Reliability versus cost: design of a probabilistic broadcast algorithm

Augusto Ciuffoletti
Department of Computer Science - University of Pisa - Italy

Abstract

We propose a probabilistic algorithm to solve the problem of distributed broadcast. A simple diffusion algorithm is introduced, and its reliability is evaluated. The cost and reliability of the probabilistic algorithm are compared with the corresponding deterministic algorithm.

Key words: broadcast, probabilistic algorithms, reliability, random diffusion, geometric topology.

Mailing address:
Augusto Ciuffoletti
Università degli Studi di Pisa - Dipartimento di Informatica
Corso Italia n. 40
56100 Pisa - ITALY

Telex #: 590291 DIPISA I
teleFAX #: +39 - (0)50 - 510226
E.Mail: augusto@di.unipi.it

1. Introduction

Algorithms can be divided into two classes, according to whether their *reliability* (the fact that their behavior fulfils the requirements) is expressed by a boolean value, or by a probability measure (i.e., a real number in the range *[0,1]*). We call the former class of algorithms *deterministic*, and the latter class *probabilistic*. A deterministic algorithm is reliable if its behavior fulfils the requirements under every (permitted) execution scenario. In this case reliability is

equivalent to correctness. The reliability of a probabilistic algorithm, on the other hand, is ρ if, within all permitted scenarios, out of n executions of the algorithm, $\rho \cdot n$ executions fulfil the requirements and $(1-\rho) \cdot n$ do not, as n tends to infinity.

From the definition outlined above, one could argue that a probabilistic algorithm introduces a further source of unreliability, without obtaining any benefit with respect to its deterministic counterpart. This opinion may change if we consider how well deterministic and probabilistic algorithms can cope with extremely non-deterministic environments, and at what cost.

In fact, an algorithm that implements a distributed activity (a *distributed algorithm*) runs in an extremely non-deterministic *environment*, due to communication delays and the possibility of failures. A distributed algorithm is designed to obtain a certain result assuming that the behavior of the environment is acceptable: the concept of acceptability may be expressed by a set of requirements usually embodied in a *system model* describing the expectations of the designer of the algorithm concerning the behavior of the environment. A *failure* occurs whenever the behavior of the system is not consistent with the system model. The reliability of the pair ”distributed algorithm” plus ”environment” depends not only on the correctness of the algorithm, but also on the strength of the requirements. The stronger they are, the less reliable the whole is.

A deterministic algorithm treats the set of the acceptable behaviors as a flat set: the algorithm obtains the desired result under any behavior that satisfies the requirements (see a taxonomy of requirements that allow the units to reach deterministic agreement in [7] and [6]). This usually makes the requirements rather strong: e.g., in clock synchronization the communication delay needs to have an upper bound [10]. On the contrary, a probabilistic algorithm tunes its reliability (i.e., the probability of obtaining the desired result) on the likelihood of the various behaviors, in order to ensure high reliability in the most frequent ones, and lower reliability in the others. This flexibility allows the designer to relax the requirements, and often improves the overall reliability. In addition, a probabilistic algorithm is simpler and less expensive than a deterministic one with similar reliability, since it does not need to cover provable infrequent special cases that would require ”exceptional” computations: this explicit avoidance of treatment of troublesome behaviors makes the program simpler and more reliable (in terms of the probability of introducing a bug in the program). An example is the probabilistic clock synchronization algorithm in [3].

The behavior of the whole “environment” plus “probabilistic algorithm” is close to that of a hardware component: usually (extremely) predictable, but admitting unpredictable failures. In this case, unpredictability means that the whole may fail even if the behavior of the environment is acceptable and the algorithm does not contain bugs, but some combination of circumstances (too complicated to be predicted) results in the failure. Therefore, the whole “environment” plus “probabilistic algorithm” behaves in turn as a non-deterministic environment: the level structuring widely used for deterministic algorithms is applicable to probabilistic ones as well.

Unfortunately, the design of a probabilistic algorithm raises a further critical point that may increase the unreliability. Namely, the design of the probabilistic model that describes the behavior of the environment. The expected reliability of the whole "environment" plus "probabilistic algorithm" is computed as a function of this model; if the model is not appropriate, the reliability may be different from the expected one. If the estimate of the reliability is higher than "reality", one may obtain an unexpectedly unreliable whole which may lead to a kind of software bug.

Simulation is of little or no help in these cases, since we are interested in evaluating the probability of extremely infrequent situations: one would have to run millions of experiments to obtain reasonable estimates of the reliability. Therefore, a precise analytic model of the environment must be built. This requirement may exclude the application of probabilistic algorithms to solve certain problems. But some interesting cases admit a sufficiently simple model.

The purpose of this paper is to give an analytic case study that can justify the above abstract issues. Section 2 introduces some basic terminology and concepts concerning the system model and defines the broadcast problem. Section 3 outlines the algorithm under some restrictive hypotheses and gives a probabilistic model of the behavior of the algorithm running on a grid mesh. Section 4 removes the restrictions introduced in Section 3 and gives an applicable algorithm.

2. System model

The *system* consists of a set of *nodes*. Each node has access to a *clock* or *time reference*, in order to coordinate its activity with the rest of the system. A *processor* runs the activity that characterizes the node. The activity may need to exchange information with other activities running on neighbor nodes. This capability of exchanging information with neighbor nodes is implemented in a specialized unit, called the *communication handler* (CH).

A CH is in fact a multi-task processor. The CH is typically shared by the manager of the queues of the messages to and from the activity supported by the processor, and by the tasks that interface the physical links. This paper focuses on a particular service offered by the CH to the resident processes: the delivery of a message to every other node, namely, a *broadcast*.

2.1. The time reference

Every node in the system can obtain the time from the same time reference. The time is read with an error of ε time units; this means that if the CH reads a clock value T, the real time is within the range $[T-\varepsilon, T+\varepsilon]$. This kind of clock synchronization is termed "external

synchronization" in [3]: depending on the design of the system, external clock synchronization may be implemented either by hardware [9] or by software [3].

The clock value can be read at any time, but cannot be updated. The granularity of the time reference is sufficient to separate the execution of two consecutive instructions by the same CH and the value of T is monotonic with respect to the order in which instructions are performed.

2.2. The communication sub-system

A *link* is a communication medium whose role is to enable the exchange of information between two CHs connected by it. If two CHs are connected by a link they are said to be *adjacent*. The entire communication sub-system may be viewed as a graph where the CHs correspond to vertexes, and the communication links to the edges. We assume that the communication links are bidirectional and therefore the corresponding graph is non-directed.

The definition of link only concerns the physical structure of the system, and does not specify if the link is in fact used, or whether the linked nodes or the link are working properly or not. In order to distinguish between those links that are operational from those that are not, we assume the presence of a *background message flow*: each pair of CHs connected by an operational link communicates at least every δ time units in each direction. The background message flow is also characterized by statistical properties: in Sect. 3.2. we give a case study and the related probability model.

The background message flow is normally maintained by the communications required by the application run by the system. If a link is left temporarily unused by the application but is to be considered as operational, it is the job of the two CHs attached to its ends to send a dummy message at least every δ time units in both directions. We call a link that supports this minimal traffic requirement *operational*. The set of operational links may change in time because of planned or unplanned events, therefore the definition of an operational link depends on time:

*A link is **operational** at time T if and only if it carried at least one communication in each direction during the time interval [T-δ, T]. If two adjacent CHs are connected by an operational link they are called **neighbors**.*

The background flow is used by the CHs to determine whether a link is operational or not. Each time a CH receives a communication along a link L, it sets a timeout to occur after a time Δ_{detect}[1] . If no new messages are received along L before the timeout expires, L is considered to

---

[1] The value of Δ_{detect} depends not only on δ, but also on the distribution of the communication delay, and on the precision of the timeout mechanism. However, it holds that $\Delta_{detect} \geq \delta$, since the CHs at the receiver's end must wait at least δ time units before concluding that the incoming link is inactive.

be non-operational. The CH cannot determine whether the failure is due to a performance failure of the link, or to a silent failure of the (ex-)neighbor, or to a deliberate de-activation of the link by the (ex-)neighbor.

The background message flow and its statistical properties are fundamental to the broadcast algorithm described in Section 3.

2.3. Scheduling and communication primitives

The CH performs a number of concurrent activities that are typically related to the message queues and communication links. The sharing of the CH hardware among these activities is controlled through a set of primitives, which are invoked by activities running on the same or on a neighbor CH. For example, an activity may request the execution of another activity at a given time. CHs invoke activities on each other using remote procedure calls [1].

The primitives that control the CH activity are summarized in the following pseudo Pascal procedures and functions:

procedure Deliver (M:message);

the message *M* is en-queued to the destination; the name of the destination is contained in the message itself.

procedure Schedule (C: ProcedureCall; T: time);

the call *C* is scheduled to be executed at time *T*; the scheduling algorithm ensures that the task is fired when the local clock reads a time within the range $[T,T+\tau_{sched}]$;

procedure Call (C: ProcedureCall; Destination: CHName);

the call is added to the tasks scheduled to be executed on the referenced CH. The procedure returns as soon as the request is issued and does not wait for acknowledgements from Destination;

procedure Cancel (C: ProcedureCall);

any call to procedure *C* is cancelled from the set of scheduled tasks.

2.4. Specification of the broadcast

We now define the broadcast problem that we are trying to solve. The *origin* of the broadcast is the CH that invokes the broadcast operation. After the broadcast has been invoked, we call *informed* those CHs that have already received the broadcast (and the origin is one of them). A more formal specification is:

$informed: S \times T \rightarrow \{true,false\}$

where *S* is the set of all CHs in the system and *T* is real time. The function *informed* describes the dynamic behavior of the broadcast: at a given time *T* it is *true* for those CHs that have received the information, *false* for those that have not. At time T_0 when the broadcast starts, just one CH is informed:

$$informed(X,T_0) = \begin{cases} true & if\ X\ is\ the\ origin\ of\ the\ broadcast \\ false & otherwise \end{cases}$$

At time $T_0+\Delta_b$ when the broadcast terminates, we want the following Weak Atomicity requirement to hold:

Weak Atomicity (WA):

$$informed(X,T_0+\Delta_b) = \begin{cases} true & if\ X\ is\ a\ neighbor\ of\ Y\ and \\ & informed(Y,T_0+\Delta_b) = true \\ false & otherwise \end{cases}$$

The definition we have just given needs some further analysis, since it differs from definitions used by other authors. The following are the widely accepted requirements of an atomic broadcast (the definition is adapted from [4]):

Termination

Every message broadcast by a correct process at time T_0 is delivered by every correct process by time $T_0+\Delta$;

Atomicity

Any message broadcast at time T is either delivered to every correct process by time $T_0+\Delta$, or is never delivered to any process;

Order

All delivered messages are delivered in the same order at all correct processes.

This definition is usually based on this additional system requirement, which is embodied in the system model:

Non partitioning

During the broadcast, the sub-system of correct processes is not partitioned by process or link failures.

Under this system requirement, it is easy to show that the WA adopted in this paper is equivalent to the termination and atomicity alone.

Proof: the transitive closure of the neighbor relation is an equivalence relation; one of the equivalence classes so defined is the sub-system of CHs that must receive the broadcast (they are correct at time T for the classic definition), and the "non partitioning" requires that the class does not split during the execution of the algorithm. WA requires that all or none of the CHs in this class receive the broadcast by time $T_0+\Delta_b$ (which is equivalent to the termination property with $\Delta = \Delta_b$).

But WA is also meaningful if link or CH failures generate a partition in the sub-system of the CHs that were correct and connected at time T. In that case, termination and atomicity properties would be hard to ensure: if the system partitions, there is no way to spread the information from one partition to the other (to ensure termination) or to determine if the information has been diffused in every CH of each partition (to ensure atomicity). We will see that WA can be ensured even if the system is partitioned.

Finally, the order property can be implemented as an additional service if the WA requirement is satisfied, and may therefore be omitted from the broadcast service. From this point of view, the algorithm we propose follows the structure of the one proposed in [12]. Section 6 will

indicate a way to implement a total ordering service starting from an algorithm that guarantees WA.

The above discussion justifies the use of the Weak Atomicity as the basic requirement that a broadcast algorithm should satisfy.

3. The background diffusion algorithm

This section describes the background diffusion algorithm, a probabilistic broadcast algorithm that exploits the existence of a background message flow. To simplify, we will not consider details that are not essential to the understanding of the algorithm, and we thus make the following assumptions:

- $\varepsilon = 0$ access to the global clock is error-free;
- $\tau_{sched} = 0$ tasks scheduled to be executed at a certain time are fired exactly at that time;
- at each time there is at most one on-going broadcast diffusion;

Section 4 outlines the modifications required to relax these assumptions. As we will see, these modifications involve adjusting the timeouts, without affecting the structure of the algorithm.

Background diffusion is implemented by piggybacking the information to be broadcast onto every message of the background message flow sent from CHs that have either received the information piggybacked onto some previous message, or have originated it. To enable the timely operation of the informed CHs, the broadcast information piggybacked to the messages also contains the time when the broadcast originated. This information is determined by the origin of the broadcast which reads the value of the clock. We shall refer to this information as the *Time Identifier* (tid) of the broadcast. Note that the background diffusion algorithm does not require the generation of explicit messages to spread the broadcast [4,5]. Instead, it uses application messages, whose frequency and distribution are described by a probabilistic model. A number of improvements may be added to this very basic algorithm, for instance avoiding sending the same broadcast several times to the same neighbor; but here we are interested in the explanation of its basic features.

The broadcast information spreads throughout the system carried by the background message flow, and reaches every CH in the system connected to the origin in less than $\delta \cdot r$ time units, where r is the diameter of the system graph. Generally the WA requirement is satisfied well before that time, since we estimate that typically two neighbors exchange information far more frequently than every δ time units. If we indicate with p_Δ the probability that WA is satisfied after Δ time units, we can re-formulate the above statement saying that p_Δ is reasonably close to 1 for values of Δ far smaller than $\delta \cdot r$. Therefore we choose an appropriate value Δ_b that meets the desired reliability ρ ($p_{\Delta_b} \geq \rho$), and design the algorithm so that the CHs participating in the broadcast stop spreading the information at time $T_0+\Delta_b$. There is a residual case of probability

$1-p_{\Delta_b}=q_{\Delta_b}$ that the requirement WA is not met at that time; the CHs should detect that situation in order to invoke some further action to enforce WA.

The CHs detect the *failure* of the algorithm by continuing the background diffusion after time $T_0+\Delta_b$ and by exploiting the following result:

Lemma (detection):
During a background diffusion, at time $T-\Delta_{detect}$ requirement WA holds if informed(R, $T-\Delta_{detect}$) =informed(R,T) for every R neighbor of an informed CH at time T, provided that from time $T-\Delta_{detect}$ to time T no informed CH acquires new neighbors.

Proof: Let us first prove the lemma when the set of operational links remains unchanged in the time interval $[T-\Delta_{detect},T]$. If no new CH has been informed in the last Δ_{detect} time units, any message exchange during this time interval was between two informed CHs. Since $\Delta_{detect}\geq\delta$ (see footnote in sect. 2.2.), a message flowed on every link connecting two neighbors (operational link def.). Therefore, if a neighbor was informed, so was the other (WA requirement).

If some link operational at time $T-\Delta_{detect}$ becomes non–operational before time T, the lemma is valid both if the neighbor CH is non-informed and if it is informed, since the lemma binds the state of neighbor CHs at time T.

(Q.E.D.)

The lemma excludes the case that new links become operational during the time interval $(T-\Delta_{detect}, T)$: if a non-informed CH becomes a neighbor of an informed one during this time interval, it might remain non-informed. Therefore, during this time interval, an informed CH is not allowed to change the state of a link from non-operational to operational. But the lemma is valid if some link is inactivated during the time interval involved.

The failure detection rule used by the CH, derived from the above lemma, is:

Any CH that moves from the non-informed state to the informed state during the time interval $(T_0+\Delta_b, T_0+\Delta_b+\Delta_{detect})$ detects an algorithm failure.

This failure detection rule does not guarantee that by time $T_0+\Delta_b+\Delta_{detect}$ every CH non-informed at time $T_0+\Delta_b$ has detected the failure, but that every CH that is a neighbor of an informed CH is either informed, or has detected the failure.

Therefore those CHs that detected the failure form a boundary between the set of informed and non-informed CHs at time $T_0+\Delta_b$. They move to an exceptional state, and do not diffuse the broadcast.

It is possible to prove that at time $T_0+\Delta_b+\Delta_{detect}$ the CHs that have detected the failure know which of the neighbors are informed, and which are not: to enforce WA, they set non-operational their links with informed CHs. As a consequence, all non-informed CHs are no longer neighbor to any other informed CH, and the WA requirement is now fulfilled.

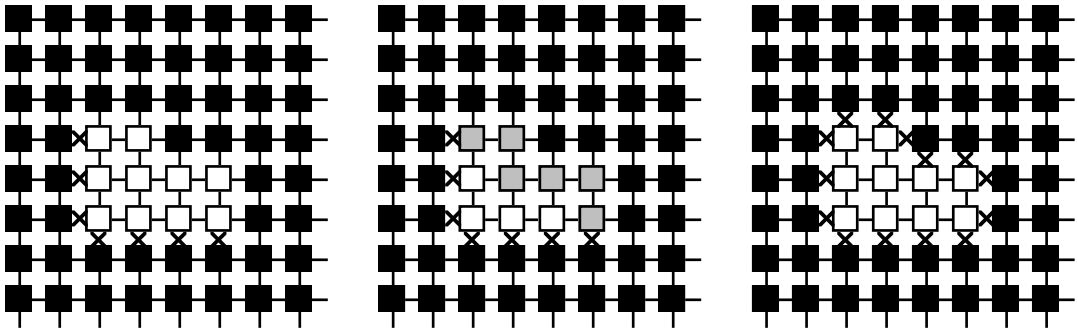


Fig. 1 (a,b,c).The broadcast fails to reach every reachable unit.

Figure 1 shows the application of the failure detection rule in a critical situation occurring in a system where CHs are connected as a mesh. Figure 1a shows the situation at time $T_0+\Delta_b$: since several links are non-operational (represented by crossed lines) some nodes (represented by white boxes) are still non-informed. In Figure 1b shaded boxes represent CHs that detect a failure at time $T_0+\Delta_b+\Delta_{detect}$ using the failure detection rule. In Figure 1c new links are set non-operational as a consequence of the failure detection.

The next section focuses on the problem of reconnecting those links that have been disconnected due to an algorithm failure

At time $T_0+\Delta_b+\Delta_{detect}$ the algorithm terminates and the informed CHs that have not detected a failure can stop participating in the broadcast algorithm: the information is delivered to the destination and no longer piggybacked to the messages.

Let us summarize the timing of the background diffusion algorithm:

At T_0:

a communication handler receives the request to broadcast an information to every other communication handler;

From T_0 to $T_0 + \Delta_b$:

the information is broadcast by piggybacking it to background communications;

From $T_0 + \Delta_b$ to $T_0 + \Delta_b + \Delta_{detect}$:

the diffusion algorithm is continued to detect communication handlers that were not informed at time $T_0+\Delta_b$. Each communication handler that becomes informed during this time interval detects the algorithm failure and moves into the disconnected state: it will not diffuse the broadcast;

At $T_0 + \Delta_b + \Delta_{detect}$:

the diffusion terminates, and each informed communication handler delivers the information to the destination process. Communication handlers that detected a failure disconnect themselves from informed communication handlers that did not detect the failure.

Whenever the algorithm is run, its behavior non-deterministically falls into one of three distinct categories:

proper: the WA property is satisfied before $T_0+\Delta_b$. The broadcast reaches every reachable CH; the probability of proper behavior is $p_{\Delta_b} = P(T_{wa} \leq T_0+\Delta_b)$, where T_{wa} is the first time the WA property is satisfied.

improper: the WA property is satisfied at $T_0+\Delta_b+\Delta_{detect}$. The broadcast does not reach every reachable CH, and the system loses some computing power to restore WA. The probability of improper behavior, provided that no unacceptable events occur, is $q_{\Delta_b} = P(T_{wa}>T_0+\Delta_b) = 1-p_{\Delta_b}$.

unacceptable: the WA property is not satisfied at $T_0+\Delta_b+\Delta_{detect}$. The algorithm fails and the system state becomes inconsistent. The cause of an unacceptable behavior is the occurrence of one or more system failures (e.g., message disruption). The probability of an unacceptable behavior is considered negligible in comparison with the others.

The program in the Appendix should give a more precise view of the implementation of the failure detection rule and of the background diffusion.

3.1. Concerning the destiny of disconnected CHs

The CHs that detect an algorithm failure spontaneously set their links as non-operational, and enter an exceptional *disconnected* state. Disconnected CHs must be able to recover from this state as soon as possible, since they have been affected by a transient failure of the broadcast algorithm. Here we outline an algorithm that ensures a quick recovery of disconnected CHs.

The broadcast timing ensures that a disconnected CH knows when the broadcast algorithm terminates, and which of the neighbors are informed and have not detected the failure. When the broadcast terminates, the disconnected CHs form a kind of firewall between the part of the system that has been informed and the part or parts that have not been informed.

The next step performed by the disconnected CHs consists of diffusing the disconnection through the links that connect them to non-informed CHs: thus they take up the diffusion during another Δ_{detect} time units after the original broadcast algorithm times-out (see program in the Appendix). After this time, they can conclude that at least another firewall has been set: in fact, every non informed CH directly linked to one of them received the broadcast and entered the disconnected state. Let us call 1-disconnected those CHs that were disconnected at the beginning of this round, and 2-disconnected those that are disconnected by this further broadcast.

The 1-disconnected CHs can now enter an intermediate state called *ready*, and ask for reconnection to the informed neighbors. The 2-disconnected will follow the same protocol that the 1-disconnected did. Therefore, 2-disconnected CHs may generate 3-disconnected CHs.

Every CH will be informed after a non-deterministic number of rounds. Figure 2 illustrates the 1-, 2-, and 3-disconnected CHs in the sample system.

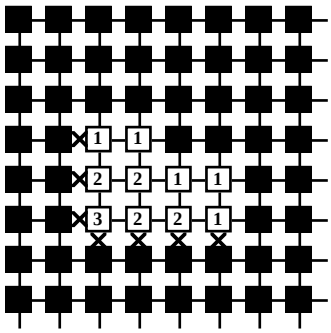


Fig. 2. CHs reconnected in successive rounds

As already mentioned, reconnection cannot happen when the informed CH to which the disconnected one has requested the reconnection is in the detection phase. Therefore, the ready CH must tolerate that the response to its request be delayed by more than Δ_{detect} time units. The response to the reconnection request contains the data necessary to the disconnected CH to restore a state consistent with the global state. The implementation of this operation was extensively studied during the 70s [8].

Now we focus our attention on the computation of the parameters that characterize the diffusion and failure detection phases.

3.2. A probabilistic model of the background diffusion process

We mentioned in the introduction that, to make a probabilistic algorithm practically applicable, its behavior must be modeled in order to quantify the probability of certain classes of behavior. These quantities are then checked against the requirements of the algorithm.

In our case, the requirements are expressed as the probability that an execution of the algorithm exhibits a proper, improper and unacceptable behavior, as defined above. For concreteness, we require that a proper behavior have probability greater than 10^{-3}, and an unacceptable behavior have probability less than 10^{-6}. The probability of an unacceptable behavior gives the measure of required reliability, since we assume that a system that experiences an unacceptable behavior of the algorithm may become inconsistent.

In addition, the model must be related to a network topology, since the behavior of the algorithm depends on this feature. In this section we describe a model for a square grid mesh of N_C nodes closed in a torus, and we will give numeric results for the case of an 8x8 grid.

The background flow is modeled as a random process $\boldsymbol{M}$ (throughout this section, random variables and processes are indicated in **bold**). We assume that it is homogeneous both in time and space: i.e., the probability of supporting a message exchange during a certain time interval is identical for any link, and independent from the previous history of the system. This assumption reflects the ideal case of a uniform distribution of a constant communication load, and is a reasonable model of the behavior of a homogeneous system over extended time periods. It is not intended to describe transient peak loads or systems containing links that permanently exhibit an anomalous communication load.

Under the above assumption, process $\boldsymbol{M}$ is a Poisson process. To normalize it with respect to the dimension N_c of the system, we assume that during each time unit the system supports on the average N_c messages: the time $\Delta\boldsymbol{t}$ elapsing between two successive message exchanges originated by a given CH therefore has an exponential distribution with a mean of one time unit.

Given the above assumptions about the system, we want to determine the two design parameters:

Δ_{detect} the time needed to detect a neighbor's failure, i.e. the maximum time interval permitted between two successive messages on the same operational link;

Δ_b the maximum time permitted to diffuse the information to every CH in the system.

Both time limits indicate the proper behavior of the link utilization and the broadcast algorithm respectively: therefore, both of them should be respected with a probability greater than 10^{-3}. We determine first the value of Δ_{detect}. Since the interval between two successive message exchanges originated by a node has an exponential distribution with a mean of one, the interval between two successive messages originated by a node on a certain link (indicated by $\Delta\boldsymbol{t}_{\mathbf{link}}$) has an exponential distribution with a mean of four time units, since each message is destined to one of the four links chosen at random. Since we require

$$P\{\Delta\boldsymbol{t}_{\mathbf{link}} \geq \Delta_{detect}\} \leq 10^{-3}$$

we conclude that

$$\Delta_{detect} \geq -4 \ln(10^{-3}) \text{ time units} \approx 28 \text{ time units}$$

Note that the success of the broadcast algorithm is not affected by the improper behavior of a link: nevertheless we have to establish a timeout to detect failures.

To compute the value of Δ_b we must evaluate the distribution of the time $\boldsymbol{T}_{\mathbf{sat}}$ needed to diffuse the information to every CH. In other words, we want to define the probability distribution of the random variable $\boldsymbol{T}_{\mathbf{sat}}$ in order to find a value Δ_b such that:

$$P\{\boldsymbol{T}_{\mathbf{sat}} \geq \Delta_b\} \leq 10^{-3}.$$

Since we know that the distribution of the period $\Delta\boldsymbol{t}$ between successive message exchanges does not vary with time, we can rewrite $\boldsymbol{T}_{\mathbf{sat}}$ as:

$$\boldsymbol{T}_{\mathbf{sat}} = \frac{1}{N_c} \sum_{k=1}^{\boldsymbol{E}_{\mathbf{sat}}} \Delta\boldsymbol{t} \quad (1)$$

where $\boldsymbol{E}_{\mathbf{sat}}$ is a random variable representing the number of message exchanges needed to inform every CH in the system, and $\Delta\boldsymbol{t}$ has an exponential distribution with mean one time unit.

The distribution of $\boldsymbol{E}_{\mathbf{sat}}$ depends on the way the information diffuses in the system, not on the timing of the system; its distribution can be evaluated using a Markov chain that describes the progress of the system from a state where just one CH is informed, to the state where all CHs are informed. This model gives an exact evaluation of the probability distribution, but is practically

intractable, since the number of different states in the Markov chain is exponential with respect to N_c. Below we outline this model and verify that it is intractable. Then we introduce a simpler model that approximates the exact one, and we use it to determine Δ_b.

The exact Markov chain $\boldsymbol{Ex}$ contains one state for each non-empty subset of CHs that corresponds to a connected sub-graph of the system, and we indicate with $\boldsymbol{Ex_e}=s_i$ the event that the system is in state s_i after e message exchanges. In other words, each state s_i represents a stage of the diffusion process, where all CHs in s_i are informed, and the rest are not. Let S be the set containing all the states of the system. Among the elements of S we identify the initial states, which correspond to the singletons, and the final state, that contains all the CHs.

The probability of transition is defined only for those pairs of states that represent the diffusion of the information to one non-informed CH after one message exchange, and for those that represent the absence of such diffusion. Formally, if we denote by # the unary operator that returns the cardinality of a set, the probability of transition from state s_i to state s_j, $P_{ij}^{\boldsymbol{Ex}}(e\text{-}1,e)=P\{\boldsymbol{Ex_e} = s_j \mid \boldsymbol{Ex_{e-1}} = s_i\}$, is defined if and only if $\#(s_j - s_i) \leq 1$.

The transition probabilities are set as follows:

$$P_{ij}^{\boldsymbol{Ex}}(e\text{-}1,e) = \begin{cases} \frac{k_{ij}}{4Nc} & if\ i \neq j \\ \frac{4N_c - p_i}{4N_c} & if\ i = j \end{cases} \quad (2)$$

where k_{ij} is the number of links from the only CH in s_j-s_i and the CHs in s_i, and p_i is the number of links from CHs in s_i to CHs not in s_i.

By applying the discrete Chapman-Kolgomorov equation we can compute the probability to move from an initial state s_{init} to the final state s_{final} with a given number k of message exchanges in the Markov chain $\boldsymbol{Ex}$; this is the same as the probability distribution of $\boldsymbol{E_{sat}}$:

$$P_{init\ final}^{\boldsymbol{Ex}}(1,k)=P\{\boldsymbol{E_{sat}} \leq k\}$$

Unfortunately, the number of states in the $\boldsymbol{Ex}$ model is exponential. To verify this, consider all paths of length $2\sqrt{N_c}$ -1 starting from the upper left corner and arriving at the lower right corner (in a planar unfolding of the torus) and moving, at each step, only south or east. The number of such distinct paths is given by

$$\binom{2(\sqrt{N_c}-1)}{\sqrt{N_c}-1}$$

In Figure 3 are the $\binom{4}{2}=6$ possible paths when $N_c = 9$.

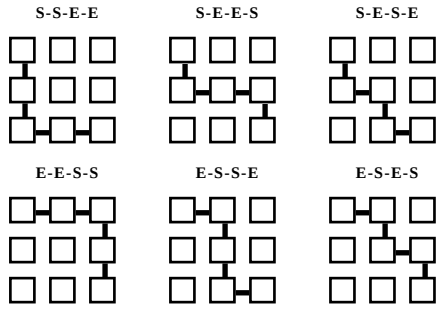


Fig. 3. Paths from (1,1) to (3,3). On top of each path is its signature (S=south, E=east)

This corresponds to a number that grows exponentially with N_C. We can associate with each path the set of CHs that it crosses. Each of these sets is necessarily a state in the ***Ex*** model, since it is a connected sub-graph of the system. Therefore the ***Ex*** model contains a number of states which is exponential with respect to N_C.

We want to reduce the complexity of the model, but we require that the new model give a pessimistic evaluation of $\boldsymbol{E}_{\mathbf{sat}}$, in the sense that, in the new approximate model ***Apx***:

$$P^{\boldsymbol{Apx}}_{1\ N_C}(1,k) \leq P\{\boldsymbol{E}_{\mathbf{sat}} \leq k\} \quad (3)$$

Studying the geometric properties of the area covered by informed CHs we can find a function that gives a lower bound on the number of links that connect informed to non-informed CHs, depending only on the number of informed CHs (we recall that for a state s_i the number of informed CHs is $\#s_i$).

Statement A*: For each state s_i the function π defined as follows:*

$$\pi_{\#s_i} = \begin{cases} 4\ Floor(\sqrt{\#s_i}) & if\ \ \#s_i < \frac{N_C}{4} \\ 2\sqrt{N_C} & if\ \ \frac{N_C}{4} \leq \#s_i \leq \frac{3N_C}{4} \\ 4\ Floor(\sqrt{N_C - \#s_i}) & if\ \ \#s_i > \frac{3N_C}{4} \end{cases}$$

is such that $\pi_{\#s_i} \leq p_i$, where Floor(x) is the greatest integer less than or equal to x and p_i is the number of links from CHs in s_i to CHs not in s_i .

The proof of Statement A is given in the Appendix. From (2), in the ***Ex*** model the probability of remaining in the same state depends on p_i: in that formula, if we substitute π_i for p_i we obtain an upper bound for the probability of remaining in the same state that depends only on the number of informed CH in that state. A model based on this substitution would therefore be a pessimistic model in the sense defined in (3). We can now simplify the ***Ex*** model by grouping states into equivalence classes. All states with the same number of informed CHs belong to the same class, and we define the probability of moving from a state of a class to a state in the following class (in the total ordering which is now induced) or of remaining in the same class. The corresponding Markov chain ***Apx*** is composed of N_C states $\sigma_1 \ldots \sigma_{Nc}$, each representing the equivalence class that groups states with 1 to N_C informed CHs. Thus, the initial and the final states are σ_1 and σ_{Nc}, respectively, and the transition probabilities are defined in accordance with (2), as:

$$P_{ij}^{\boldsymbol{Apx}}(e\text{-}1,e) = \begin{cases} \frac{\pi_i}{4N_c} & \text{if } i=j+1 \\ \frac{4N_c-\pi_i}{4N_c} & \text{if } i=j \end{cases} \quad (4)$$

The rule used to build the ***Apx*** model guarantees that it is a pessimistic model for the number of message exchanges to diffuse the information to every CH and therefore the requirement expressed in (3) is satisfied. This model contains N_c states (instead of the exponential number of states in the exact model) with $2N_c$ -1 edges, and can be used to obtain a pessimistic estimate for the time needed to terminate the broadcast. In fact, the following inequality holds:

Statement B: $P\{\boldsymbol{T_{sat}} \geq \Delta_b\} \leq 1 - P_{1\,N_c}^{\boldsymbol{Apx}}(1,\Delta_b N_c)$ *when* Δ_b *is an integer*

A proof of the inequality is given in the Appendix, together with a comparison between the numerical estimates obtained by the pessimistic model and the results obtained by simulating the broadcast algorithm in a system with $N_c = 64$.

Using the Chapman-Kolgomorov equation we can evaluate the expression on the right hand side of the inequality, and obtain a pessimistic evaluation of the probability that the duration of the diffusion process is more than Δ_b time units (see Table 1).

Δ_b	33	34	35	41	42	43
$1 - P_{1\,N_c}^{\boldsymbol{Apx}}(1, \Delta_b N_c)$	$1.1\ 10^{-3}$	$4.5\ 10^{-4}$	$2.6\ 10^{-4}$	$2.1\ 10^{-6}$	$9.2\ 10^{-7}$	$3.9\ 10^{-7}$

Table 1. Probability that the duration of a broadcast exceeds Δ_b time units.

Since the reliability requirements are that $P\{\boldsymbol{T_{sat}} \geq \Delta_b\} \leq 10^{-3}$, we choose a Δ_b of 34 time units.

The time needed to complete the probabilistic algorithm is $\Delta_b + \Delta_{detect}$ or 32+28 = 62 time units. As expected, this is far below 224, the time ($\Delta_{detect} \cdot r$) needed to complete the deterministic algorithm (remember that r is the diameter of the system). On the other hand, the deterministic algorithm gives a cleaner response since it prevents some nodes from being excluded from the system, while the probabilistic algorithm does not ensure that every node is informed. The designer is thus faced with the question: “Is it worth adopting an algorithm that takes four times longer to avoid the (infrequent) event that one node is (safely) excluded from the system?”. If the answer is no, the probabilistic algorithm is more suitable than the deterministic one; if the answer is yes, we can still make the probabilistic algorithm more reliable: in our case (see Table 1), if we choose $\Delta_b + \Delta_{detect}$=70, the probability that a node is safely excluded is 10^{-6}. The above dilemma is posed again: now the probability of exclusion of a single node is equal to the accepted probability of unsafe failure, and the execution time of the probabilistic algorithm is still less than one third of the time needed by the deterministic algorithm.

4. Relaxing the assumptions

The algorithm described above was designed under restrictive assumptions. In this section we modify the algorithm in order to relax the assumptions

$\varepsilon = 0$ the access to the global clock is error-free;

$\tau_{sched} = 0$ tasks scheduled to be executed at a certain time are fired exactly at that time;

at each time there is at most one on-going broadcast diffusion.

As we shall see, the generalization of the algorithm merely consists of adjusting the timeouts by taking into account the presence of delays.

4.1. Inaccurate synchronization

In order to carry out informal proofs, we introduce a graphic representation of the relation between clocks and real time [13]. In this representation (see Figure 4) at a certain time the timing mechanism of each node is modeled as a tuple (*time*,*clock*). The first coordinate represents real time, and the second the clock value of the node at that time. If we relax the $\varepsilon = 0$ condition, admitting that there may be a difference between the value of a clock and real time, bounded by some constant ε, then all points representing node clocks must be within the strip between the equations $y=x+\varepsilon$ and $y=x-\varepsilon$.

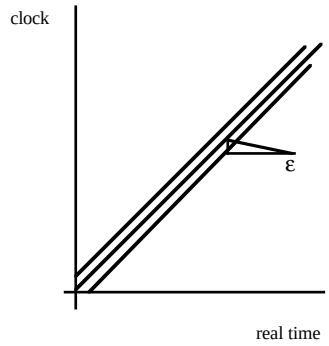


Fig. 4. Time graph.

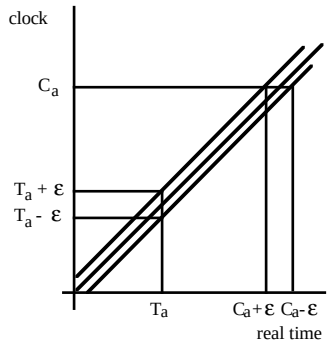


Fig. 5. Timing with bound error.

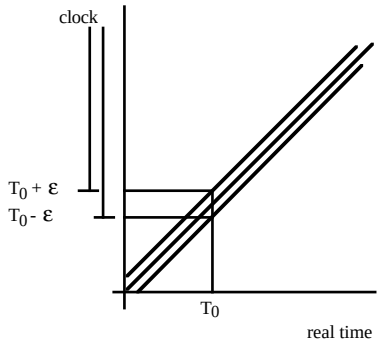


Fig. 6. Uncertainty about the tid

Therefore, given a certain real time T_a, we can graphically indicate that every node has a clock value in the interval (T_a-ε ,T_a+ε), or, conversely, that if a node reads the clock value C_a, then the real time is inside (C_a-ε, C_a+ε) (see Figure 5).

Parallel to the *Y* axis we shall represent two "histories", or sequences of events, as observed and timed by the nodes using their clocks as a reference. The corresponding *X* coordinates represent the real timing of these events. Since all nodes produce the same events at the same clock value, the real times of the events will spread in an "uncertainty" area.

If the clock error is bounded by a constant quantity ε, it can be shown that the previous algorithm ensures the WA at time:

$$T_1 = T_0 + \Delta_b + \Delta_{detect} + 2\varepsilon$$

The following considerations justify the result.

The diffusion originates at time T_0; therefore, the CH that begins the diffusion sets the *tid* to a quantity within the interval [T_0-ε,T_0+ε];

The two lines along the *Y* axis now represent the case of an extremely early *tid*, set when the origin was ε time units early, and that of an extremely late *tid* (see Figure 6). The timeout to the diffusion phase is set to *tid*+Δ_b. But, since the time is read with a maximum error of ε, the CHs will begin detecting algorithm faults from within the time interval [T_0+Δ_b-2ε,T_0+Δ_b+2ε].

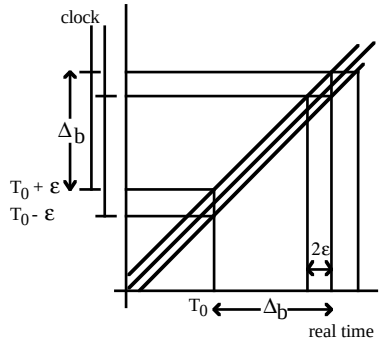


Fig. 7. Uncertainty about transition to detection phase.

Therefore there is a time interval in which the behavior of CHs may differ because of their clock values. For example, at real time $T_0+\Delta_b$ a late CH may have a clock value of $T_0+\Delta_b-\varepsilon$, and an early CH may have a clock value of $T_0+\Delta_b+\varepsilon$ and both may be non-informed. If they receive a message containing the information, the former will simply move to the informed state, while the latter will disconnect itself since it will be in the failure detection phase.

Although different, both behaviors are admissible. The disconnection of an acceptable early CH is undesirable, but does not disrupt the overall consistency of the algorithm: the fact that a node gets disconnected during the execution of the algorithm does not affect its correctness. The evaluation of the failure probability, on the other hand, must take into account the possibility of early nodes. A pessimistic evaluation of the probability of failure is:

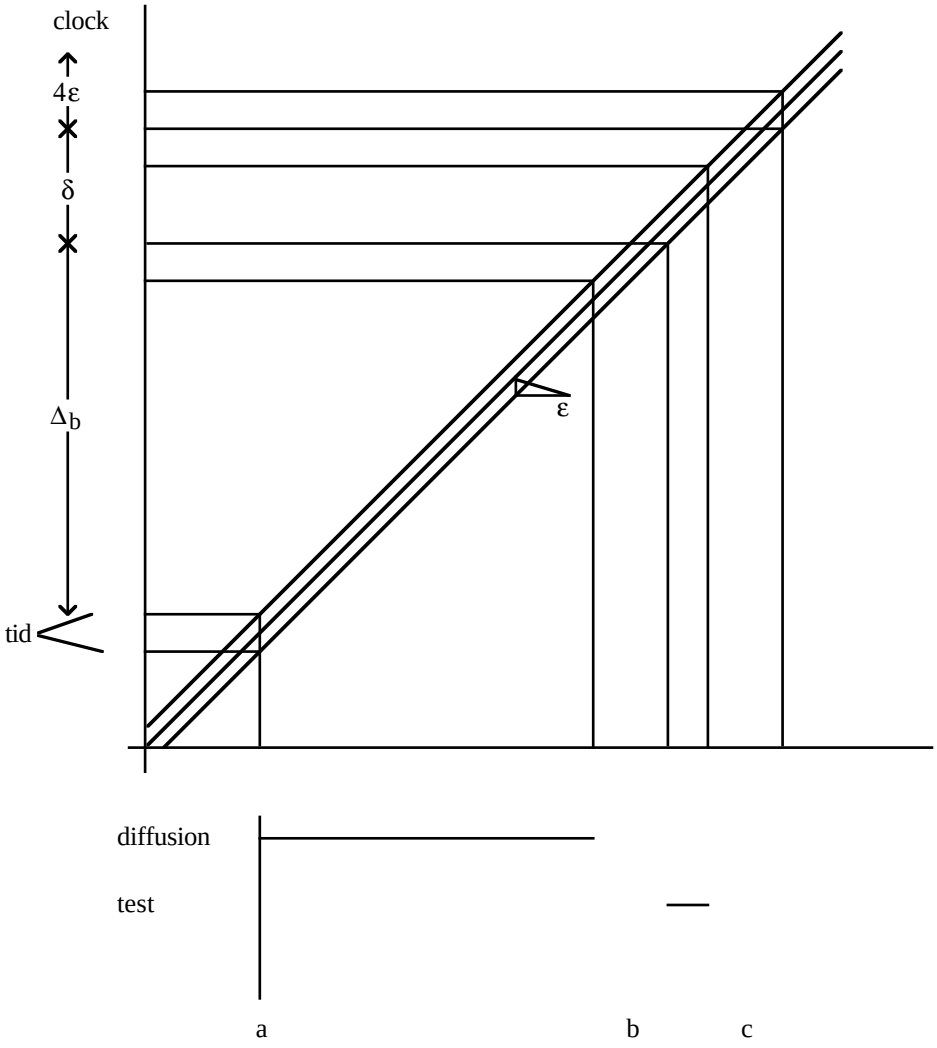
$1 - P(T_{sat} \leq \Delta_b - 2\varepsilon)$

The timing difference from early to late nodes can induce a more subtle inconvenience. Let us consider the case of two neighbor CHs, p_{early} and p_{late} ε time units early and late respectively, participating in a diffusion that began at time T_0, with $tid=T_0$. The first one definitely leaves the diffusion at time $T_0-\varepsilon$, delivers the diffused message to the local destination and terminates, by removing the rumor data structure (see program). If p_{late} sends a message to p_{early} before time $T_0+\varepsilon$, p_{early} detects a failure (since it has removed the rumor from its data structure, and therefore looks as if it is non informed while in fact it is) and disconnects. Again, the behavior does not disrupt the consistency but is nevertheless undesirable (see Figure 7)

It is easy to avoid this undesirable behavior, by delaying the removal of the rumor data structure from the local memory. The algorithm terminates when the node reads time $tid+\Delta_b+\Delta_{detect}$, but the data structure containing the rumor is erased 4ε time units later.

In conclusion, if we assume that clocks are synchronized with a maximum error of ε, it follows that WA is reached by time $T_0+\Delta_b+\Delta_{detect}+2\varepsilon$, and the probability that every CH connected to an informed CH is informed is $P(T_{sat}\leq\Delta_b-2\varepsilon)$. In addition, the rumor data structure is deleted 4ε time units after the termination of the diffusion in order to manage late messages.

Figure 8 summarizes the timing of the broadcast algorithm with inaccurate synchronization: the shaded areas in the phase transition diagram indicate the time uncertainty of the real transition times of a node.


clock
4ε
δ
Δ_b
tid
ε
diffusion
test
a
b
c

we have to wait an additional τ_{sched} time units before removing the rumor in the termination procedure.

4.3. Multiple broadcast

Even if we assume that broadcasts are not frequent operations, allowing just one broadcast at a time can be too strict a requirement. Moreover, it would be hard to enforce, since "concurrent" broadcasts start independently.

It is not difficult to extend the previous algorithm in order to support concurrent broadcasts: one only needs to extend the rumor data structure to represent a set of broadcast information, each one with its own *tid*, which can be used as a key to identify the broadcast. To solve ambiguities that may arise if two broadcasts are originated with identical *tid*s from different sources, the broadcast record should also contain the name of the CH that originated it [4]. Furthermore, the receive procedure should be modified accordingly. To speed up its operation, the rumor data structure could contain the broadcasts ordered by *tid* and *origin*, provided that a total ordering is defined on the CH.

To ensure that each CH delivers the broadcast messages in the same order, we must be sure that a total ordering is defined over the set of all tuples (*tid*, *origin*), and that a broadcast with a certain identifier is not delivered to the recipient until the CH is sure that none of the broadcasts received later precedes the delivered one.

The first condition is easily satisfied by introducing an (arbitrary) ordering on the CHs. The criteria to determine when the second condition is satisfied is based on the timing of the background diffusion algorithm: in fact, at time T every CH that is not excluded from the system and that will not be excluded from the system in the next Δ_{detect} seconds has received all broadcasts sent with $tid<T-\Delta_b-\varepsilon$. Since clock readings are affected by an error of $\pm\varepsilon$, a CH reading time T may conclude that either it has received every broadcast with $tid<T-\Delta_b-2\varepsilon$ or that it is going to be excluded by the system in the next Δ_{detect} time units. In other words, the message may be delivered to the recipient after time $T=tid+\Delta_b+2\varepsilon$, since after that time we are sure that if another broadcast were received with a lower *tid*, this would disconnect the CH and the node. If more than one broadcast is originated with the same *tid*, the one originated by the CH that precedes the others has to be delivered first. A similar criteria to ensure the consistent ordering of broadcast delivery may be found in [4].

However, to preserve WA, messages are delivered to the recipient at time $tid+\Delta_b+2\varepsilon+\Delta_{detect}$, past the deadline that ensures no preceding broadcasts may be received. Therefore, the modification required to enforce the order requirement merely consists of managing a more complicated data structure to keep several diffused messages at the same time, and in a termination procedure that delivers all the messages with the same *tid* in the correct order.

5. Conclusions

Designing distributed algorithms using the probabilistic paradigm can improve system performance without degrading its reliability. But designing a probabilistic algorithm may be very difficult, since the creation of a probabilistic model of its behavior is part of the design. When the model is particularly simple, as in [3] or in the case outlined in this paper, the system design should take into account the possibility of including probabilistic algorithms.

Acknowledgements

Dr. Domenico Luminati (Dept. of Mathematics - Univ. of Pisa) significantly contributed to proving Statement A, and Prof. M. Pratelli (Dept. of Mathematics - Univ. of Pisa) gave some useful indications on how to prove Statement B. I received some very useful advice from Prof. P. Grigolini (Dept. of Physics - Univ. of Pisa) that convinced me of the validity of my work before the paper was written. The anonymous referees provided me with insights and suggestions on earlier drafts.

6. Bibliography

[1] Birrell A.D.; Nelson B.J. Implementing remote procedure calls. ACM Transactions on Computer Systems; February 1984; 2(1): 39-59.

[2] Ciuffoletti A. Modeling Stochastic Diffusion in a Mesh Network: Dipartimento di Informatica. March 1992; TR 5/92.

[3] Cristian F. Probabilistic clock synchronization. Distributed Computing; 1989; (3): 146-158.

[4] Cristian F.; Aghili H.; Strong R.; Dolev D. Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. In proc.: International Symposium on Fault-Tolerant Computing;June 1985; 1985. 200-206.

[5] Demers A.; Greene D.; Hauser C.; Irish W.; Larson J.; Shenker S.; Sturgis H.; Swinehart D.; Terry D. Epidemic Algorithm for replicated database maintenance. In Proc.: ACM Symposium on Principles of Distributed Computing; August 1987; Vancouver (CANADA); 1987. 1-12.

[6] Dolev D.; Dwork C.; Stockmeyer L. On the minimal synchronism needed for distributed consensus. Journal of ACM; January 1987; 34(1): 77-97.

[7] Fischer M.J.; Lynch N.A.; and Paterson M.S. Impossibility of distributed consensus with one faulty process. Journal of ACM; April 1985; 32(2): 374-382.

[8] Kohler W.H. A survey of techniques for synchronization and recovery in decentralized computer systems. Computing Surveys; June 1981; 13(2): 149-183.

[9] Kopetz H.; Ochsenreiter W. Clock Synchronization in Distributed Real-Time Systems. IEEE Transactions on Computers; August 1987; 7(3): 404-425.

[10] Lamport L.; Melliar-Smith P.M. Synchronizing clocks in the presence of faults. Journal of ACM; January 1985; 32(1): 52-78.

[11] Massey W.S. Algebraic Topology: An Introduction. pp. 29-43. Eds: Halmos P.R., Gehring F.W., Moore C.C.. Graduate Texts in Mathematics. 56. Springer-Verlag. 1967.

[12] Melliar-Smith P.M.; Moser L.E.; Agrawala V. Broadcast protocols for distributed systems. IEEE Transactions on Parallel and Distributed Systems; January 1990; 1(1): 17-25.

[13] Schneider F. B. A Paradigm for Reliable Clock Synchronization: Cornell University Ithaca N.Y.; February 1986; TR 86-735.

Appendix

Pascal description of the broadcast algorithm.

```
program broadcast;
type
   time = longint;
   message = string;
   CHName = integer;
   ProcedureCall = string;
         {procedure name and parameters}
   AState = (connected, disconnected, ready);
{*** background broadcast data structure}
   DiffusedMessage = record
      tid: time;
      information: message;
      end;
{*** internode message}
   packet = record
      sender: CHName;
      content: message;
      tail: DiffusedMessage;
      end;
const
   Δb = 31;
   Δdetect = 28;
   EmptyMessage = '';
var
   rumor: DiffusedMessage;
   state: AState;

function MyClock: longint;
   {reads the clock}
   begin
   …
   end;

function MyName: CHName;
   {returns the name of the CH}
   begin
   …
   end;

function IsANeighbor (N: CHName): boolean;
   {verifies whether N is one of the neighbors or not}
   begin
   …
   end;

procedure TurnOffLinkTo (P: CHName);
   {disconnects the link with P}
   begin
   …
   end;

procedure Reconnect;
```

```
    {Sends a reconnection request along every}
    {turned off link and processes the answers}
    {Broadcasts the failure detection through}
    {connected links}
    begin
    …
    end;

procedure Deliver (M: message);
    {Delivers the message to the destination process,}
    {whose name is included in the message}
    begin
    …
    end;

procedure GetRumor (Pk: packet; var switching: boolean);
    {assumes that when executed, rumor is empty }
    {or contains the same rumor in Pk}
    begin
        switching := (Rumor.information = EmptyMessage) and (Pk.tail.information <> EmptyMessage);
        if switching then
            Rumor := Pk.tail;
    end;

procedure Schedule (C: ProcedureCall; T: time);
    {see paper}
    begin
    …
    end;

 procedure Cancel (C: ProcedureCall);
    {see paper}
    begin
    …
    end;

 procedure Call (C: ProcedureCall; Destination: CHName);
    {see paper}
    begin
    …
    end;

procedure Send (M: message; P: CHName);
    var
        Pk: packet;
    begin
        if IsANeighbor(P) then
            begin
                cancel('DummyMessageTo ( P )');
                schedule('DummyMessageTo ( P )', MyClock + Δdetect);
                Pk.sender := MyName;
                Pk.content := M;
                if state = connected then
                    Pk.tail := rumor;
                call('receive ( Pk )', P);
            end;
    end;

procedure Receive (Pk: packet);
```

```
    var
        AnotherRumor: DiffusedMessage;
        switching: boolean;
        P: CHName;
    begin
        P := Pk.sender;
        if IsANeighbor(P) then
            begin
                cancel('Lost ( P )');
                schedule('Lost ( P )', MyClock + Δdetect);
                deliver(Pk.content);
                GetRumor(Pk, switching);
                schedule('Terminate', Pk.tail.tid + Δb);
                if   (switching or (state = disconnected)) and
                    (MyClock < Pk.tail.tid + Δb + Δdetect) and
                    (MyClock > Pk.tail.tid + Δb) then
                    begin
                        TurnOffLinkTo(P);
                        state := disconnected;
                    end;
            end;
    end;

procedure BackgroundDiffuse (R: message);
    begin
        with rumor do
            begin
                tid := MyClock;
                information := R;
                schedule('Terminate', tid + Δb);
            end;
    end;

procedure Terminate;
    begin
        if state = connected then
            begin
                deliver(rumor.information);
                rumor.information := EmptyMessage;
            end
        else
            schedule('Reconnect', MyClock + Δdetect);
    end;

procedure DummyMessageTo (R: CHName);
    var
        M: message;
        P: CHName;
    begin
        if IsANeighbor(P) then
            begin
                cancel('Failed(R)');
                M := EmptyMessage;
                send(M, R);
            end;
    end;
```

Proof of Statement A

The first step in the proof consists of characterising the arrangement of the informed CHs, and is therefore concerned with the topology of the part of the system containing the informed CHs. A concise introduction to the notation and concepts used below are in [11]. We establish a homeomorphism between the CHs and a set of squares covering a torus Σ that preserves adjacency. The CHs that have received the broadcast are represented by a 2-manifold σ, with $\sigma \subseteq \Sigma$. Since the broadcast is only diffused between adjacent units, σ is connected. For the sake of brevity, we introduce the following terminology:

a trivial boundary is the boundary of a disc $D \subset \Sigma$;

a hole is a disc $D \subseteq (\Sigma - \sigma)$;

a cylinder is a connected surface with two non-trivial boundaries.

We prove the following lemma.

Lemma 1:

Let Σ be a torus and $\sigma \subseteq \Sigma$ a connected surface; then one of the following conditions holds:

a) σ is a disc with $k \geq 0$ holes or

b) σ is a cylinder with $k \geq 0$ holes or

c) σ is a torus with $k \geq 0$ holes.

Proof:

Let σ have $b \geq 0$ boundaries. If all boundaries are trivial we may distinguish two cases: either each of them delimits a hole, and then we are in case c) with $k=b$, or some of them delimit discs that are not holes. In that case the surface σ is contained in the union of several discs: since σ is connected, it is contained in only one disc, and we are in case a) with $k=b-1$.

If there are non-trivial boundaries, they are at least two, since one non-trivial boundary does not disconnect the torus. Let us cut the torus along two of them: the surface S that we obtain has four boundaries (since every cut generates two boundaries in a orientable surface) and the Euler characteristic is $\chi(S)=\chi(\Sigma)=0$. A connected surface with four boundaries has a $\chi \leq -2$, and therefore S is not connected. Since each cut increments by at most one the number of disconnected components, then S is composed by two components. We distinguish four possible distribution of the four boundaries. If all four boundaries delimit the same component, then σ is disconnected and this is excluded. If three boundaries delimit the same component, the other component is a connected surface with one boundary, i.e. a disc, and the corresponding boundary is trivial; but this is excluded since we assume that all considered boundaries are non trivial. Therefore both surfaces are delimited by two non trivial boundaries, and they are two cylinders. Since σ is connected, then it is entirely contained in one of the cylinders, that we call C.

A further non trivial boundary would cut C into a surface with four boundaries and $\chi=\chi(C)=0$: again this necessarily would consist into two cylinders of which one contains σ: but this is impossible, since in that case one of the boundaries used in the previous step would delimit two regions contained in $N-\sigma$, which contradicts the assumption that it was a boundary of σ. Therefore there are exactly two non-trivial boundaries that delimit a cylinder, and the other boundaries necessarily delimit discs in $N-\sigma$: we are in case b) with $k=b-2$.

We now give the lower bounds to the number of links connecting an informed CH to a non-informed one in each of the three cases.

Lemma 2 a).

Let s_k be a set of $\#s_k$ CHs that are arranged in a disc with k≥0 holes.

Then $p_k \geq 4\ Floor[\sqrt{\#s_k}]$.

Proof:

Let *R* be an array of CHs that exactly contains s_k, i.e.:

$R=\{(x,y) \mid \exists\ (x,y_k)\in\ s_k$ and $\exists\ (x_k,y)\in\ s_k\}$.

Let L and H be the number of columns and rows of R: therefore

$\#s_k \leq LxH$ (1)

Each row and column contains at least two distinct links between an informed and a non informed unit (i.e., each line containing a point of a disc crosses the boundary in at least two points) and therefore:

$p_k \geq 2(L+H)$. (2)

By algebraic manipulation, we obtain from (1) and (2):

$$p_k \geq 2\left(\frac{\#s_k}{H}+H\right)$$

The expression on the right hand side has a minimal value (obtained by derivation) when $H=\sqrt{\#s_k}$: therefore:

$$p_k \geq 4\sqrt{\#s_k} \geq 4\ \text{Floor}\left[\sqrt{\#s_k}\right]$$

Lemma 2 b).

Let s_k be a set of $\#s_k$ CHs that are arranged in a cylinder with k≥0 holes.

Then $p_k \geq 2\sqrt{N_c}$.

Proof:

The two components of the boundary of the cylinder are circles which are either parallel to the *x* or *y* axis: therefore each of the $\sqrt{N_c}$ columns or rows crosses the boundary in at least two CHs, and the lemma follows.

Lemma 2 c).

Let s_k be a set of $\#s_k$ CHs that are arranged in a torus with k≥0 holes .

Then $p_k \geq 4\ Floor[\sqrt{N_c-\#s_k}]$.

Proof:

If we consider separately each of the *k* holes C_i, $1\leq i\leq k$, with a construction similar to lemma 2 a):

$$p_k \geq \sum_{i=1}^{k} 4\sqrt{\#C_i}\ \textit{with}\ \sum_{i=1}^{k}\#C_i=(N_c-\#s_k)$$

Since for every *a,b*, $\sqrt{a}+\sqrt{b} \geq \sqrt{a+b}$, we conclude (by induction on *k*)

$$p_k \geq 4\sqrt{N_c-\#s_k}\ \geq 4\ Floor\left[\sqrt{N_c-\#s_k}\right]$$

Lemma 3

In the assumption and the notation adopted in sect. 3.2:

$$p_k \geq Min\left[4\ Floor[\sqrt{\#s_k}]\ ,2\sqrt{N_c},\ 4\ Floor[\sqrt{N_c-\#s_k}]\right]$$

Proof: from lemmas 1, 2a, 2b, 2c.

We can now set

$\pi_{\#s_k} = Min\left[4\ Floor[\sqrt{\#s_k}]\ ,2\sqrt{N_c}\ ,4\ Floor[\sqrt{N_c - \#s_k}]\right]$

and prove that:

if $\#s_k < \frac{N_c}{4}$ *then* $4\ Floor[\sqrt{\#s_k}] < 2\sqrt{N_c}$ *and* $4\ Floor[\sqrt{\#s_k}] < 4\ Floor[\sqrt{N_c - \#s_k}]$

if $\frac{N_c}{4} < \#s_k < 3\frac{N_c}{4}$ *then* $2\sqrt{N_c} < 4\ Floor[\sqrt{\#s_k}]$ *and* $2\sqrt{N_c} < 4\ Floor[\sqrt{N_c - \#s_k}]$

if $\#s_k > 3\frac{N_c}{4}$ *then* $4\ Floor[\sqrt{N_c - \#s_k}] < 2\sqrt{N_c}$ *and* $4\ Floor[\sqrt{N_c - \#s_k}] < 4\ Floor[\sqrt{\#s_k}]$

to conclude that Statement A holds. The proofs of the above formulas are omitted.

Proof of statement B

The Markov chain ***Apx*** can be represented as a sequence of Bernoulli trials: each trial corresponds to a message exchange and consists of selecting one of the two transitions possible at each state: remain in the same state (failure, or absence of diffusion) or move to the next state (success, or diffusion). Each time a success occurs, the probability of a further success changes slightly (see (4)). The diffusion terminates after N_c-1 successful transitions. This interpretation facilitates the analytic solution of the ***Apx*** model: the number of message exchanges needed to terminate the diffusion is a sum of random variables $\boldsymbol{x_i}$ with geometric distribution:

$$P_{1\,N_c}^{\boldsymbol{Apx}}(1,k) = P\left\{\sum_{i=1}^{N_c-1} \boldsymbol{x_i} \le k\right\} \quad \textit{with } \boldsymbol{x_i} \sim G\left(\frac{\pi_i}{4N_c}\right) \quad (5)$$

From the assumptions about the system, we know that each trial has an associated random duration, which is expressed by the random variable $\boldsymbol{\Delta t}/N_c$. We can therefore derive from the previous expression the distribution of a random variable $\boldsymbol{Y}$ that represents the time interval before N_c successful transitions occur in the ***Apx*** model. From the construction of ***Apx*** we can derive:

$$P\{\boldsymbol{Y} \le \Delta_b) \le P(\boldsymbol{T_{sat}} \le \Delta_b) \qquad (6)$$

and, adding the period between each of the N_c - 1 trials:

$$\boldsymbol{Y} = \sum_{i=1}^{N_c-1}\left(\sum_{j=1}^{x_i}\frac{\boldsymbol{\Delta t}}{N_c}\right) = \frac{1}{N_c}\sum_{i=1}^{N_c-1}\left(\sum_{j=1}^{\boldsymbol{x_i}}\boldsymbol{\Delta t}\right) \qquad (7)$$

It is possible to show that each additive term has an exponential distribution, with a mean value equal to the product of the mean values of $\boldsymbol{x_i}$ and $\boldsymbol{\Delta t}$ (the complete proof can be found in [2]):

$$\sum_{j=1}^{\boldsymbol{x_i}}\boldsymbol{\Delta t} \sim E\left(\frac{\pi_i}{4N_c}\right) \qquad (8)$$

Since the exponential distribution is the continuous extension of the geometric distribution, from (5) and (8) we derive

$$P\left\{\sum_{j=1}^{\boldsymbol{x_i}}\boldsymbol{\Delta t} \le t\right\} = P\{\boldsymbol{x_i} \le t\} \textit{ when t is an integer}$$

and therefore, from (7)

$$P(\boldsymbol{Y} \le t) = P\left\{ \frac{1}{N_c} \sum_{i=1}^{N_c-1} \boldsymbol{x_i} \le t \right\} = P\left\{ \sum_{i=1}^{N_c-1} \boldsymbol{x_i} \le t\ N_c \right\} \text{ when } t \text{ is an integer}$$

and, from (5)

$$P^{\boldsymbol{Apx}}_{1\ N_c}(1,\ t\ N_c) = P\left\{ \sum_{i=1}^{N_c-1} \boldsymbol{x_i} \le t\ N_c \right\} = P(\boldsymbol{Y} \le t) \text{ when } t \text{ is an integer}$$

Finally, from (6) we conclude

$$P^{\boldsymbol{Apx}}_{1\ N_c}(1,\ \Delta_b\ N_c\ ) = P(\boldsymbol{Y} \le \Delta_b\ ) \le P(\boldsymbol{T_{sat}} \le \Delta_b) \text{ when } \Delta_b \text{ is an integer} \qquad \textbf{QED}$$

The proof justifies our use of a pessimistic estimate for the time needed to inform every CH. In order to quantify the error introduced by this estimate, we simulated the behavior of the broadcast algorithm. The system state is represented by an 8x8 array of cells. Each cell can be in one of two states (informed, non-informed). In the initial state, every cell is non-informed, except one. A sequence of random steps is then generated. At each step, one CH is chosen at random together with one of its neighbors. A message exchange between the two is then performed: as a consequence, if the former is informed, the latter becomes (or remains) informed too, otherwise the state of the two remains unchanged. Figure 9 compares the distribution of the sample obtained from 10000 runs of the algorithm on the simulator and the distribution obtained from the pessimistic model. White boxes represent the frequency (indicated on the *y* axis) with which the simulated broadcast requires more than Δ_b time units (indicated on the *x* axis). Black boxes indicate the probability (on the *y* axis) of the same event computed using the approximation given by statement B. The pessimistic model is shown to give an acceptable approximation of the real behavior.

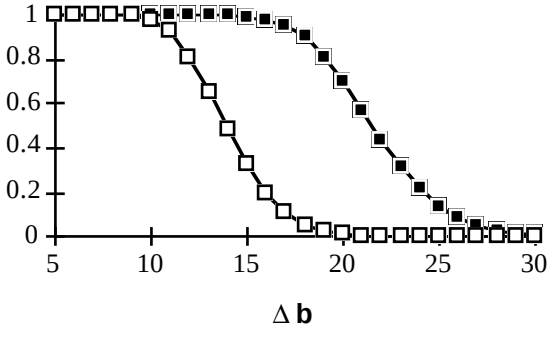


Fig. 9. Comparison between pessimistic estimates and experiment

Biographical notes:

Augusto Ciuffoletti graduated in computer sciences at the University of Pisa in 1980. From 1980 to 1983 he worked with Selenia and OtoMelara on projects funded by the Italian National Research Council. Since 1984 he has been doing research at the University of Pisa, where he is now an assistant. He is particularly interested in studying efficient and reliable low level services (such as broadcast).