Combining different Failure Detectors for Solving a Large-Scale Consensus Problem

S. Haddad
Lamsade
haddad@lamsade.dauphine.fr

Lamsade, Univ. Paris-Dauphine
Pl. Maréchal de Lattre de Tassigny
75775 Paris Cédex 16, France

F. Nguilla Kooh
Lamsade, Esigetel
nguilla@[lamsade.dauphine.fr,esigetel.fr]

Esigetel, Dépt. de la Recherche
1, rue port de Valvins
77215 Fontainebleau-Avon, France

Abstract

Dependable services in distributed systems rely on some kind of agreement. Such an agreement can be obtained by solving the consensus problem. Most of the proposed consensus' algorithms are based on mutual knowledge of the participants and thus inadequate to wide area networks (WANs). In previous papers, we proposed protocols which deal with WANs constituted of interconnected physical groups of machines (LANs/domains). These protocols rely on Chandra and Toueg' unreliable failure detectors model for asynchronous systems extended to handle broadcast addresses and safeness of a whole subnet. Nevertheless, we assumed the same characteristics for local and distant failure detectors. In the present paper, we propose a new algorithm which takes into account a different behavior for the detectors. More precisely, local detectors behave as the S class of Chandra and Toueg and so are more reliable than the distant detectors which behave as the $\diamond S$ class. Moreover, the distant failure detector we define, has the ability to test the failure of a local network via a broadcast address. We prove the correctness of the new algorithm and give some implementations hints in the Internet context.

1 Introduction

In fault-tolerant distributed systems, local or remote processes often require to reach some kind of agreement [12, 3]. As pointed out in [8], the consensus paradigm is not only interesting from a theoretical point of view [12, 11, 5], but it can be used for practical purposes (group's membership, management of replicas, banking, trading, flight reservation etc.).
To reach an agreement, processes launch a consensus protocol. This protocol enables a set of processes to decide on a common value which depends on their initial inputs, despite the possible presence of faulties.
Formally, it requires that 1)if a process decides a value then this value must have been proposed by some process (*Validity*); 2)a process decides at most once (*Integrity*); 3)no two processes decide different values (*Uniform Agreement*); and that 4) every correct process eventually decides some value (*Termination*).
In the absence of faults, attaining a satisfactory mutual agreement is an easy matter. However such an hypothesis is seldom satisfied in real systems.
In purely asynchronous systems, Fisher et al [6] have shown the impossibility to solve the consensus problem even with at most one faulty process. In order to circumvent this negative result, several authors introduced new models by strengthening the assumptions about the environment. One of them is the *asynchronous system model with unreliable failure detectors* [2]. This model is interesting since it does not make restrictive assumptions on the environment.
Different algorithms solving the consensus problem have been proposed. Nevertheless, most of them are not intended for wide area networks such as Internet. Indeed, Internet provides opportunities and challenges for collaborative distributed applications [1]. In previous works [9, 10], we have proposed protocols which solve consensus problem in a system involving local networks. These protocols use a class $\hat{S}$ of failure detector that we have defined. This class is an extension of the class S of unreliable failure detector defined by Chandra and Toueg. We have shown that our proposition brings more benefits with respect of the exchanged messages than the original algorithm in the context of WANs. The new consensus algorithm we develop have similar goals than those mentioned

in [9, 10]. It enables local strategies for deciding values, it consumes less communications messages (and especially the long-distant communications) and uses anonymous communications between LANs. Each LAN is represented by its current non-faulty processes (we assume one process per machine). This machine form a physical group and simulate a macro-process which participates to the wide consensus. Within any LAN, the processes apply a local consensus protocol during each macrostep of the global protocol in order to ensure **consistency of processes' states and actions**. The main contributions of the new protocol are to guarantee that a value decided is a value proposed by some domain. In addition, the protocol terminates not only if some processes crashed but even if some domains become unreachable. Moreover, our approach takes into account the difference between communications aspects in local networks and the technical constraint in WAN setup. Hence, it considers two forms of hypothesis especially for our failure detectors : stronger hypotheses in LAN and weaker ones in WAN. The present work assumes that :
- our environment involves groups of processes located in LANs. Inside a LAN, communications are point-to-point and distant communications between LANs are anonymous via broadcast addresses,
- any process maintains an unreliable failure detector to test the failure of another process in the same LAN or the global failure of another LAN. For reason scale, the new unreliable failure detector called $S \diamond S$ combines two detector classes: the class S used within LAN and $\diamond S$ class between domains, but adapted to WAN requirements.
This paper is organized as follows. Section two details our system model as well as the hypotheses on the detectors. Section three describes the $S \diamond S$ protocol, sketches a proof of its correctness and discusses about some implementation issues. Section four concludes the paper and present some upcoming work.

2 The system model

We consider an environment (e.g. Internet) where E represents the WAN and is constituted of a set of domains. $E = (D_1, ..., D_g, ..., D_G)$ with G the number of domains. Each domain D_g includes a fixed number of processes. $D_g = \{P_{(g,1)}, ..., P_{(g,p)}, ..., P_{(g,Ng)}\}$ where $P_{(g,p)}$ denotes process p of S_g and Ng is the number of processes of the domain D_g. Each $P_{(g,p)}$ has a network address known by the processes of D_g and each domain D_g maintains a broadcast address known by the processes of the other domains. Subsequently, $P_{(g,p)}$ (resp. D_g) will denote both the process (resp. the domain) and its network address (resp. its broadcast address). We consider an asynchronous system where underlying communication network is reliable. Thus, messages are not lost, generated or garbled. To ensure the communication inside the group, we use the following primitives : 1) *R_GroupBroadcast* is used when a process need to send a message reliably to the all members of E via the broadcast address of their respective domain (Refer to [1, 2] for some implementations). 2) *R_Deliver* required to deliver reliably to the application a message received at a network address. 3) Moreover, when a process waits for a message from a domain it considers only the first message sent by one of its members and stops its waiting statement on the broadcast address of the expected domain.

2.1 Failure detection model

The unsolvability of the consensus problem in asynchronous distributed systems due to the difficulty to detect failures in a complete and accurate way in such system, leads Chandra and Toueg to introduce the concept of *unreliable failure detectors*. A failure detector is a module that gives to a process hints about failures of the other processes of the systems with which it communicates. Chandra and Toueg pointed out that there are two undesirable behaviours for a detector : not detecting a faulty process and suspecting a correct process. One way to control the detectors is to require a kind of *completeness* (i.e. detecting a faulty process) and *accuracy* (i.e. not suspecting a correct process). Completeness enforces the detection of any faulty process while accuracy restricts the mistakes (i.e. incorrect suspicions) the failure detectors can make.
In our model, we consider only correct or fail-stop process (resp. domain). A **domain is correct as long as it has a correct process, otherwise it is faulty**. We do not address Byzantine failures.
Any process has a *Combined Failure Detectors* to control processes of its own domain or the other domains. It cannot test individually the failure of remote processes (let us recall that it does not even know their network addresses). Our failure detection model $S \diamond S$ relies on two classes ($\diamond S$ and S) of unreliable failure detectors [2]. $\diamond S$ class is characterized by strong completeness and eventual weak accuracy properties while S class is characterized by strong completeness and weak accuracy properties (see previous section).
The *Combined Failure Detectors* (**CFD**) representing our class $S \diamond S$ failure detector behaves in two ways : In internal communication case it acts like class S as a *Member Failure Detector* (**MFD**) to inspect the other

members of a domain . And in the context of communications between domains, its acts as a *Domain Failure Detector* (**DFD**) on domain address to control if some domain is non-operational.
The properties that describe the combined unreliable failure detectors classes $S \diamond S$ are explained as follows :
Combined Accuracy
-*Domain Weak Accuracy* : In each domain, there is a correct member which will never be suspected by any member of this domain **as long as the domain remains non-faulty**.
- *Wide Eventual Weak Accuracy*: In the environment E, there is a time after which some correct domain will no longer be suspected by any correct member of E
Combined Completeness
- *Domain Strong Completeness*: Each faulty member of a domain, will be eventually suspected by any correct member of this domain.
- *Wide Strong Completeness* : In the environment, each faulty domain will be eventually suspected by any correct member of any correct domain.

3 The Wide consensus

3.1 Informal description of the protocol

All the processes of each domain behave in uniform way with respect to the other domains. For local agreement, they execute a local consensus protocol. This protocol may differ from those executed in the other domains. During the wide protocol, the reception of an external message sent by another domain, compells the receiver to launch a local consensus. This enforces all members of each domain to be synchronized in order to ensure local consistency within each domain. Another point that differentiates our algorithm to the originate one is the proposed values. In our case, input values are not individual processes' values but domain values agreed by local consensus. Thus, a domain could be viewed as a macro-process.
The algorithm works in asynchronous rounds :
At each round some domain coordinates the whole group of domains. We note this "coordinating domain" C_g (equal to the domain round number modulo G). A process which belongs to this domain is called a "*coordinating*" process (noted $P_{(c,p)}$) and the processes of the other domains "*simple processes*".
-Phase 0: At the beginning of the protocol, each domain chooses its value by the local consensus procedure involving all members of the respective domain. The *LocalConsensus* procedure is quite similar to of Chandra-Toueg's *Propose* procedure, excepted that there is an additional parameter that distinguishes different executions of this function (All processes' messages will be timestamped with this parameter). Each domain's member proposes for the local consensus a value that it wishes to be the domain's input value for the wide consensus protocol. At the end of the local consensus, all domain's members agree on a value.
Then, at each round, it executes partially or totally the four phases that follow with respect to the role it play during the protocol **as long as a final decision has not been taken**.
- **Phase 1** : Processes send their domain's *estimate* to the *coordinating* domain via its broadcast address. The first estimates are initial domains' inputs.
- **Phase 2** : Each process of the current "coordinating domain" gathers $\lceil (G+1) \rceil / 2$ estimates related to the first messages received from $\lceil (G+1) \rceil / 2$ domains. When a member collects this number of *estimates*, it selects the most recent defined. It launches the local consensus procedure in order to be synchronized with its peers. After that, any *coordinating* member sends its domain's *estimate*. This estimate is the one with the highest timestamp (i.e. the highest round)
- **Phase 3** : In this phase the "simple processes" send an acknowledgment if they receive the estimate of the coordinating domain and a non-acknowledgment if they suspect this domain.
- **Phase 4** : As phase 2, the coordinating domain wait for a majority of acknowledgments or non-acknowledgments. The first process which reaches this number launches a local consensus. If the consensus gives only acknowledgments, the processes decide from the estimate of the expected domain.

3.2 The algorithm

We consider the point of view of the process p of a domain g noted $P_{(g,p)}$ or more simply (g,p).
Notations and data structures
ADP represents the domains participating to the wide consensus; $\triangle_{(g,p)}$ *is a vector which contains messages received at different rounds*; $estimate_{(g,p)}$ is (p,g)'s estimate (proposed value); $r_{(g,p)}$ is p's current round number; $ts_{(g,p)}$ is the last round in which (g,p) updated $estimate_{(g,p)}$, initially set to 0; r_g is the last round in which the domain g updated $estimate_g$; $\alpha \in \mathbf{CFD}_{(\mathbf{g},\mathbf{p})}$ means that process (g,p) suspects α by querying its combined failure detector **CFD**.
A process (g,p) executes some phases of the algorithm below with regard to the role played by its domain.

CombinedWANConsensus($estimate_{(g,p)}$)
Phase 0 (local): /* Executed by each member (g,p) of a domain g*/
0.1. $\triangle_{(g,p)}[r_{(g,p)}] \leftarrow InputValue_{(g,p)}$
0.2. $state_{(g,p)} \leftarrow undecided$
0.3. $ts_{(g,p)} \leftarrow 0$
0.4. $r_{(g,p)} \leftarrow ts_{(g,p)}$
0.5. /* Each domain g calculates its $estimate$*/
0.6. **LocalConsensus**($\triangle_{(g,p)}[r_{(g,p)}]$,$r_{(g,p)}$)
{The result is returned in $\triangle_{(g,p)}[r_{(g,p)}]$}
/* Rotate through coordinating domains until decision is reached*/
While $state_{(g,p)} = undecided$
$r_{(g,p)} \leftarrow r_{(g,p)} + 1$
$c_g \leftarrow (r_{(g,p)} \bmod n) + 1$
{c_g is the index of the current coordinating domain}
Phase 1 : /* Each process p sends the $estimate_{(g,p)}$ to the current coordinating domain*/
1.1. **Send**$(p, r_{(g,p)}, estimate_{(g,p)}, ts_{(g,p)})$ **To** $c_{(g,p)}$
Phase 2 /*Each member of the coordinating domain gathers $\lceil (G+1) \rceil /2)$ estimates and proposes a new domain's estimate*/
2.1. **If** $(g,p) \in c_g$ **Then**
2.2. **Wait until** [**For** $\lceil (G+1) \rceil /2)$ domains g',
2.3. $\exists (g',q)$: **Received** $((g',q), r_{(g,p)},$
2.4. $estimate_(g',q), ts_(g',q))$ **From** $(g',q)]$
2.5. $\triangle_{(g,p)}[r_{(g,p)}] \leftarrow$
2.6. $\{((g',q), r_{(g,p)}, estimate_(g',q), ts_(g',q))$
2.7. $|(g,p)$ **Received** $((g',q), r_{(g,p)},$
2.8. $estimate_{(g',q)}, ts_(g',q))$ **From** (g',q) $\}$
2.9. **LocalConsensus**($\triangle_{(g,p)}$, $r_{(g,p)}$)
2.10 $t \leftarrow$ largest $ts_{(g',q)}$ **Such That**
2.11 $((g',q), r_{(g,p)}, estimate_{(g',q)}, ts_(g',q))$
2.12 $\in \triangle_{(g,p)}[r_{(g,p)}]$
2.13 $estimate_{(g,p)} \leftarrow$ **Select one** $estimate_{r_{(g',q)}}$
2.14 /* Deterministic selection */
2.15 **Such That**$(q, r_{(g,p)}, estimate_{(g',q)},$
2.16 $ts_{(g',q)}) \in \triangle_{(g,p)}[r_{(g,p)}]$
2.17 **Send** $(p, r_{(g,p)}, estimate_{(g,p)})$ **To ADP**
Phase 3 /*All processes wait for the new domain's estimate sent by a member of the current coordinating domain*/
3.0. $estimate_c \leftarrow null$
3.1. **Wait until** $[\exists (g,p) \in c_g$
3.2. **Received**$(c_{(g,p)}, r_{(g,p)}, estimate_{c_{(g,p)}})$
3.3. **From** $c_{(g,p)}$ **Or** $c_g \in CFD_{(g,p)}]$
3.4. **LocalConsensus**($estimate_{c_{(g,p)}}$, $r_{(g,p)}$)
3.3. **If** $estimate_c \neq null$ **Then**
3.5. /* p received $estimate_{c_{(g,p)}}$*/
3.6. $estimate_{(g,p)} \leftarrow estimate_{c_{(g,p)}}$
3.7 $ts_{(g,p)} \leftarrow r_{(g,p)}$
3.8 $r_g \leftarrow ts_{(g,p)}$
3.9. **Send** $(p, r_{(g,p)}, ack)$ **To** $c_{(g,p)}$
3.10 **Else**
3.11 **Send**$(p, r_{(g,p)}, nack)$ **To** $c_{(g,p)}$
3.12 /* p suspects that c_g crashed */

Phase 4 {Each member of the current coordinating domain waits for $\lceil (G+1) \rceil /2$ replies. If $\lceil (G+1) \rceil /2$ domains adopted their domain's estimate, all correct members of the coordinating domain R-broadcast a decide message after a local consensus }
4.0. **If** $(g,p) \in c_g$
4.1. **Wait until** [**For** $\lceil (G+1) \rceil /2)$ domains g',
4.2. $\exists (g',q)$: **Received**$((g',q), r_{(g,p)}, ack)$
4.3. **Or** $((g',q), r_{(g,p)}, nack)]$
4.2. $V_{g,p} \leftarrow null$
4.3. **If** [**For** (G+1)/2 domains g', $\exists (g',q)$:
4.4 **Received** $((g',q), r_{(g,p)}, ack)]$
4.5. $V_{g,p} \leftarrow ack$
4.6. **Else**
4.7. $V_{g,p} \leftarrow nack$
4.8. **LocalConsensus**$(V_{g,p}, r_{(g,p)})$
4.9. **if** $V_{g,p} = ack$
4.10 **R_GroupBroadcast** $((g,p), r_{(g,p)},$
4.11 $estimate_{(g,p)}, decide)$
/* If (g,p) R-delivers a decide message,(g,p) decides accordingly */
H.0. **When** R-delivers $((g',q), r_{(g',q)},$
H.1. $estimate_{(g',q)}, decide)$
H.2. **If** $state_{(g,p)} = undecided$
H.3. **Decide**$(estimate_{(g',q)})$
H.4. $state_{(g,p)} \leftarrow decided$

Figure 1 : The $S \diamond S$-Consensus protocol

3.3 Proof's sketches of the consensus conditions

The proof relies mainly on the two results *R1* and *R2* below. For simplicity, we confine our demonstration on sketches since we follow the Chandra-Toueg proof's steps. The reader can find more details in [2]. Let us recall that our proposition relies on the hypotheses that there is at least one correct simple process which is never suspected as long as its domain is not crashed and that there is a majority of domains which are correct during any run.
-*R1* :**Two processes will take the same decision** Let us take the time at which the first decision (estimate) has been taken. The coordinating processes have eventually received a majority of acknowledgements. That implies that $\lceil (G+1) \rceil /2$ of domains detain this estimate. We will prove that this estimate holds indefinitely. We apply the demonstration in [2]

related to processes to domains. It is proved by recurrence that if a coordinator (a coordinating domain in our case) decides a value during a round **Rd**, the next coordinating domain will decide the same value during the round **FurtherRd**. The proof is by induction on the round number. Suppose that this holds during a round **NextRd** with **Rd** $\leq$ **NextRd**. We will show that this is also verified at a round **FurtherRd** with **NextRd** $\leq$ **FurtherRd**. During the round **FurtherRd** the current coordinating domain collects a majority of preceding estimates and takes the most recent one. Since $\lceil (G+1) \rceil /2$ domains detain have kept the estimation which leads to the decision, the coordinating domain will necessary receive this estimate. And this estimate is the most recent one and will be selected by the coordinating domain. By recurrence that holds in the following rounds.
-*R2* :**No process is blocked on a wait statement** Let us suppose that a process blocks on a wait statement and let us take the earliest round and the earliest point on that round where this blocking occurs. If it is local consensus then all all correct simples processes are participating. By the consensus (local detectors) properties there can't be a blocking [2]. If it is an external waiting statement by a simple process, the wide strong completness property ensures that the waiting will stop. If it is an external waiting statement by a coordinating process, the hypothesis of a majority of correct domains guarantees the reception of at least a majority of messages (by the hypotheses) and consequently no blocking could occur.
The required properties are satisfied. The *Validity* property is verified. All the *estimates* received by a coordinating member are values proposed by domains (thanks to the *LocalConsensus* procedure (*line 0.6*)). It is obvious that each correct member in the environment decides at most once (*line H.3*()*Integrity*). The proof of the *Uniform Agreement* is very close to the one presented by Chandra and Toueg. In fact, no two processes (resp. domains,if we consider a domain as a macro-process) decide differently. It is a direct consequence of result *R1*. The *Termination* condition relies on the completeness property and on the result *R2*.

3.4 Discussion and implementations issues

Let G be the number of domains, M the fixed (for simplicity) number of members in each domain. In the wide area network we have $N = GM$ processes. During a round, the number of long-distant exchanged messages is divided by M when using our approach. Furthermore, the advantage of our protocol is twofold : 1) It guarantees in a group context, a common knowledge and states of action in any step of the global protocol; 2) It converges more quickly than the original algorithm since each coordinating member waits only $\lceil (G+1) \rceil /2$ messages instead of $\lceil (N+1) \rceil /2$ messages before taking some decision.
For prototyping purpose, one have to focus on two main implementation problems :the group communication and the implementation of detectors.

3.4.1 Domains and broadcast address

We have retained the Object Group Service (OGS)) running on the top of a CORBA[1] environment.
It provides a reliable support for group communication [4]. The object messaging service gives basic mechanisms for managing asynchronous point-to-point messages. Thus, it overcomes the limitations of the standard CORBA object invocations based on the synchronous RPC-like mechanism. According to our protocol, groups of objects have the role of domains assuming that objects which belong to a group are located on a LAN. Groups are distinguished by their *references* like CORBA objects references. A group is a set of a various number of objects; their location is hidden from the entity that holds the reference and may vary over the time.

3.4.2 Implementation issues of the combined failure detectors

The *Eventual weak accuracy* property of class $\diamond S$ guarantees that there is a time after which a correct domain will no longer be suspected by any correct domain. This suits with Internet environment since it enables to take into account the constraint of routing, the policy or means of the providers. In the other hand, the class S of failure detector, with its *Weak accuracy* condition, though stronger than the "eventual" version, is quite adequate to LAN. In contrast to Internet, it is frequent to have high data flow in LAN and transit delay is predictable. The uncertainty stems from the potential servers' overloads. The combination of these two failure detectors represents an equitable mixture for real applications. It is conceivable to admit more reliability inside LAN than between LANs.
Different implementations of the unreliable failure detector have been proposed. Some of them fit only with LAN [13] or can't be used to solve the consensus in asynchronous systems[7]. It has been demonstrated that completeness property of the detector is

[1]CORBA: Common Object Request Broker Architecture, proposed by the Object Management Group(OMG) [?].

not difficult to achieve while no algorithm can satisfy eventual weak accuracy in asynchronous systems. That holds also in our case. Although, Guerraoui and Schiper proved that it is sufficient if a failure detector satisfies this property "long enough" [8]. Different kind of implementations of failure detectors are considered :1)general implementations where each process send periodically a *I_m_alive* (resp. *Are_you_alive*) message when the expected process needs to inform (resp. to inquire after) the processes with which it communicates. And 2) implementations adapted to consensus algorithm where failure detectors' messages are only sent in relevant steps of the algorithm. The first model is more general but it is costly in terms of messages exchanged. The last one has the advantage to reduce network usage in terms of long-distant messages. Since the detectors are based on time-out mechanism there is a trade-off between latency (short timeouts to detect rapidly a failure) and accuracy (long timeouts to avoid incorrect suspicions).

4 Conclusion

We have presented a mecanism which can be viewed as a building block for fault-tolerant distributed systems involving groups and where a group agreement is required. It relies on a new failure detector class we have defined and called $S \diamond S$. This detector class is a combination of two classes of Chandra and Toueg's unreliable failure detector named S and $\diamond S$ that we have customized to WAN requirements. The new protocol involves subsets of participants instead of individual participants. The new approach is hierarchical and allows a better adaptation of the combined failure detectors parameters (such as timeouts) to the topology of the network and reduces the number of messages exchanged in the system between LANs as pointed out in[4]. Furthermore, it takes into account real characteristics of WAN and LAN environments by considering weaker detectors' hypothesis for LANs and stronger ones for WANs. In addition, it provides *flexibility* since the domains are anonymous and the local networks may be different. The protocol works with only a minimum knowledge about the domains and the local consensus can be implemented differently.

Acknowledgements

Work supported in part by the Franco-Moroccan Integrated Action under contract 97/074/SI/R1

References

[1] O. Babaoglu and A. Schiper. On group communication in large-scale distributed systems. *OS review, JACM*, 29(1):62–67, Jan. 1995.

[2] T. D. Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *JACM*, 43(2):225–267, 1996.

[3] D. Dolev, N. A. Lynch, S.S. Pinter, E.W. Stark, and W.E. Weil. Reaching aproximate agreement in the presence of faults. *Journal of ACM.*, 33(3):499–516, July 1986.

[4] P. Felber. *The CORBA Object Group Service.* PhD thesis, EPF, Lausanne-Suisse, 1998.

[5] C. Fetzer and F. Cristian. On the possibility of consensus in asynchronous systems. In *Proc. of the Int. Symp. on F-T systems*, Dec. 1995.

[6] Michael J. Fisher, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *JACM.*, 32(2):374–382, April 1985.

[7] V. K. Garg and J. R. Mitchell. Implementable failure detectors in asynchronous systems. Technical report, May 1998. TR-PDS-1998-004.

[8] R. Guerraoui and A. Schiper. Consensus : the big misunderstanding. In *Proc. of the IEEE Int. Workshop- FTDCS'97, Tunis*, Oct. 1997.

[9] S. Haddad, F. K. Nguilla, and A. El-Fallah S. A consensus protocol for wide area networks. In *Proc.of DAPSYS'98, Budapest*, Sept. 1998.

[10] F. Nguilla Kooh. Hierarchical approach for solving agreement problems in wide distributed systems. In *Proc. of IASTED Int. Conf. PDCN'98, Brisbane-Australia*, 1998.

[11] L. Lamport and M. Pease R. Shostak. The byzantine generals problem. *ACM Trans. Program. lang. Syst.*, 4(3):52–78, July 1982.

[12] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM.*, 27(2):228–234, April 1980.

[13] N. Sergent. *Soft real-time analysis of asynchronous agreement algorithms using petri nets.* PhD thesis, EPF, Lausanne-Suisse, 1998.