Consensus Based on Failure Detectors with a Perpetual Accuracy Property

Achour MOSTEFAOUI and Michel RAYNAL
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
Tel: (+33) 2 99 84 71 88 Fax: (+33) 2 99 84 25 33
{mostefaoui,raynal}@irisa.fr

Abstract

This paper is on the Consensus problem, in the context of asynchronous distributed systems made of n processes, at most f of them may crash. A family of failure detector classes satisfying a Perpetual Accuracy property is first defined. This family includes the failure detector class $\mathcal{S}$ (the class of Strong failure detectors defined by Chandra and Toueg) central to the definition of a class ($\mathcal{S}_x$) where x is the minimum number ($x \geq 1$) of correct processes that can never be suspected to have crashed.

Then, a protocol that solves the Consensus problem is given. This protocol works with any failure detector class ($\mathcal{S}_x$) of this family. It is particularly simple and uses a Reliable Broadcast protocol as a skeleton. It requires $n - x + 1$ communication steps, and its communication bit complexity is $(n - x + 1)(n - 1)|v|$ (where $|v|$ is the maximal size of an initial value a process can propose).

Keywords: Asynchronous Distributed System, Consensus, Crash Failure, Perpetual Accuracy Property, Reliable Broadcast, Unreliable Failure Detector.

1 Introduction

The *Consensus* problem is now recognized as a fundamental problem when one has to design or implement reliable asynchronous distributed systems in presence of process crashes. Informally, the Consensus problem can be defined in the following way: each process proposes a value and all non-crashed processes have to agree on a common value, which has to be one of the proposed values. It has been shown that practical agreement problems can be reduced to the Consensus problem. As an example, let us consider the *Atomic Broadcast* problem: all processes have to agree on the same message delivery order. This is a typical agreement problem that can be solved by reducing it to the Consensus problem [1].

Unfortunately, solving the Consensus problem in an asynchronous distributed system where processes may crash is not a trivial task. It has been proved by Fischer, Lynch and Paterson that the Consensus problem has no deterministic solution in those systems as soon as processes (even only one) may crash [2]. The intuition that underlies this impossibility result lies in the inherent difficulty of safely distinguishing a process that has crashed from a process that is "very slow", or from a process with which communications are "very slow". To circumvent this impossibility result, in a seminal work [1], Chandra and Toueg have introduced the *Unreliable Failure Detector* concept, and studied how unreliable failure detectors can be used to solve the Consensus problem in asynchronous distributed systems with process crash failures.

A failure detector can be seen as an oracle that provides each process with a list of processes it suspects to have crashed. A failure detector can make mistakes by not suspecting a crashed process or by erroneously suspecting a non-crashed process. Chandra and Toueg have studied several classes of unreliable failure detectors. A class is defined by a *Completeness* property and an *Accuracy* property. The Completeness property is on the actual detection of crashes. The aim of an Accuracy property is to restrict the mistakes a failure detector module can make. Furthermore, an Accuracy property can be *Eventual* or *Perpetual*. An Accuracy property is *Eventual* when it is allowed to be satisfied only after some time by the failure detector. It is *Perpetual* when it has to be satisfied from the beginning by the failure detector [1].

In this paper, we are interested in solving the Consensus problem in asynchronous distributed systems prone to process crashes augmented with unreliable failure detectors satisfying a Perpetual Accuracy property. More precisely, we consider a family of unreliable failure detector classes whose Perpetual Accuracy property is parameterized by the minimum number (x) of correct processes that can not be suspected to have crashed. (The failure detector class denoted $\mathcal{S}$ [1] belongs to this family. It corresponds to the case where all but one process may be suspected). The proposed Consensus protocol is particularly simple, namely, it

uses a *Reliable Broadcast* protocol as a skeleton. Due to its intrinsic simplicity, the protocol reveals itself to be efficient. It requires $(n - x + 1)$ communication steps and costs $(n - x + 1)(n - 1)$ messages (where n is the number of processes), each message carrying a single value (namely, a value proposed by a process, the size of this value being $|v|$).

The paper is composed of 6 sections. Section 2 presents the asynchronous distributed system model augmented with unreliable failure detectors providing a Perpetual Accuracy property. Section 3 defines the Consensus problem. Then, Section 4 presents the Consensus protocol: its underlying principles (Section 4.1) and its formal description (Section 4.2). Its proof constitutes Section 5. Finally, Section 6 provides a few concluding remarks.

2 System Model

The system model is patterned after the one described in [1, 2]. A formal introduction to failure detectors is provided in [1].

2.1 Asynchronous Distributed System

We consider a system consisting of a finite set of $n > 1$ processes, namely, $\{p_1, p_2, \ldots, p_n\}$. A process can fail by *crashing*, *i.e.*, by prematurely halting. It behaves correctly (*i.e.*, according to its specification) until it (possibly) crashes. By definition, a *correct* process is a process that does not crash. Let f denote the maximum number of processes that can crash. Processes communicate and synchronize by sending and receiving messages through channels. Channels are not required to be FIFO. They can duplicate messages, but they cannot loss, alter or create messages. So, a message sent by a process p_i to a process p_j is eventually received by p_j, if p_j is correct. It is the multiplicity of processes and the communication by message-passing that make the system *distributed*. There is no assumption about the relative speed of processes or the message transfer delays. This absence of timing assumptions makes the distributed system *asynchronous*.

Given a system of n processes, we assume that at least one of them is correct, so $f \leq n - 1$. In the following $\mathrm{S}(n, f)$ denotes any distributed system made of n processes, at most f of them may crash.

2.2 Unreliable Failure Detectors

Let us consider a system $\mathrm{S}(n, f)$. Informally, a failure detector consists of a set of modules, each one attached to a process: the module attached to p_i maintains a set (named $suspected_i$) of processes it currently suspects to have crashed. Any failure detector module is inherently unreliable: it can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct one. Moreover, suspicions are not necessarily stable: a process p_j can be added to and removed from a set $suspected_i$ according to whether p_i's failure detector module currently suspects p_j or not. As in [1], we say "process p_i suspects process p_j" at some time t, if at time t we have $p_j \in suspected_i$.

As indicated in the introduction, a failure detector class is defined by two abstract properties, namely a *Completeness* property and an *Accuracy* property. In this paper we consider the following Completeness property [1]:

- Completeness: Eventually, every crashed process is permanently suspected by every correct process.

Chandra and Toueg have defined four Accuracy properties [1]: two Perpetual Accuracy properties and two Eventual Accuracy properties. The Perpetual Accuracy properties are:

- Strong Accuracy: No process is suspected before it crashes.
- Weak Accuracy: Some correct process is never suspected.

Combined with the Completeness property, these Perpetual Accuracy properties define the following two classes of failure detectors [1]:

- $\mathcal{P}$: The class of *Perfect* failure detectors. This class contains all the failure detectors that satisfy the Completeness property and the Strong Accuracy property. A failure detector of this class never makes mistakes.
- $\mathcal{S}$: The class of *Strong* failure detectors. This class contains all the failure detectors that satisfy the Completeness property and the Weak Accuracy property. A failure detector of this class can make an arbitrary number of mistakes.

2.3 The Class $\mathcal{S}_x$

As far as failure detectors for a system $\mathrm{S}(n, f)$ are concerned, let us consider the following Perpetual Accuracy property, where $1 \leq x \leq n - f$:

- Bounded Accuracy: There are x correct processes that are never suspected [1].

We define $\mathcal{S}_x$ as the class of unreliable failure detectors for $\mathrm{S}(n, f)$ that satisfy the Completeness property and the Bounded Accuracy property parameterized with x. Note

[1] Of course there is no a priori knowledge on which are these x processes.

that a failure detector of a class $\mathcal{S}_x$ can make an arbitrary number of mistakes. Moreover, given n and f, we have the following inclusions:

$$\mathcal{P} \subseteq \mathcal{S}_{n-f} \subseteq \cdots \subseteq \mathcal{S}_2 \subseteq \mathcal{S}_1 = \mathcal{S}$$

The class $\mathcal{S}_{n-f}$ includes $\mathcal{P}$ because a failure detector of $\mathcal{S}_{n-f}$ may suspect a process before it crashes, while a failure detector of $\mathcal{P}$ can not.

3 The Consensus Problem

In the Consensus problem, every correct process p_i *proposes* a value v_i and all correct processes have to *decide* on some value v, in relation with the set of proposed values. More precisely, the *Consensus* problem is defined by the three following properties [1, 2]:

- Termination: Every correct process eventually decides on some value.
- Validity: If a process decides v, then v was proposed by some process.
- Agreement: No two correct processes decide differently.

The agreement property applies only to correct processes. So, it is possible that a process decides on a distinct value just before crashing. *Uniform Consensus* prevents such a possibility. It has the same Termination and Validity properties plus the following agreement property:

- Uniform Agreement: No two processes (correct or not) decide differently.

In the following we are interested in the *Uniform Consensus* problem.

4 The Protocol

This section defines a Consensus protocol for an asynchronous distributed system $S(n, f)$, augmented with a failure detector of a class $\mathcal{S}_x$ (with $1 \leq x \leq n - f$).

4.1 Underlying Principle

An instance of the *Uniform Reliable Broadcast* problem [5] allows a process p_i to send a message to all the processes, in such a way that (1) if p_i is correct the message is delivered to all correct processes, and (2) otherwise, if the message is delivered to any process, then it is delivered to all correct processes.

Actually, the Consensus problem can be seen as a "combination" of n instances of the Reliable Broadcast problem (one instance per process), such that a single message (taken from the set of messages that have been sent) is delivered to processes. This simple observation suggests to use a Reliable Broadcast protocol as a skeleton to build a Consensus protocol. By broadcasting its initial value, a given process (*e.g.*, p_1) tries to impose this value as the decision value. As this process may crash during the broadcast, the other processes have to forward its value when they receive it, in order to overcome this possible crash [5]. Moreover, as the process that is supposed to initiate the Reliable Broadcast (here, p_1) may crash before launching it, the Reliable Broadcast protocol is modified to allow a process that has not received a value, to participate in the Reliable Broadcast by "forwarding" its own initial value that then replaces the "not received" value. This constitutes the design principle that underlies the proposed protocol: it is a Uniform Reliable Broadcast protocol where the value initially sent can be changed during the execution, according to process crashes and (possibly erroneous) suspicions.

4.2 The Consensus Protocol

The Consensus protocol is described in Figure 1. A process p_i participates in a Consensus execution[2] by invoking the function Consensus(v_i). The local variable est_i represents the current local estimate of the decision value, it is initialized to v_i (line 1). The statement **return** (est_i) terminates the participation of p_i in the Consensus, and provides it with the decision value est_i (line 12). As defined in Section 2.2, $suspected_i$ is the set of processes currently suspected by p_i. From the definition of x (namely, $1 \leq x \leq n - f$), we conclude that any set of $n - x + 1$ processes contains at least one correct process that is never suspected.

The protocol works in the following way.

- It first tries to realize a Reliable Broadcast of the initial value of p_1. To this end, without previously waiting for values from other processes (lines 2-4), p_1 sends its initial value to all the processes (line 6).
- When p_2 receives the value of p_1, it adopts this value as current estimate (line 4) and participates in its Reliable Broadcast by forwarding it to all processes p_j with $j > 2$ (line 6). In that way, p_2 overcomes a possible crash of p_1 that would occur after it has sent v_1 to p_2 and before it has sent it to other processes. If p_2 (maybe erroneously) suspects p_1 before receiving its initial value v_1 (line 3), then p_2 participates in the Reliable Broadcast by forwarding its own initial value

[2] As we will see, the protocol uses process identities. In order to balance the load when there are several Consensus executions, each Consensus execution can have its own assignment of process identities.

```
Function Consensus(v_i)
(1)   est_i ← v_i;
(2)   for j from 1 to min(i − 1, n − x + 1) do
(3)       wait until ((p_j ∈ suspected_i) ∨ (est_from_j is received from p_j));
(4)       if (est_from_j has been received from p_j) then est_i ← est_from_j endif enddo;
(5)   if (i ≤ n − x + 1) then
(6)       forall j such that i + 1 ≤ j ≤ n do send(est_i) to p_j enddo;
Broadcast % (7)   forall j such that 1 ≤ j ≤ i − 1 do send(est_i) to p_j enddo;
Agreement %(8)    for j from i + 1 to n − x + 1 do
(9)           wait until ((p_j ∈ suspected_i) ∨ (est_from_j is received from p_j));
(10)          if (est_from_j has been received from p_j) then est_i ← est_from_j endif enddo;
(11)  endif;
(12)  return (est_i)
```

Figure 1. The Consensus Protocol

v_2 instead of v_1 (line 6). Preventing the protocol from blocking, this guarantees its termination.

- More generally, a process p_i sequentially considers all processes p_j such that $1 \leq j < i$. With respect to each of them, p_i behaves as p_2 behaved with respect to p_1 (lines 2-4). After all these waitings have been completed, p_i has in est_i the last value it has received or v_i if it has received no value (lines 3-4). Then it participates in the Reliable Broadcast by forwarding its current value of est_i to all the processes p_j with $j > i$ (line 6).

- Due to erroneous suspicions concerning p_j by other processes, the value forwarded (and currently adopted) by p_j is not necessarily the last value received by a correct process p_i ($i > j$). To ensure that p_j will decide on the same value as the other processes, a process p_i is required to forward its current estimate also to all processes p_j such that $j < i$ (line 7). Hence, in order to take into account the values sent to it at line 7 by the other processes, a process p_j executes lines 8-10: it sequentially waits values from processes p_i ($i > j$), and updates its current local estimate to the last of these values (if any) it receives.

- As indicated previously, due to the definition of x, any set of $n - x + 1$ processes contains at least one correct process that is never suspected. The protocol exploits this property in the following way: the subset[3] made of the processes $p_1, \ldots, p_{n-x+1}$ defines the decision value and imposes it to the subset of processes p_j such that $j > n - x + 1$. More precisely:

  - The set of processes p_i such that $1 \leq i \leq n - x + 1$ realizes the modified Uniform Reliable Broadcast as described previously. The upper bounds of the **for** loops of lines 2 and 8, and the test done at line 5 guarantee that all processes of this subset and only them participate "actively" in the modified Reliable Broadcast whose aim is to establish the decision value.

  - The set of processes p_j such that $n-x+1 < j \leq n$ is made of "passive" processes in the following sense. A process p_j of this set does not participate in the forwarding of values required by the Reliable Broadcast (test at line 5). Its work is limited to the reception of forwarded values and to the corresponding updates of its local estimate est_j (lines 2-4).

  To summarize, any process p_i waits for a value (or a suspicion) from exactly $n-x+1$ processes. According to the value of i, some of these waitings are done at lines 2-3 while the others are done at lines 8-9.

[3] This subset is only one of the possible subsets satisfying the property. Its advantage lies in its easy management by the protocol.

5 Proof

5.1 Validity

Theorem 1 *If a process p_i decides v, then v was proposed by some process.*

Proof The only line at which **return** is executed is line 12, the returned value being an estimate value (est_i). For any process p_i, initially $est_i = v_i$ (line 1), and then, est_i can be modified at line 4 or at line 10. Due to the reliable channel assumption (no alteration, no spurious messages), in both cases the new value of est_i is the value of another estimate sent at line 6 or at line 7. Consequently, est_i always contains a value proposed by some process. $\square_{Theorem\ 1}$

5.2 Termination

Lemma 1 *No 2-4)* (P1). *Moreover, every correct process p_j such that $j \leq n - x + 1$ broadcasts an estimate value* (P2).

Proof The proof is by induction on the increasing sequence of processes identities. Note that only non-crashed processes can block in a loop.

• Base case. Let us first consider the case $i = 1$, *i.e.*, p_1. As $i - 1 = 0$, it follows from line 2, that p_1 has not to execute the first loop, so it can not remain blocked forever in this loop. Consequently, if p_1 is correct, it eventually executes lines 6-7 and broadcasts v_1.

• Induction case. Considering $i > 1$, let us assume (induction hypothesis):
- P1($i-1$): $\forall\ p_k$, such that $1 \leq k \leq i-1$, p_k does not remain blocked forever in the first loop (either it exits from the loop or it crashes), and
- P2($i-1$): $\forall\ p_k$, such that $1 \leq k \leq i-1$, if p_k is correct and $k \leq n - x + 1$, it broadcasts an estimate value.

We first show that P1(i) is satisfied (namely, p_i can not remain blocked forever within the first loop). If p_i crashes, it is no longer blocked in the first loop. If p_i is correct, then for any p_k such that $k \leq min(i-1, n-x+1)$ we have:
- If p_k is correct, then, due to the second part of the induction hypothesis, namely P2$(i-1)$, p_i eventually receives an estimate from p_k (note that due to an erroneous suspicion, p_i can also suspect p_k before receiving this estimate).
- If p_k crashes, then due to the Completeness property of the failure detector, p_i eventually suspects it.

P2(i) is trivially obtained from P1(i). If p_i is correct and if $i \leq n - x + 1$, as it exits from the first loop, it eventually executes lines 6-7, and consequently broadcasts an estimate value. $\square_{Lemma\ 1}$

Lemma 2 *No correct process remains blocked forever in the second loop (lines 8-10).*

Proof Due to Property P1 of Lemma 1 (at least) all correct processes enter the second loop. Moreover, due to line 5, the lemma is trivially true for all correct processes p_i such that $i > n - x + 1$. So, we consider in the following a correct process p_i such that $i \leq n - x + 1$. Line 8 indicates that p_i waits for an estimate value or a suspicion from $p_{i+1}, \cdots, p_{n-x+1}$. Let p_j be any of these processes. There are two cases:
- If p_j crashes, then due to the Completeness property of the failure detector, p_i will suspect it.
- If p_j is correct, then, due to property P2 of Lemma 1, p_j eventually broadcasts an estimate value.

So, for any j such that $i + 1 \leq j \leq n - x + 1$, p_i eventually receives an estimate from p_j or suspects p_j.
It follows that no correct process blocks in the second loop. $\square_{Lemma\ 2}$

Theorem 2 *Every correct process eventually decides on some value.*

Proof The proof follows directly from the property P1 of Lemma 1, Lemma 2 and line 12. $\square_{Theorem\ 2}$

5.3 Uniform Agreement

Let p_k be a correct process that is never suspected and such that $1 \leq k \leq n - x + 1$. Due to to the definition of x, such a process does exist. Moreover, due to the property P2 of Lemma 1, p_k broadcasts an estimate value. Let v be this value.

Lemma 3 $\forall j$ *such that $k \leq j \leq n$, if p_j broadcasts a value at lines 6-7, then this value is equal to v. (Note that if p_j crashes during this broadcast, this value may be sent only to a subset of the processes).*

Proof The proof is by induction on the sequence (starting at k) of process identities.

• Base case: $j = k$. This case is trivially satisfied due to the definition of v.

• Induction case. Considering $j > k$, let us assume (induction hypothesis) that $\forall l :\ k \leq l \leq j-1$, if p_l broadcasts a value then this value is equal to v. Note that we have $k \leq min(j-1, n-x+1)$.

If p_j crashes before line 6, then the property is trivially satisfied. So, we consider that p_j starts executing lines 6-7. In the first loop (lines 2-4), p_j waited for an estimate value from (or a suspicion of) $p_1, p_2, \ldots, p_{min(j-1,n-x+1)}$. As, due to Lemma 1, p_j exits from the first loop, for each process p_h (with $1 \leq h \leq min(j-1, n-x+1)$), either p_j has received a value from p_h or has suspected p_h.

As the waiting is done sequentially (first a message from/suspicion of p_1, etc.), as $k \leq min(j-1, n-x+1)$ and as p_k is not suspected, p_j receives the value v from p_k and updates accordingly its local estimate est_j. Then, due to the induction hypothesis, if p_j receives an estimate value from a process p_l ($k < l \leq j-1$), this estimate value is equal to v. So, est_j remains equal to v. It follows that if p_j broadcasts a value at lines 6-7, this value is equal to v. $\square_{Lemma\ 3}$

Theorem 3 *No two processes decide differently.*

Proof We show that the decision value is v. Let p_i be a (correct or not) process that decides. Before deciding, for any p_j such that $1 \leq j \leq n-x+1$, p_i has either received a value from p_j or suspected p_j. If $i \geq n-x+1$ these values (or suspicions) have been received during the first loop (lines 2-4). If $i < n-x+1$, some of them have been received during the first loop, others during the second loop (lines 8-10). Considering the values p_i has received before deciding (at line 12), we have:
- First, p_i has necessarily received the value v from p_k. This is due to Theorem 2 (p_k has terminated and has consequently executed lines 6-7), and to the fact that p_k is not suspected. Moreover, this value has overwritten the values p_i had previously received from any p_l such that $1 \leq l < k$. This is due to the sequentiality of the waitings that forces p_i to wait first a value from (or a suspicion of) p_1, then a value from (or a suspicion of) p_2, etc., until p_k.
- Then, due to Lemma 3, if p_i has received a value from a p_l such that $k < l \leq n-x+1$, this value is equal to v.
$\square_{Theorem\ 3}$

6 Concluding Remarks

Cost of the protocol Let us assume there is no crash. It is easy to deduce from the structure of the protocol that, in the worst case, a decision is obtained after $n-x+1$ communication steps (a single process broadcasts a value during a communication step: p_1 during the first step, p_2 during the second step, etc., until p_{n-x+1}).

Let $|v|$ be the maximal size (expressed in number of bits) of an initial value. At the j^{th} communication step ($1 \leq j \leq n-x+1$), p_j sends a value to each other process. So, the bit complexity of a step is $(n-1)|v|$. As there are $n-x+1$ communication steps, the message complexity is $(n-x+1)(n-1)$, and consequently, the comunication bit complexity of the protocol is $(n-x+1)(n-1)|v|$.

As we can see, the value of x has a direct effect on the efficiency of the protocol. This provides an interesting insight on the relation between the quality of service offered by a failure detector and the efficiency of the protocol that uses it.

Related work To our knowledge, the only Consensus protocol based on a failure detector of the class $\mathcal{S}$ (=$\mathcal{S}_1$) that has been designed so far is due to Chandra and Toueg [1]. This protocol requires n communication steps. Moreover, at each step, each process broadcasts an array of size n including the new values it has learnt during the previous step. So, the size of a message is $n|v|$ and at each step n^2 messages are sent. It follows that the communication bit complexity of this protocol is $n^3(n-1)|v|$. In the same context ($x = 1$), the proposed protocol requires the same number of communication steps, but has a lower communication bit complexity, namely, $n(n-1)|v|$.

Guerraoui has proposed a Consensus protocol that assumes a failure detector of the class $\mathcal{P}$ (the class of Perfect failure detectors) [3]. This protocol is based on a principle close to the one that underlies the proposed protocol. This protocol does not work with a failure detector of a class $\mathcal{S}_x$ ($x \geq 1$), and does not ensure Uniform Agreement (it ensures the Agreementonly among correct processes).

Early decision Let us consider the case $x = 1$. Whatever the value of f and the erroneous process crash suspiscions, the proposed protocol always requires n rounds. The interested reader will find in [6] a $\mathcal{S}$-based Consensus protocol that allows early decision. More specifically, if there are no erroneous suspiscions the protocol described in [6] terminates in $2(f+1)$ rounds in the worst case.

Consensus with eventual accuracy Finally, the reader interested in Consensus protocols based on failure detectors providing only eventual weak accuracy will consult [1, 4, 6, 7, 8].

References

[1] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, March 1996.

[2] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, April 1985.

[3] Guerraoui R., Revisiting the Relationship between Non-Blocking Atomic Commitment and Consensus. *Proc. 9th Int. Workshop on Distributed Algorithms (WDAG95)*, LNCS 972, pp. 87-100, September 1995.

[4] Hurfin M. and Raynal M., A Simple and Fast Asynchronous Consensus Protocol Based on a Weak Failure Detector. *Distributed Computing*, 12(4):209-223, 1999.

[5] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press (S. Mullender Ed.), New-York, pp. 97-145, 1993.

[6] Mostefaoui A. and Raynal M., Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a Generic Quorum-Based Approach. *Proc. 13th Int. Symposium on Distributed Computing (DISC'99), (Formerly WDAG)*, LNCS 1693, pp. 49-64, Bratislava, 1999.

[7] Mostefaoui A. and Raynal M., Unreliable Failure Detectors with Limited Scope Accuracy and an Application to Consensus. *Proc. 19th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'99)*, LNCS 1738, pp. 329-340, Chennai, December 1999.

[8] Schiper A., Early Consensus in an Asynchronous System with a Weak Failure Detector. *Distributed Computing*, 10:149-157, 1997.