1

A Consensus Protocol for Wide Area Networks*

Serge HADDAD, LAMSADE
haddad@lamsade.dauphine.fr

François NGUILLA KOOH, LAMSADE, ESIGETEL
nguilla{@lamsade.dauphine.fr,@esigetel.fr}

Amal EL-FALLAH SEGHROUCHNI, LIPN
elfallah@lipn.univ-paris13.fr

LAMSADE: Laboratoire d'Analyse et de Modélisation des Systèmes pour l'Aide à la Décision. UPRESA 7024, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, F-75775 Paris Cedex 14, France.
LIPN: Laboratoire d'informatique de l'Université Paris-Nord. UPRESA 7030, Institut Galilée, Av. J.-B. Clément, F-93430 Villetaneuse.
ESIGETEL: Ecole Supérieure d'Informatique et de Génie des Télécommunications, 1 rue port de Valvins 77215 Fontainebleau-Avon.

Abstract— **We propose a consensus protocol for wide area networks (WAN) composed of interconnected local networks. We follow the approach initiated by Chandra and Toueg, based on unreliable failure detectors for asynchronous systems. We introduce a new detector class related to WAN properties similar to the detector of class S. The main new functionality of our detector is the ability to test the failure of a domain via a broadcast address. The algorithm described in this paper, allows each local network to apply its own decision strategy for the local proposed value. After proving the correctness of our algorithm, we will show that the number of exchanged messages is one order less than that of the original algorithm.**

Keywords— **Consensus, WAN, Broadcast Address, Unreliable Failure Detectors**

I Introduction

Most dependable services in distributed systems rely on some kind of agreement. The consensus problem is considered as a general paradigm for agreement. Although a huge number of algorithms have been defined, most of them are based on mutual knowledge of the participants. Fault-tolerant systems often require means by which independent processors or processes can reach an exact mutual agreement [1] or Byzantine agreement [2]. In order to obtain such an agreement, the consensus paradigm has been introduced by several authors ([3] [4] [5] etc). As pointed out in [6], consensus is not only interesting from a theoretical point of view but it has numerous applications in fault-tolerant distributed systems.

Roughly speaking, a consensus protocol enables a system of processes, in which some may be faulty, to reach agreement.

In the absence of faults, attaining a satisfactory mutual agreement is an easy matter. However this hypothesis is unrealistic in distributed systems. So the model of such systems must include more general assumptions on their behavior. The usual alternative assumptions are, on the one hand, the delay of the communications and computations (synchronous versus asynchronous) and, on the other hand, the kind of faults (crash failures versus byzantine failures).

In purely asynchronous systems, Fisher et al [4] have shown the impossibility to solve the consensus problem even with, at most, one faulty process. The idea of the proof is that it is impossible to safely distinguish a crashed process from a very slow one. In order to circumvent this impossibility result, new models have been introduced by strengthening the assumptions about the environment :

- Different *partially synchronous models* are defined depending on the fulfillment of five basic criteria for synchronicity [7] [8]. In this approach, the authors exhibit the minimal combinations that are enough to solve the consensus problem.
- The processes of *timed asynchronous models* manage a local clock and, by restricting the mutual behavior of the clocks, during stable and unstable periods, algorithms are again available [9].
- The processes of *asynchronous system models, with unreliable failure detectors*, test the failure of other processes by a "black-box" mechanism which guarantees some accuracy and completeness properties. For each class of Chandra and Toueg failure detectors [10], an algorithm is defined.

In all of these approaches, the correctness of algorithms may depend on the number of faults.

However, most of the numerous algorithms which solve the consensus are not designed for wide area networks. Our main motivation is to tackle a large scale distributed system that could be developed on environments such as Internet. Indeed, Internet provides opportunities and challenges as infrastructure for collaborative distributed applications [11]. In such systems, an agreement could be necessary between faraway local networks of processes.

Our approach is based on :

- A partition of processes in local area networks (LANs) where local communications are point to point inside a LAN and where distant communications between LANs are "anonymous" via broadcast adresses,
- Unreliable failure detector associated with any process.

***Work partially supported by a Franco-Morocco Integrated Action Num. : 97/074/SI/R1**

This detector enables to test the failure of another process in the same LAN, or the global failure of another LAN. This unreliable failure detector, called $\hat{S}$, has properties similar to the S detector of Chandra and Toueg.

We develop a consensus algorithm with the goals of enabling local strategies to decide values, reducing the number of communications (and especially the far communications) and using anonymous communications between LANs. The principles of the new protocol are twofold. At first, the LANs - each LAN represented by its current non faulty processes - simulate the original protocol. Moreover, within any LAN, the processes apply again the original protocol during each macrostep of the global protocol in order to guarantee consistency of processes' states and actions. In addition to the initial goals, it appears that our algorithm produces less messages than the original one.

To describe our consensus protocol for wide area networks, we organize this paper in five sections. Section two presents the consensus problem and the asynchronous system model with unreliable failure detectors. Section three details our system model emphasizing the hypotheses on the detectors. Section four describes the "WAN" consensus, sketches a proof of its correctness and evaluates its complexity in number of messages. In section five we present practical domains where it could be applied and some possible extensions of this work.

II Solving the consensus problem with unreliable failure detectors

The consensus problem has been initiated by Pease et al [3], and solved in synchronous systems with Byzantine failures by Lamport et al [12] [13]. They consider that processes fail by crashing and communicate only by two-party messages. We describe informally a consensus protocol as follows: at the beginning of the protocol each process is given an input value and at the end, the non-crashed processes must have decided on a common output value belonging to the set of input values.

This problem is defined more precisely by four properties :

- *Validity*: if a process decides a value then this value must have been proposed by some process.
- *Integrity*: a process decides at most once.
- *Agreement*: no two correct processes decide different values.
- *Termination*: every correct process eventually decides some value.

Uniform Consensus Problem is characterized by the same properties except that the *Agreement* property is substituted by the *Uniform agreement* property : no two processes (crashed or not) decide differently. In [14] it is discussed why uniformity is important for recovery. Indeed, the algorithm presented in this paper fulfills the *Uniform agreement* property.

The unsolvability of consensus problem in asynchronous distributed systems is mainly due to the impossibility to detect failures in a complete and accurate way in such system. So Chandra and Toueg introduce the unreliable failure detectors. They pointed out that there are two undesirable behaviors 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 processing) and accuracy (i.e. not suspecting a correct process). There are two parameters for the properties leading to eight classes of detectors :

- Strongness (Weakness) : all (one) detectors fulfill(s) the property.
- Permanent (Eventual) : the detector permanently (eventually) fulfills the property.

We give below two examples of properties which characterize the S detectors class :

- **Strong Completeness**: Eventually every faulty participant is permanently suspected by every correct participant.
- **Weak Accuracy**: There is a correct participant that is never suspected.

Notations

Strong:$\rightarrow$ St; *Weak*:$\rightarrow$ We;
Accuracy $\rightarrow$ A; *Completeness* $\rightarrow$ C;

.....	St A	We A	$\diamond$ (St A)	$\diamond$ (We A)
St C	P	S	$\diamond$ P	$\diamond$ S
We C	Q	W	$\diamond$ Q	$\diamond$ W

Figure 1: Table of failure detectors classes.

Chandra and Toueg show that strong completeness can be achieved by weak completeness. So there remain only four relevant classes : 1) -the perfect detector class P for which the consensus problem is trivial, 2)-the $\diamond S$ detector which is proved in [15] to be the weakest in solving the consensus problem (with the additional constraint that a majority of processes remain correct) and 3) 4) two intermediate classes (S and $\diamond P$). To solve $\diamond P$ class, one needs to use the algorithm for $\diamond S$ class while a specific algorithm is proposed for the S class. Moreover, this algorithm has the advantages that the number of messages exchanged is bounded and that it works for at least one correct process. Our solution for WANs is based on Chandra and Toueg algorithm ("*Propose*") given in Appendix A. This algorithm proceeds in n rounds where n is the number of processes. Each process p maintains a vector of values V_p (one per process) initialized by its own value for its item and by the null value for the other items. In the first round, each correct process broadcasts its initial vector and in the next rounds it only relays (by broadcasting) the new values received at the previous round. According to the first $n-1$ rounds and due to the class S properties, all correct processes receive at least all the values received by the non-suspected correct process (in the current execution). Thus in the last round, they exchange their vector and update it (item by item) with the least defined value. Their vectors are thus all identical to the vector of the non-suspected correct process and include at least one non-null value (the input value of the non-suspected correct process). They make decision by applying any predefined strategy.

III The conceptual framework

We consider an environment E composed of a set of domains :
$E = (S_1, ..., S_g, ..., S_G)$ with G the number of domains.
Each domain S_g includes a fixed number of processes :
$S_g = \{P_{(g,1)}, ..., P_{(g,p)}, ..., P_{(g,N_g)}\}$ where $P_{(g,p)}$ denotes process p of S_g, and N_g is the number of processes of the domain S_g.
Each $P_{(g,p)}$ has a network address known by the processes of S_g and each domain S_g maintains a broadcast address known by the processes of the other domains. Subsequently, $P_{(g,p)}$ (resp. S_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 but with no limit to the transit delay. Thus, it does not lose, generate or garble messages. Each process of a domain can both communicate with the other members of its domain through the network addresses or send a message to all the members of another domain through the broadcast address of this domain.
In our model, a process is correct or crashed (in this case we say that it is faulty). We do not cover the case of Byzantine failure.
Any process has a WAN failure detector for testing the 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 $\hat{S}$ relies on the class S of unreliable failure detectors [10] characterized by strong completeness and weak accuracy properties (see previous section).
There are two ways to address the failure question in the set of domains with respect to internal and external point of view of each domain:

- A member of a domain can be suspected by the other members of its domain. In the case of communication within a domain, the failure detector $\hat{S}$ has the same behavior as the class S of Chandra-Toueg's unreliable failure detectors.
- In an inter-domains communications context, a broadcast address becomes unreachable if and only if all of the domain members are faulty. **If all of these members are crashed, the domain is said to be faulty**, otherwise we say that the domain is correct.

The *Wide Failure Detector* (**WFD**) can be seen as an implementation of our class $\hat{S}$ failure detector. It acts as if each process maintains two failure detectors: in the case of internal communication it acts like a *Member Failure Detector* (**MFD**) to inspect other members. And in the context of communications between sites, it acts as a *Domain Failure Detector* (**DFD**) on a domain address to control if some domain is not operational. We do not discuss how the *DFD* monitoring can be achieved since it is out of the scope of this paper.
The properties that describe the wide unreliable failure detector class $\hat{S}$ are explained as follows.

- *Wide Weak Accuracy* :
  - In each domain, there exists at least one correct member which is never suspected **as long as the domain remains non-faulty**.
  - In the environment E, there is a correct domain which will never be suspected by any member of E.
- *Wide Strong Completeness* :
  - Each faulty member of a domain is eventually suspected by any correct member of this domain.
  - In the environment, each faulty domain will eventually be suspected by any correct domain.

IV Wide consensus: Algorithm, proof and complexity

IV.A General description of the protocol

The basic idea of the algorithm (*LocalConsensus*) is described in four phases:

Phase 0: each domain chooses its value by the local consensus procedure executed by each member of the domain. Note that the *LocalConsensus* is like the *Propose* procedure of Chandra-Toueg's algorithm, except that :
- the proposed value (i.e. the first parameter) is now the current vector of knowledge of the domains values,
- there is a second parameter which distinguishes different executions of this function (all of the processes messages will be stamped with this parameter).

This value will be proposed within local consensus as the domain value in the wide consensus protocol.

Phase 1: $G-1$ asynchronous rounds are proceeded by each process of any domain. Each process of a domain waits for only the first message coming from each domain. In addition, in each round, an additional local consensus is executed to ensure that in the next round, the domain's members will send the same vector of values . This guarantees consistency of actions inside a domain.

Phase 2: The goal of this phase is to ensure that all members of the whole set of domains will have the same vector of knowledge (wide consistency). It is closed to the phase of the *Propose* algorithm and it does not require additional local consensus.

Phase 3 proceeds to make a global decision since at this step, all correct processes share the same vector.

IV.B The algorithm

IV.B.1 Data structure

We consider the point of view of the process p of a domain g noted $P_{(g,p)}$ or more simply (g,p).

- *ADP* is the list of all domains participating in the "WAN" consensus;
- $V_{(g,p)}$ is a vector of the domains values, containing initially for domain g the value proposed by member p, and for the other domains the null value $(-)$;
- $\triangle_{(g,p)}$ is a vector of new values received during the current round which will be sent to other $G-1$ domains. Initially $\triangle_{(g,p)}$ will be set to $V_{(g,p)}$

- *Messages* is an array indexed by rounds containing all correct domains messages received at the corresponding round in phase 1 of the algorithm. (**a domain message is the first message received from this domain**).
- *LastMessage* variable is used to save the messages received by a member of a domain in phase 2 of the algorithm.
- Two communication primitives are used :
  - **Send** $(r_{(g,p)}, \triangle_{(g,p)}, (g,p))$ to ADP means that at a round r, the member p of the domain g sends the vector $\triangle_{(g,p)}$ to all the anonymous addresses of the domains.
  - **Received**$(r_{(g,p)}, \triangle_{(g',p')}$,(g',p')) means that at a round r, the member p of domain g receives the vector $\triangle_{(g',p')}$ sent by a process (g',p') of domain g'.
- **LocalConsensus**$(\triangle_{(g,p)}, r_{(g,p)})$ means that during any round $r_{(g,p)}$process p of domain g proposed as input value its vector $\triangle_{(g,p)}$ for the local consensus and the result is put into the same vector.
- **Max**$(V_{(g,p)}, \triangle_{(g,p)})$ function compares the two vectors component by component and takes the most defined value. (Let us note that there can be no conflict between equally defined values).
  Example :
  Max
  $(\prec V_{(g,p)}, -, V_{(g',p')}, -\succ_4, \prec -, , V_{(g'',p'')}, V_{(g',p')}, -\succ_4$
  gives $\prec V_{(g,p)}, V_{(g'',p'')}, V_{(g',p')}, -\succ_4$
- **GlobalDecisionStrategy** is a function known by all processes in the environment. For instance it can be the one included in Chandra and Toueg algorithm. It allows all processes to take the same global decision value relying on the same vector of domains values.

Each process p of each domain g executes the algorithm below.

WANConsensus(InputValue)
$V_{(g,p)} \leftarrow \prec -, -, ..., -\succ_G$
/*$V_{(g,p)}$:*Vector of proposed values maintained by m*/*

Phase 0 (local):
/ Executed by each member m of a domain g*/*
0.1 $V_{(g,p)}[g] \leftarrow InputValue$
0.2 */* Each g chooses a value*/*
0.3 $V_{(g,p)} \leftarrow$**LocalConsensus**$(V_{(g,p)}, 0)$
0.4 $\triangle_{(g,p)}[S_g] \leftarrow V_{(g,p)}$

Phase 1 :
/*$G-1$ *asynchronous rounds are proceeded**/
1.1 **For** $r_{(g,p)} \leftarrow 1$ **To** $G-1$ **Do**
1.2 **Send** $(r_{(g,p)}, \triangle_{(g,p)}, (g,p))$ **To** (ADP\g) $\cup$ (g,p)

1.3 **Wait Until** $[\forall g' \in ADP, \exists p' \in g'$
Received$(r_{(g,p)}, \triangle_{(g',p')}$,(g',p'))
Or g' $\in WFD$]
/ Query the Wide Failure Detector */*

1.4 $Messages_{(g,p)}[r_{(g,p)}] \leftarrow$
$\{((r(g,p), \triangle_{(g',p')}, (g',p'))$
$|$**Received**$(r_{(g,p)}, \triangle_{(g',p')}, (g',p'))\}$
*/*The Wait statement is non-blocking since each member of a domain waits for only one message of a domain or it suspects this domain*/*

1.5 $\triangle_{(g,p)} \leftarrow \prec -, -, ..., -\succ_G$

1.6 **For** $k \leftarrow 1$ **To** G
1.7 **If** $V_{(g,p)}[k] = -$ **And** $\exists(r_{(g,p)}, \triangle_{(g',p'),(g',p')} \in Messages_{(g,p)}[r_{(g,p)}])$
1.8 **Such that** $\triangle_{(g',p')}[k] \neq -$ **Then**
1.9 $\triangle_{(g,p)}[k] \leftarrow \triangle_{(g',p')}[k]$
1.10 **EndFor**

1.11 **LocalConsensus**$(\triangle_{(g,p)}, r_{(g,p)})$
/ Executed since some members of the same domain would received different values*/*

1.12 $V_{(g,p)} \leftarrow$ **Max**$(V_{(g,p)}, \triangle_{(g,p)})$
/ The new $V_{(g,p)}$ is updated by all values that differ of $-$ taken in $V_{(g,p)}$ and $\triangle_{(g,p)}$ */*

1.13 **EndFor**

Phase 2
2.1 **Send** $V_{(g,p)}$ **To** (ADP\g) $\cup$ (g,p)

2.2 **Wait Until** $[\forall g' \in ADP, \exists p' \in g'$
Received$(V_{(g',p')})$
Or g' $\in WFD]$
/ Query the Wide Failure Detector */*
2.3 $LastMessage_{(g,p)} \leftarrow$
$\{V_{(g',p')} |$ **Received**$(V_{(g',p')}\}$

2.4 **For** k $\leftarrow 1$ **To** G
2.5 **If** $\exists V_{(g',p')} \in LastMessage_{(g,p)}$
With$V_{(g',p')}[k] = -$ **Then**
2.6 $V_{(g,p)}[k] \leftarrow -$

2.7 **EndFor**

Phase 3
3.1 GlobalValue = **GlobalStrategy**$(V_{(g,p)})$

Figure 2: Solving consensus in WAN with the class detector $\hat{S}$.

IV.C Proof of the consensus properties

We follow the same lines as the Chandra and Toueg proof. We only sketch the proof of the algorithm above. The reader can refer to [10] for more details

Result 1: No process is blocked in a wait statement.
Suppose that during a round, a process blocks, and let us take one such process with the earliest round possible. $\hat{S}$ behaves like S inside a domain. Thus the

wait statement of local consensus cannot be the earliest blocking one. Thus, if the expected process waits on a faulty domain, the detector $\hat{S}$ will detect this domain, otherwise the domain has at least a correct process and this process is not blocked at the previous round (by our choice). So it must have sent its message in the current round. This message will eventually be received. This contradict our hypothesis.

Result 2: At the end of each round all processes of any domain share the same vectors $V_{(g,p)}$ and $\triangle_{(g,p)}$. This is a direct consequence of the local consensus execution.

Result 3: At the end of phase 1, each process' $V_{(g,p)}$ vector is more defined than the corresponding vector or any correct process of the non suspected domain, let us call it $V_{(g_0,-)}$.

Each non-null value of the vector $V_{(g_0,-)}$ is obtained during some round. If it is not the last round, this value must have been sent and received by the other processes of the other domains. Otherwise, due to the management of $\triangle_{(g,p)}$ there is a chain of $G-1$ different domains which send this value. Thus all domains ($\neq g_0$) must have received this value

Result 4: With "macro-processes" considerations and since there exists a process in the environment which is never suspected (*Weak accuracy*), all processes in the environment have the same vector. Each process receives $V_{(g_0,-)}$ during phase 2. By *result 3* we can state that all processes have the same vector at the end of phase 2.

We can now assert the required properties.

- The *Validity* property is verified. The global non-null value of the vector $V_{(g,p)}$ are the input values proposed by a member of a domain. The related vector at the end of phase 2 has at least one non-null value which is the one decided at phase 0 by the non-suspected domain. So every outcome value is a domain value since it is a value decided by the *LocalConsensus* procedure (*line 0.3*).
- The protocol satisfies the *Integrity* property i.e. each correct member in the environment decides at most once (*line 3.1*).
- No two members decide differently (*Agreement*) (*line 3.1*). This is a direct consequence of result 4
- *Termination* condition relies on the completeness property (the strong form) of the unreliable failure detector. With respect to the reliability of the communication network it can be argued that a correct member either receives a proposition from a domain by a message of one of the members of this domain or suspects the expected domain. So the wait statement before executing the two phases **Phase 1, Phase 2,** are non blocking. This is a direct consequence of result 1. Consequently, the protocol terminates globally.

IV.D Complexity

Evaluating the cost of distributed algorithm is not an easy matter. It can be done by taking into account the following criteria : the number of messages exchanged, the communication steps or latency degree [16] and in real networks, the termination time. For simplicity, we will confine our evaluation to the messages exchanged and we will fix the number of processes inside one domain to be N. G is the number of domains. Thus, the total number of members in the environment, is P = GN.

In a context without domains (i.e. all members of the environment communicate directly) we obtain the following evaluations of the cost of the communications (number of messages):

Without Domains	**With Domains**
$G^3N^3 + G^3N^3$	$G^2N^2(N-1) + G^2N^2(G-1)$

Figure 2: Evaluation of the number of messages in the two cases .

Without domains considerations : at phase 1, each of the P processes send P messages (messages sent to all processes including itself) in $P-1$ times (related to the $P-1$ rounds). And by adding the number of messages sent at phase 2 we approach P^3 (i.e. G^3N^3 messages exchanged).

With physical partitions, two levels of communication are taken into account:

locally: N^3 messages are exchanged due to the local consensus launched by each member of each domain. So, we have $G*N*(N-1)^3$ for all domains. In phase 1 and phase 0, the local consensus procedure is launched G times. That gives a number of exchanged messages equal to $G*(G(N-1)^3$.

globally (distant communications): Let us consider phase 1 and phase 2. In G rounds N processes of each domain send $N*(G-1)$ messages. These messages correspond to the messages sent to all processes of the environment (i.e. to all domains excluding their own domain (i.e. to $G-1$ domains)). That gives about $G*G*N*(N*(G-1))$.

By combining all of these evaluations we obtain the results in the table above. As we can remark, the gain is approximately an order of messages exchanged equal to G in inter-domains communications and about an order equal to N communications within a domain. It is clear that our proposition brings more benefits with respect to the exchanged messages. Note that communications between domains are more expensive than inside a site. For distributed applications running in a WAN where this kind of consensus is frequently used, our algorithm appears to be a good alternative since it provides more gains than the original one.

V Discussion and some concluding elements

The significance of wide area networks has increased as the population of computers connected to them and the range of software supporting their use has grown [17].

Due to the development of new technologies such as Intranet, Internet, it becomes fundamental to think differently and to conceive protocols which consider parameters or problems arising in such environments. This means, for

instance, conceiving hierarchical algorithms.

In this paper we presented a "WAN" consensus protocol that aims to deal with wide area networks seen as a set of domains of logical or physical components belonging to these domains by their geographic proximity. This protocol uses a class $\hat{S}$ *of Wide unreliable Failure Detector* . This class can be seen as an adaptation of Chandra and Toueg's protocol which solves consensus by using class S of unreliable failure detectors for such environments. The main improvements of our algorithm are :

Flexibility since the domains are anonymous and the local networks can be different. The protocol works with only a minimum knowledge of the domains and the local consensus can be implemented differently.

Performance with respect to the number of exchanged messages.

Sergent presents in [18] the results she obtained by simulation in two network communication models: Ethernet and FDDI. Her work provides clues for adapting the system parameters in order to obtain the best performance for the consensus algorithm in such models . Considering our approach, local participants (to local consensus) may use these "local" parameters. But for "wide" parameters, it remains an open issue. We are going to carry out more investigations in this field.

The algorithm also guarantees reliability, availability (for quality of service requirements), and refinement since a domain can be partitioned into domains and so forth. In addition it deals with real applications involving domain communication where agreement protocols are needed. We plan to implement our class $\hat{S}$, as well as point out the way to combine different classes of failure detectors for solving consensus in a WAN. More precisely, we plan to maintain different failure detector properties depending on local or remote processes.

It is known that Internet provides opportunities and challenges as infrastructure for collaborative distributed applications [11]. In such systems, an agreement could be necessary between faraway processes. In distributed computing, several reasons can be mentioned for using our "WAN" Consensus in group context [11] [19] [20] [21] but with physical semantics :

- To maintain a share state between domains (physical groups) : for fault-tolerance involving relatively small domain dimensions or for load balancing.
- Groupware : for applications such as teleconferencing or any kind of groupware where the group's membership ought to be guaranteed.
- "WAN" Consensus will be useful for some Internet services like the netnews service for distribution of new types of messages to newsgroups located in domains or Internet Request Chat (IRC). It will also be useful in situations where a decision must be taken in order to accept or not a type of message. WANConsensus could also be deeply interesting for the resolution of some kind of technical problems such as routing or name resolution problems.
- For mobility (or migration) when an agreement is required among domains before any successive migration of a particular object.

We are now defining models of implementation of the unreliable failure detectors. A comparison will be made between general implementations of failures detectors and those customized to consensus algorithms as pointed out in [18], but for Internet environment. These models will be used for instance for real cooperative applications with real time guarantees considerations by taking into account the inherent constraints of such environment.

References

[1] Andrzej Goscinski, *Distributed operating systems: the logical design*, Addison-Wesley Publishing Compagny Inc., 1991.

[2] Pankaj Jalote, *Fault tolerance in distributed systems*, PTR Prentice Hall, 113 Sylvan Avenue. Englewood Cliffs, New Jersey 07632, 1994.

[3] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults", *Journal of ACM.*, vol. 27, no. 2, pp. 228–234, April 1980.

[4] Michael J. Fisher, Nancy A. Lynch, and Michael S. Paterson, "Impossibility of distributed consensus with one faulty process", *Journal of ACM.*, vol. 32, no. 2, pp. 374–382, April 1985.

[5] Gabriel Bracha and Sam Toueg, "Asynchronous consensus and broacast protocols", *Journal of ACM.*, vol. 32, no. 4, pp. 824–840, Octobre 1985.

[6] Rachid Guerraoui and André Shipper, "Consensus : The big misunderstanding", *In Proceedings of the IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS'97)*, Tunis, October 1997.

[7] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchrony needed for distributed consensus", *Journal of ACM*, vol. 34, no. 1, pp. 77–97, January 1987.

[8] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony", *Journal of ACM*, vol. 35, no. 2, pp. 288–323, April 1988.

[9] Christof Fetzer and Flaviu Cristian, "On the possibility of consensus in asynchronous systems", *Proceedings of the 1995 pacific Rim International Symposium on fault-tolerant systems*, December 1995.

[10] T. D. Chandra and Sam Toueg, "Unreliable failure detectors for reliable distributed systems", *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[11] O. Babaoglu and A. Shipper, "On group communication in large-scale distributed systems", *Operating systems review, Journal of ACM*, vol. 29, no. 1, pp. 62–67, January 1995.

[12] L. Lamport and M. Pease R. Shostak, "The byzantine generals problem.", *ACM Trans. Program. lang. Syst.*, vol. 4, no. 3, pp. 52–78, July 1982.

[13] L. Lamport and M.J Fisher, "Byzantine generals and transaction commit protocols.", *Technical report*, vol. OP. 62. SRI International, Menlo Park, Calif, 1982.

[14] Michel Raynal, "Fault-tolerant distributed systems: a modular approach to the non-blocking atomic commitment problem", Rapport de recherche INRIA Num. 2973, February 1997.

[15] T. D. Chandra and Sam Toueg, "The weakest failure detectors for solving consensus", *In Proceedings of the 11th the ACM Symposium on Principles of Distributed Computing*, pp. 147–158, 1992.

[16] A.Schiper, "Early consensus in an asynchronous system with a weak failure detector", *Dsitributed Computing*, vol. 10, no. 3, pp. 149–157, April 1997.

[17] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: concepts and design*, Second Edition. Addisson-Wesley Publishing Compagny Inc., 1994.

[18] Nicole Sergent, *Soft real-time analysis of asynchronous agreement algorithms using petri nets*, PhD thesis, Département d'informatique, Ecole polytechnique de Lausanne, 1998.

[19] Kenneth P. Birman, Robert Cooper, and Barry Gleeson, *Design Alternatives for process group membership and multicast*, vol. Reliabble Distributed Computing with Isis Tooklit, pp. 109–132, IEEE Computer Science Society Press, 10662 Los Vaqueros Circle PO Box 3014 Los Alamitos, CA 90720-1264, 1994.

[20] Silvano Maffeis, "Adding group communication and fault-tolerance to corba", *Proceedings of the USENIX Conference on Objected Oriented Technologies, Monterey, CA,*, June 1995.
[21] Silvano Maffeis, "The object group design pattern", *Proceedings of the USENIX Conference on Objected Oriented Technologies, Toronto*, June 1996.

Appendix

A

Chandra and Toueg's algorithm: solving consensus using any detector $D \in S$

We have n processes participating. And each process executes the following protocol:

The algorithm proceeds in n rounds; n is the number of processes. Each process p maintains a vector of values V_p (one per process) initialized by its own value for its item. The other items are initialized with null value $-$. In the first round, each correct process broadcasts its initial vector. In the next rounds it broadcasts the new values received at the previous round. By the first $n-1$ rounds, due to the class S properties, all correct processes receive at least all the values received by the non-suspected correct process (in the current execution). So in the last round, they exchange their vector and update it (item by item) with the least defined value. Their vectors have the same values than the non-suspected correct process's vector. These vectors have at least one non-null value (the input value of the non-suspected correct process). At the end of the protocol all correct processes decide by applying the predefined strategy (the first non-null item).

Procedure *Propose*(v_p)
{
$V_p \leftarrow \prec -, -, ..., - \succ_n$
{ *p's estimate of the proposed values* }
$V_p[p] \leftarrow v_p$
$\triangle_p \leftarrow V_p$

Phase 1 :{asynchronous rounds r_p, $1 \leq r_p \leq n-1$}
For $r_p \leftarrow 1$ **To** $n-1$ **Do**
Send $(r_p, \triangle_p, p)$ **To all**

Wait Until $[\forall q$: **Received**$(r_p, \triangle_q, q)$ **Or** q $\in D_p$]{*Query the failure detector* }
$msgs_p[r_p] \leftarrow \{(r_p, \triangle_q, q) \mid$ **Received**$(r_p, \triangle_q, q)\}$
$\triangle_p \leftarrow \prec -, -, ..., - \succ_n$

For $k \leftarrow 1$ **To** n
If $V_p[k] = -$ **And** $\exists (r_p, \triangle_q, q) \in msgs_p[r_p])$
With $\triangle_q[k] \neq -$ **Then**
$V_p[k] \leftarrow \triangle_q[k]$
$\triangle_p[k] \leftarrow \triangle_q[k]$

EndFor
EndFor

Phase 2
Send V_p **To All**
Wait Until $\forall$ q: **Received**(V_q)
Or q $\in D_p$ /* *Query the failure detector* */
$lastmsgs_p \leftarrow \{V_q \mid$ **Received**$(V_q)\}$

For k $\leftarrow 1$ **To** n
If $\exists V_p \in lastmsgs_p$ **With** $V_q[k] = -$ **Then**
$V_p[k] \leftarrow -$

EndFor

Phase 3
Decide(first non-$-$ component of V_p)
}