $r_p = r_q = 1$, so $f_i = 1$. Yet, F was 0 at the start of the first wave. We continue to prove $F(e) \geq f_i$.
Call a message *bad* if its stamp is not $\geq f_i$, and call a process bad if its x-register is not $\geq f_i$. Note that only bad processes can send bad messages, and only the receipt of a bad message can make a good process bad. Suppose that not $F(e) \geq f_i$. Then there must be a bad message or a bad process at time e. Because no process was bad at the time it was visited by wave i, a bad process at time e implies it received a bad message after it was visited by wave i. Let M be the first bad message, received by a process after wave i visited this process. If M was sent after wave i, its sender was bad after wave i, and hence must have received an earlier bad message after wave i visited it, contradicting the choice of M. Because M was received after wave i it was not sent before wave $i-1$ by assumption. It follows that the bad message was sent between wave $i-1$ and i, and so its sender was bad between these two waves. This contradicts the fact that it reported a value $\geq f_i$ to the i^{th} wave. □

(Even if we take r_p as the infimum of x_p at the time of the visit of the i^{th} wave and the stamps of messages p sent during the i^{th} observation period, the infimum of the r_p would be "safe".)

Of course in general it is not possible to ensure that a message arrives before a predefined point in time. But we will see that when the communication channels are FIFO, it is possible to *postpone the visit of the wave in q* long enough to ensure that the condition of theorem 6.2 is fulfilled. We do this by *flushing* the channels: let each process send a special

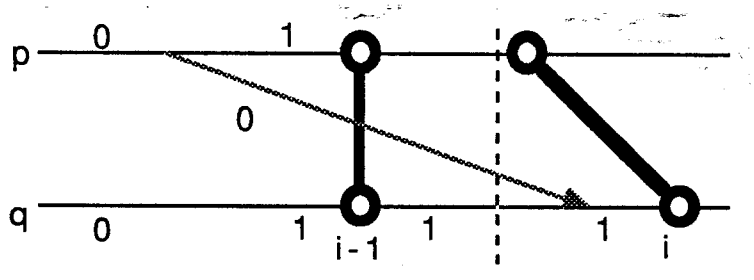

Figure 6

marker message over all of its outgoing communication channels at the beginning of each observation period. A process does not end its observation period (i.e., does not pass the next wave) until it has "consumed" a marker from each incoming channel. (It needs a next marker in each observation period. However, when more then one marker is received within one observation period, the surplus is saved for the next observation period.)

Theorem 6.3: Assume communication channels work in a FIFO fashion. Flushing ensures that messages sent before wave $i-1$ are received before wave i.

Proof: Suppose p sends q a message M before wave $i-1$. At the time this wave passes p, p sends its i^{th} marker to q. So M is sent before the i^{th} marker and, by the FIFO property of the channel, it is also received before this marker. q does not pass the i^{th} wave before it has received this marker, so M is received before the i^{th} wave visits q. □

We can superimpose flushing on algorithm B:

> **Algorithm D:** Directly after initialization, each process sends a marker over all of its outgoing channels. A process does not pass a wave until it has consumed a marker from all of its incoming channels. To each wave a process reports the infimum of the values its x-register had since the previous wave passed. The infimum of the reported values is broadcast as the new approximation of F.

In the following theorem it is assumed that not only basic messages, but also markers are received in finite time:

Theorem 6.4: Algorithm D satisfies safety and progress.

Proof: The safety follows from theorems 6.2 and 6.3. Suppose $F(t) \geq k$ for some time t, some $k \in X$. Because the way the approximations are computed is the same as in algorithm B, it can be proven as in theorem 4.3 that as soon as 3 waves have started and completed after t, all processes p have $f_p \geq k$. It remains to show that no wave is deferred infinitely. When all processes eventually begin their i^{th} observation period, they will all send an i^{th} marker on all of their outgoing channels. Because the markers will all be received in finite time, all processes will eventually be enabled to end their i^{th} observation period, pass the i^{th} wave and send an $(i+1)^{th}$ marker over all outgoing channels. Because after initialization all processes start their first observation period (and send markers for the first time) it follows by induction that all waves will eventually complete. □

A disadvantage of this algorithm is that it uses communication bandwidth of all communication channels, whereas the other algorithms use bandwidth on a fixed subtopology only. As an example we sketch the implementation on a ring with a leader, to show how the visits are deferred. For the leader the code remains the same as in section 4.2:

leader:

send «TOKEN, $\top$, $\bot$» to *Suc*(*leader*);
repeat
    wait until «TOKEN, *a*, *f*» is received ;
    send «TOKEN, $\top$, *a*» to *Suc*(*leader*);
until basic computation terminates.

For the other processes code for handling markers is added:

$p \neq$ *leader:*

$r_p := x_p$; $f_p := \bot$;
send «MARKER» over all outgoing channels ;
repeat
    wait until «TOKEN, *a*, *f*» and for each incoming channel
        a «MARKER» have arrived ;
    (* Now the visit takes place *)
    $f_p := f$;
    send «TOKEN, $a \wedge r_p$, f» to *Suc*(p);
    $r_p := x_p$;
    send «MARKER» over all outgoing channels
until the basic computation terminates.

During waits, basic messages are handled as specified by the basic process, after which $r_p := r_p \wedge x_p$ is executed. It is possible that a process receives two markers over a line before it can pass a wave. In this case only one marker must be "consumed", the other one counts for the next round.

7 DIA in networks using communication with Δ–bounded delay

In this section we assume that message delay is bounded by a constant time Δ. That is, a (basic) message is received within time Δ after it is sent. We will adapt the solutions of the previous sections slightly so they are correct for this communication model. It is necessary that processes have a *local clock*: this means that every process can measure the time, elapsed between two events within this process.

Algorithm C was presented in which a process p included the X-stamp of a message it sent in its report r_p, until it learned (by receiving an acknowledgement) that it was received. Of course, no acknowledgements are needed for this purpose when message delay is bounded a priori: p knows the message is received when a time Δ has elapsed since its sending. So, p

just includes the message in its reports during a fixed time Δ after it sent it. The reader will not be surprised by the following theorem, analogous to 4.2, in which Δ-bounded delay is assumed:

Theorem 7.1: Let r_p be the infimum of all values x_p had during p's i^{th} observation period and the X-stamps of messages p sent later than Δ before the beginning of the i^{th} observation period. Then $f_i := \inf(\{r_p : p \text{ is a process}\}) \leq F(e)$, where e is the time the i^{th} wave completes.

Proof: Choose t as in the proof of theorem 4.2. If «M,x» is in transit from p to q at time t, it was sent later than time Δ before the beginning of p's i^{th} observation period, so $r_p \leq x$ and $f_i \leq x$. Further $r_p \leq x_p^{(t)}$, so $f_i \leq x_p^{(t)}$. It follows $f_i \leq F(t) \leq F(e)$. □

We leave it for the reader to formulate the algorithm more precisely, prove that it has the property of progress and give sample implementations.

Next we present a DIA algorithm that is a modification of Algorithm D. When message delay is bounded by a constant Δ, no message can cross two waves if there is a time delay of at least Δ between two consecutive waves.

Theorem 7.2: When there is a time of at least Δ between the last visit of one wave and the first visit of the next wave, a message sent before wave $i-1$ is received before wave i.

Proof: Obvious. □

This theorem can replace theorem 6.3, and yields a second algorithm for DIA in combination with theorem 6.2. Again its formulation and a proof of its progress are left to the reader. An implementation of this algorithm differs from the programs in section 4 only in that we must ensure, that no process passes wave $i+1$ within time Δ after the last process passed wave i. This is easy: a process that has computed the result of one wave and is about to broadcast it, waits a time Δ before it does so. In the case of the ring wave algorithm, this means that only the leader needs to know Δ and have a clock.

Both solutions given in this section can also be used in case the local clocks have a bounded drift ρ. In that case the processes wait until a time $(1+\rho)\cdot\Delta$ has elapsed on their clocks, instead of just Δ.

8 DIA in arbitrary networks

In the previous four sections rather simple, elegantly implemented DIA algorithms were presented. In all cases the result of a wave could be computed as a function of the values, reported by the processes. The algorithms relied on some special property of the communication system (synchronous, bidirectional, FIFO, and Δ-bounded delay, respectively). In this

section a DIA algorithm is presented that relies on no such properties. The price we have to pay is high: the new approximation of F in this algorithm depends not only on the reported values, but also on information gathered in previous waves. Further, the reports have no longer the format of just an element of X, but they are a message list and a value from X. This makes implementation rather inelegant.

The following theorem is a straightforward application of the definition of F:

Theorem 8.1: Let r_p be the infimum of the values x_p had between waves $i-1$ and i. Let $f_i = \inf(\{r_p : p \text{ is a process}\}) \wedge \inf(\{x : \text{«M, } x\text{» was sent before wave } i, \text{ but not received before wave } i-1\})$. Then $f_i \leq F(e)$ (where e is the time the i^{th} wave completed).

Proof: If «M, x» is in transit at t, then obviously it was sent before wave i, and not received before wave $i-1$ so $f_i \leq x$. Also $f_i \leq r_p \leq x_p^{(t)}$. So $f_i \leq F(t)$. □

Theorem 8.2: Let r_p be x_p at the moment wave i passes p. Let $f_i = \inf(\{r_p : p \text{ is a process}\}) \wedge \inf(\{x : \text{«M, } x\text{» was sent before wave } i, \text{ but not received before wave } i\})$. Then $f_i \leq F(e)$ (where e is the time the i^{th} wave completed).

Proof: Again call a message (process) bad if its X-stamp (x-register) is not $\geq f_i$. As in the proof of theorem 6.2, **not** $F(e) \geq f_i$ implies that a bad message, sent before wave i, is received after wave i. This contradicts with the definition of f_i. □

Theorem 8.2 suggests the following algorithm:

> **Algorithm E:** Start with an empty multiset. To each wave a process reports its x-value and a list of sent and received messages. After each wave the messages are inserted/deleted in the multiset, so that it contains the messages, specified in theorem 8.2. Next the infimum of the values in the multiset and the reported values is computed and broadcasted as a new approximation of F.

Theorem 8.3: Algorithm E satisfies safety and progress.

Proof: The safety follows from theorem 8.2. Suppose $F(t) \geq k$ at some time t, for some $k \in X$. Let i be the first wave starting after t. The x_p-values reported to this wave are all $\geq k$, and all messages crossing wave i have stamps $\geq k$. Hence $f_i \geq k$. □

Implementation on a ring (or other network) with a leader is easy: the leader can maintain the multiset of messages. Implementation using the tree wave algorithm from section 3 seems hard, because the leader(s) is (are) located differently in each round, but it needs the multiset. Broadcasting this multiset together with the result of a wave seems inelegant because of the huge mass of information that can be contained in it. Note that when X is a small finite set (like $\{a,p\}$ in the case of termination detection) the set can be represented compactly by giving for each $k \in X$ the number of times k appears in it. In this case broadcasting may be feasible, see [BMR85].

It is possible that the receipt of a message is reported earlier than its sending. We suggest a structure in which such messages can be inserted "negatively", and that "negative" and "positive" insertions annihilate each other. Such a structure is also used (and described) in [Je85] (be it for a different purpose).

9 Conclusions, discussion

In this report we defined and studied the problem of Distributed Infimum Approximation. We gave several algorithms that solve the problem. Most of them are based on existing algorithms for Termination Detection, in fact a special case of DIA.

Because the basic computation can go on "infinitely", the total number of DIA messages can also be infinite. We can only say something about the number of messages a DIA algorithm exchanges *per update* of the local approximations. Call the number of messages a wave takes W, then all algorithms, except algorithm D, use W messages per update. Algorithm D uses $W+E$ messages (E is the number of communication channels) because besides the wave messages a marker is sent over each communication channel. Algorithm C also doubles the message complexity of the basic communication, because each basic message must be acknowledged. The size of the messages is in all algorithms, except algorithm E, equal to the space needed to represent one (two for the ring wave) element of X. The markers in algorithm D can have constant, small size. The messages in the algorithm E can be much longer, for they contain a message list.

It is possible to control the frequency of the waves as follows: if one wants an update of the local approximations once in time D, give one process (or more processes) a clock and ensure that this process does not pass a wave within time D after it passes a previous wave. Here, by the way, we touch at a less desirable property of our algorithms: they can be slowed down forever by a defective process. This however seems to be inherent to wave algorithms.

We think that the basic ideas in this report are very useful, but the given algorithms and theorems may be subject to improvements.

The function F we approximated in this report was defined as an *infimum* and guaranteed to be *monotone* by the process behavior. It may be interesting to investigate the approximation of *non-monotonic* functions, although it is not so clear what this could mean.

Also it may be interesting to study other operators than the infimum. Remember, the infimum (as a binary operator) satisfies commutativity, associativity and idempotency (A7, A8, A9). Operators satisfying these three properties are considered to be candidates for evaluation with a wave algorithm in [HMR86]. The following theorem says that infimum functions are "generic" for these operators:

Theorem 9.1: Let X be a set and ■ be an operator on X, satisfying commutativity, associativity and idempotency. Then there is a partial ordering $\leq$ on X, such that ■ is just taking infimum with regard to $\leq$.

Proof: Let X and ■ be given. Define $\leq$ by $x \leq y \Leftrightarrow x = x \blacksquare y$. We leave it to the reader to show that (1) $\leq$ as defined is a partial ordering (2) ■ is taking infimum with regard to $\leq$. □

Of course more general functions can be considered when some of the properties are not assumed. In [Kl43] it is proven that theorem 9.1 holds also for operators that satisfy the weaker ($a = b \blacksquare x$ **and** $b = a \blacksquare y$) $\Rightarrow a = b$ instead of idempotency.

Acknowledgements: I am indebted to the members of the Utrecht Monday Morning Club (MOC) for their useful remarks and ideas.

10 References

[BMR85] Beilken, C., F. Mattern, and M. Reinfrank, Verteilte Terminierung – ein Wesentlicher Aspekt der Kontrolle in verteilten Systemen, Bericht nr 41/85, Fachbereich Informatik SFB 124, University of Kaiserslauten, Kaiserslauten, West Germany, 1985.

[BT84] Bracha, G., and S. Toueg, A distributed algorithm for generalized deadlock detection, Proc. 3rd Annual ACM conference on Principles of Distributed Computing, Ottawa, 1984, pp 295–301.

[CM85] Chandy, K.M., and J. Misra, A paradigm for detecting quiescent properties in distributed computations, Report TR–85–02, University of Texas at Austin, Austin, Texas.

[Fr80] Francez, N., Distributed termination, ACM ToPLaS 2 (1980), 42-55

[HMR86] Helary, J-M, A. Maddi, M. Raynal, Controlling knowledge transfers in distributed algorithms, application to deadlock detection, Publication Interne 278, IRISA, Rennes, France.

[Hu85] Hughes, J., A distributed garbage collection algorithm, in: J.P. Jouannoud (ed), Functional programming languages and computer architecture, Lecture Notes in Computer Science vol. 201, Springer Verlag, Heidelberg, 1985, pp. 256-272.

[Je85] Jefferson, D., Virtual Time, ACM ToPLaS 7 (1985), 404–425.

[Kl43] Klein-Barmen, F., Über gewisse Halbverbände und Kommutative Semigruppen, Teil I, Mathematisches Zeitschrift 48 (1943) 275–288.

[Ra83] Rana, S.P., A distributed solution to the distributed termination problem, Inf. Proc. Lett. 17 (1983), 43-46.

[SL87] Sarin, S.K., N.A. Lynch, Discarding Obsolete Information in a Replicated Database System, IEEE Transactions on Software Engineering SE–13 (1987) 39–47.

[TL86] Tan, R.B., and J. van Leeuwen, General Symmetric Distributed Termination Detection, Report RUU–CS–86–2, Department of Computer Science, University of Utrecht, Utrecht, The Netherlands (submitted to Computers and Artificial Intelligence).

[TT87] Tel, G., R.B. Tan, The Equivalence of Some Network Problems, notes July 1987.