A Solution for the Consensus Problem in a Mobile Environment *

Nadjib BADACHE — USTHB, Institut d'informatique, 16111 Bab-Ezzouar, Algeria — badache@ist.cerist.dz

Michel HURFIN — IRISA - INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France — hurfin@irisa.fr

Raimundo MACÊDO — LaSiD-CPD-UFBA, Campus de Ondina, 40170-110 Salvador BA, Brazil — macedo@ufba.br

Abstract

Atomic Broadcast, Non-Blocking Atomic Commitment or View Synchrony are classic agreement problems encountered when designing or implementing fault-tolerant distributed systems. Specific protocols that solve such agreement problems can be designed based on a common building block, namely the Consensus service. Unfortunately, the consensus problem has no deterministic solution in an asynchronous distributed system that is subject to even a single process crash failure. Among the solutions proposed to circumvent this impossibility result, the concept of unreliable failure detectors proposed by Chandra and Toueg is particularly attractive. They defined a protocol that solves the consensus problem when the assumption that the underlying failure detector belongs to the class $\diamond\mathcal{S}$ holds true. Efficient solutions to practical agreement problems can be obtained by changing the validity property which characterized the original consensus problem. The Chandra-Toueg's protocol can be extended to cope with the new definitions of the validity property.

This paper presents an extension of their protocol allowing this fundamental agreement problem to be solved in a mobile environment. In such an environment, the problem is more challenging: based on their initial states, a set of mobile hosts must agree on a common decision despite disconnections, changes of location and failures of mobile/fixed hosts.

keywords: Asynchronous Distributed Systems, Mobile Computing, Fault Tolerance, Consensus Problem, Unreliable Failure Detectors.

*A less extensive version of this paper appeared in the Proc. of the 18th IEEE International Performance, Computing, and Communications Conference, IPCCC'99, Feb/1999, Poenix, Arizona, USA. See [3].

The second author is supported by a France Telecom/CNET Grant 981B123 and the third author is supported by CNPq Grants 300013/87-6 (Project BCG) and 680067/95-7 (Project LOCUS/ProTeM-CC III

1 Introduction

The wide use of portable computers and the advances in wireless networking technologies have greatly enhanced mobile computing which is now a major trend in both the computer and telecommunication industries. Intrinsically, a mobile host (*i.e.*, a portable, handheld or embedded computer) changes its location periodically. The communications infrastructure that ties together mobile hosts is a mix of traditional wired networks and wireless networks (cellular phone networks, satellite microwave networks and local area networks based on infrared, microwave or radio transmission techniques). The connection to the network is usually temporary with periods of (voluntary/involuntary) disconnection. Mobile systems are often subject to environmental adversities which can cause loss of messages or data [15]. A mobile host can crash or suffer from frequent and intermittent disconnections from the rest of the network. Thus, designing fault-tolerant distributed applications in such environments is a complex endeavor.

In recent years, several paradigms have been identified to simplify the design of fault-tolerant distributed applications in a conventional static system. The Consensus paradigm is one of the most fundamental since it abstracts other agreement problems. Given a fixed set of processes, the consensus problem is defined as follows: each process proposes an initial value to the others and, despite failures, all non-crashed processes have to agree on a common decision value, which depends on the initial proposals. Any solution to this basic problem can be used to solve other problems such as non-blocking atomic commitment or atomic broadcast [6, 11, 12, 14]. Non-blocking atomic commitment requires all participants in a transaction to take the same decision, namely COMMIT or ABORT the transaction. Atomic broadcast allows processes to agree on both a set of messages and a single delivery order for these messages. In both examples, a consensus service can be used as a basic building block. The semantics associated with the proposed and decided values differ from one agreement problem to another. Yet, the difficulty is always the same: the decision should not be postpone forever (termination property) and has to be unanimous (agreement property).

Due to its wide applicability, the consensus problem has been extensively studied [6, 7, 8, 9, 10, 11, 12, 13, 14, 16]. Unfortunately, this problem has no deterministic solution in an asynchronous distributed system that is subject to even a single process crash failure [10]. Intuitively, this is because (in an asynchronous setting) it is impossible to distinguish a very slow process (or a process with which communications are very slow) from a crashed process with any certainty. Among the solutions proposed to circumvent this impossibility result, the concept of unreliable failure detectors proposed by Chandra and Toueg is particularly attractive [6]. In this approach, each process is equipped with a failure detector module which provides it with a list of processes it currently suspects to have crashed. A failure detector can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct one. Failure detector's classes have been defined by Chandra and Toueg [6] in terms of two abstract properties, namely *completeness* and *accuracy*. Several protocols have been proposed to solve the consensus problem, assuming that a majority of processes does not crash and that the underlying failure detector belongs to the class $\diamond\mathcal{S}$ (see section 2.2). It has been shown in [7] that these conditions are the weakest ones to solve the consensus problem. These protocols are in no way trivial: this is due to the fact that they do not require reliable failure detectors.

Despite its usefulness, no work has been devoted to this problem in a mobile computing environment (to our knowledge), although a shorter version of our work also appeared in [3]. Unfortunately , existing protocols are not suited to a mobile environment: the consensus problem is even more challenging in such an environment. The aim of this paper is to identify the inadequacies of existing protocols and to propose a solution to the consensus problem in a mobile computing environment. The rest of this paper is organized as follows: in Section 2, protocols that solve the consensus problem in a conventional asynchronous distributed system, are discussed. In particular, the Chandra-Toueg's protocol is briefly described. Section 3 describes the mobile system model we use. A protocol to solve the consensus problem in a mobile environment is presented in Section 4. This solution extends the Chandra-Toueg's protocol to cope with mobility. Finally, we conclude in Section 5. (A correctness proof is given in Appendix A.)

2 Consensus in a Static System

2.1 The Consensus Problem

We consider an asynchronous distributed system consisting of n processes denoted p_1, p_2,..., p_n. Processes communicate and synchronize by sending and receiving messages through channels. The distributed system is asynchronous: no assumptions are made as to the relative speed of processes or message transfer delays. Each pair of processes is connected by a reliable link[1]. A process may fail solely 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 during the course of an infinite run.

In the consensus problem, all correct processes have to reach a common decision on some value v, which must belong to the set of proposed values. The consensus problem is defined in terms of two primitives called *propose* and *decide*. Initially each process p_i selects a value v_i from a set of possible values and invokes the primitive *propose* with this value as a parameter: we say that p_i proposes v_i. A process ends its participation in the consensus by executing *decide(v)*: we say that it decides the value v. Formally the following properties have to be held :

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

In [10], Fischer, Lynch and Paterson have shown that consensus cannot be solved deterministically in an asynchronous system that is subject to even a single crash failure. This impossibility stems from the difficulty in determining whether a process has actually crashed or whether it is simply very slow. To overcome this difficulty, Chandra and Toueg propose to augment the asynchronous model of computation with the concept of *unreliable failure detectors* [6].

[1]This property is assumed for sake of simplicity. As shown in [5, 9], the proposed protocol can also be extended to cope with "fair lossy channels".

2.2 Unreliable Failure Detectors

A distributed failure detector is a set of n failure detector modules, one per process. The failure detector module attached to p_i is an oracle in charge of giving hints about processes suspected to be faulty: it maintains a list $Suspected_i$ containing the identities of processes it currently suspects to have crashed. The failure detector is referred to as unreliable since each module can make mistakes by erroneously adding or removing processes to its list of suspects.

A failure detector is defined in terms of two abstract properties, namely *completeness* and *accuracy*. Roughly speaking, completeness requires that a failure detector suspects every process that actually crashes, while accuracy restricts the mistakes that a failure detector can make.

In [6], Chandra and Toueg define eight classes of failure detectors, depending on the nature of the completeness and accuracy properties. In this paper, we are interested in the $\diamond\mathcal{S}$ class of failure detectors. This class of failure detectors which has been proved to be sufficient to solve consensus [6], and even to be the weakest one [7], is specified by the following two properties :

- **Strong Completeness:**
  Eventually every process that crashes is permanently suspected by every correct process.
- **Eventual Weak Accuracy:**
  There is a time after which some correct process is never suspected by any correct process.

2.3 Consensus Protocols based on $\diamond\mathcal{S}$ Failure Detectors

Several protocols designed to work with $\diamond\mathcal{S}$ failure detectors have been proposed [6, 13, 16]. They all require that a majority of processes is correct. They all are based on the rotating coordinator paradigm and proceed in consecutive asynchronous rounds. Each round r is coordinated by a predetermined process p_c defined by $c = (r \bmod n) + 1$. Thus, processes deal with a crash of the current coordinator by moving to the next round. The accuracy property of the failure detector ensures that there will be eventually a round during which the coordinator will not be suspected. The number of rounds performed by each process is arbitrary: it depends on the occurrence of failures and also on the behavior of the failure detector modules. Consequently, it is possible that not all the processes decide in the same round. So, in each protocol, a specific locking mechanism ensures there is a single decision value.

While Chandra-Toueg's protocol uses a centralized scheme (all messages are to/from the coordinator), the two others use a decentralized scheme (each process sends messages to all processes). In all the protocols, the coordinator of round r tries to impose a particular value as the decision value. Yet, in Chandra-Toueg's protocol, this value is not necessarily the estimate of the coordinator at the time it starts round r: on the contrary, this value is computed after the coordinator has gathered estimates from other processes. For this reason, this protocol (unlike the two others) can be extended to solve a slightly different problem (See Section 2.5). This quality persuaded us to select the Chandra-Toueg's protocol as the basis of the proposed solution.

2.4 The Original Chandra-Toueg's Protocol

In this protocol, each process p_i manages a local variable that represents its current estimate of the decision value (initially, the value of the estimate is equal to the initial value v_i proposed by p_i). During the execution of the successive rounds, this value is updated and converges to the decision value. More precisely, each round is divided into 4 phases.

1. In the first phase, each process sends to the current coordinator its own estimate of the final value.
2. The second phase is only executed by the coordinator. It gathers estimates from a majority of processes[2], and selects the estimate whose timestamp is the greatest one[3]. Then the coordinator suggests this estimate by sending it to all the processes.
3. In the third phase each process p_i waits for the receipt of a new estimate from the coordinator. Either p_i suspects the coordinator to have crashed or p_i receives and adopts the new estimate. In the former case, a process sends a negative acknowledgment to the coordinator. In the latter case it sends a positive acknowledgment and updates the timestamp associated with its new estimate by setting it to the current value of its round counter.
4. The fourth phase is only performed by the coordinator. It waits for a majority of acknowledgment messages. If it receives only positive acknowledgments, it reliably broadcasts a decision message[4]. Otherwise, the coordinator proceeds to the next round.

Note that an estimate is irremediably locked as soon as a majority of processes have sent a positive acknowledgment to the coordinator: then, no other value can be selected to be the final decision. When a process terminates round r, it immediately proceeds to round $r + 1$, except if it has received a decision message with the value v. In this case, the process decides the value v and terminates.

2.5 An Extended Consensus Problem

As formulated, the consensus problem is a pure agreement problem. Each process proposes a credible outcome to the consensus service which then forces the adoption of one of these values. Before launching a consensus, a process has to compute the initial value it will proposed, bearing in mind that this value will perhaps become the decision value. Thus, to solve practical agreement problems, processes must generally execute a preliminary exchange phase before executing the consensus protocol. During this phase each process p_i broadcasts relevant local information and then waits until it either receives information on any process p_k or it suspects p_k. At the end of this phase, each process has its own global view of the current global state and can compute the initial value it proposes to consensus. For example, consider the Non Blocking Atomic Commitment problem which assures that all correct participants in a transaction adopt the same decision, namely

[2]The coordinator is assured to receive at least a majority of estimates, because a majority of processes is correct by assumption.

[3]Initially every estimate is timestamped with 0.

[4]Informally, Reliable Broadcast guarantees that (1) all correct processes deliver the same set of messages, (2) all messages broadcast by correct processes are delivered, and (3) no spurious messages are ever delivered.

COMMIT or ABORT the transaction. During the exchange phase, each process broadcasts its local decision (a YES vote or a NO vote) to the other processes. Then, if it has received a YES vote from each process, it proposes COMMIT to the consensus. If it has received a NO vote or if it has not received a vote from some process that it suspects to have crashed, it proposes ABORT to the consensus.

As investigated in [14], the exchanged phase can be suppressed (and consequently the total number of exchanged messages can be reduced) by adopting another practical building block which extend the original consensus problem defined in Section 2.1. The difference between the extended consensus problem and the original one lies in the validity property. The decision is no longer a value proposed by a process but a collection of values proposed by different processes.

In this paper, we adopt the following validity property:

- **Validity:** if a process decides a set of values V, then the set V contains only initial values proposed by processes and its cardinality is at least equal to α ($1 \leq \alpha \leq n$).

The α parameter is set by the upper layer application program when it requires that a minimal number of processes participate in the underlying agreement protocol. Of course, any protocol used to solve this extended problem do not necessarily terminate if more than $n - \alpha$ processes have crashed definitively.

As shown in [14], slight modifications of the Chandra-Toueg's protocol allow this new problem to be solved. The extended protocol does not require a preliminary exchange phase: the global view is computed by the coordinator while it gathers information from each processes during the second phase of a round. Unfortunately, solving the same problem in an mobile environment is more challenging. Before explaining the inadequacies of the above solution, we briefly describe the particularities of a mobile system.

3 The Mobile System Model

A mobile system is a distributed system consisting of two distinct sets of entities : a set of mobile hosts (MHs) and a set of fixed hosts referred to as Mobile Support Stations (MSSs). The MSSs and the communication paths connecting them form a static distributed system which is similar to the system described in Section 2.1.

A cell is defined as the geographical area covered by a MSS. A MSS serves as a base station if it is able to communicate with the MHs located within its cell via a wireless medium[5]. A MH can directly communicate with a MSS (and vice versa) if and only if this MH is located within the cell serviced by the MSS. In order to send messages to another MH that is not in the same cell, the source MH has to contact its local MSS which forwards the messages over the static network to the local MSS of the target MH. The receiving MSS, in its turn, forwards the messages over the wireless network to the target MH. When a MH moves from one cell to another, an *Handoff procedure* is executed by the MSSs of the two cells.

If its current base station fails by crashing, the connection between a MH and the rest of the system is broken. Yet the MH can reconnect to the network by moving into another cell covered by a correct base station. A MH may fail or voluntarily disconnect from the system. When a MH fails, its volatile state is lost. However, disconnections can be treated as planned failures which can be anticipated and prepared for [2].

[5]A fixed host which is not a base station compares with a base station whose cell is never visited by mobile hosts.

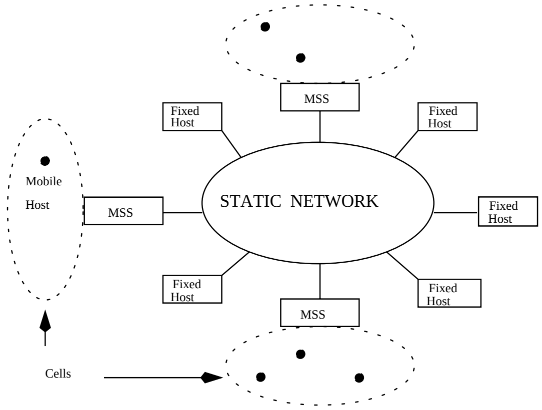

Figure 1: Mobile System Model

3.1 Characteristics of Mobile Hosts

The bandwidth of the wireless link connecting a MH to a MSS is significantly lower than bandwidth of the links between static hosts [2]. In addition, MHs have tight constraints on power consumption relative to desktop machines, since they usually operate on stand-alone energy sources such as battery cells. Consequently, they often operate in a doze mode or voluntarily disconnect from the network. Transmission and reception of messages over wireless links also consume power at a MH. So, distributed protocols for mobile systems need to minimize communication over wireless links. Furthermore, MHs are less powerful than fixed hosts and have less memory and disk storage. Hence, while designing distributed protocols for mobile systems, the following factors should be taken into account [1, 4]:

- the amount of computation performed by a MH should be kept low,
- the communication overhead in the wireless medium should be minimal,
- protocols should be scalable with respect to the number of MHs,
- protocols should easily handle the effects of MHs's disconnections and connections.

4 Consensus in a Mobile System

4.1 The Extended Consensus Problem

In the following, we consider a broadcast group $G = (G\_MSS, G\_MH)$ of communicating mobile hosts, where G_MH is a set of m mobile hosts roaming in a geographical area (like a campus area) covered by a fixed set G_MSS of n base stations. The m mobile hosts are denoted h_1,h_2,$\cdots$,h_m whereas the n base stations are denoted MSS_1, MSS_2,$\cdots$,MSS_n. In so far, local mobile hosts of base station MSS_i will refer to mobile hosts that belong to G_MH and are currently locating in the MSS_i cell. In this environment, the consensus problem is defined over the set G_MH of mobile hosts. Each mobile host h_k proposes a value v_k and the mobile hosts have to decide on a common value V which is a set of values proposed by at least α different mobile hosts. More formally, the new validity property is defined as follows: $(1 \leq \alpha \leq m)$. We assume that at least α mobile hosts will communicate their initial value. In other words, less than $m - \alpha$ mobile hosts have crashed definitively.

4.2 Assignment of Tasks to Mobile and Fixed Hosts

Due to the resources constraints of mobile hosts and the limited bandwidth of the wireless links, the proposed protocol has to be executed by the set of MSSs on behalf of the set G_MH of mobile hosts. We assume that the consensus is initiated by one or several mobile hosts which can be located in different MSSs. Without previously consulting the other mobile hosts, a mobile host requests that its current base station launches the consensus. The contacted base station reliably forwards the request to the other base stations. At the end of this initialization phase, either all (or none of) the correct base stations execute the rest of consensus protocol. Then the activity of a MSS is divided into three main subtasks: (1) a MSS interacts with mobile hosts located in its cell to collect their initial values. (2) a MSS interacts with other MSSs to agree on a subset of proposed values and (3) a MSS interacts with the mobile hosts located in its cell to communicate the final outcome. In our approach, a base station which participates in the consensus protocol, always acts on behalf of a subset of mobile hosts. More precisely, the value V_i proposed by a base station MSS_i is a collection of values proposed by mobile hosts. Initially, V_i contains only values from mobile hosts connected to MSS_i. After exchanging messages with other base stations, V_i will also include values from mobile hosts that have never moved into the cell of MSS_i. While the consensus is not completed, a base station builds up its collection untill it contains values from at least α distinct mobile hosts. When a mobile host enters a new cell, the corresponding base station requests its initial value if the base station is not yet aware of it. The mobile host communicates this value even if it has already given this information to several other base stations[6].

4.3 Inadequacies of the Chandra-Toueg's Solution

As seen in Section 2.4, the locking mechanism used in Chandra-Toueg's protocol relies on the assumption that a majority of processes is correct and participates in each round. From this point of view, mobility appears to be a major difficulty. Let us consider the following scenario: all the mobile hosts are located in the same cell. If the corresponding base station has not crashed, it collects α initial values and proposes this set of values to the coordinator during phase one of a round. The other base stations have to act on the behalf of no mobile hosts: they must participate in the consensus by proposing an estimate equal to the empty set (otherwise the protocol may block). More generally, a base station cannot postpone the sending of its estimate to the coordinator until this estimate contains (possibly) α initial values. Yet, the protocol must allow a base station to communicate a more accurate estimate later.

Now assume that all the mobile hosts are in the cell of a crashed base station. They progressively move to cells managed by correct based stations. Unfortunately, these base stations can already have participated in a round r by sending an estimate equal to the empty set (See above). In that case, the coordinator of round r has possibly received a majority of estimates (all equal to the empty set) and already proposed the empty set as the new estimate. More generally, as the coordinator can be erroneously suspected by some base stations[7], it is not certain it will eventually obtained a new estimate (by piecing together the received proposals) containing at least α initial values. Yet, even if coordinators propose new estimates that are not acceptable, the protocol must ensure

[6] This new exchange is not useless because the base stations previously informed may have crashed.

[7] In that case, these base stations have proceed to the round $r+1$ and will never communicate to the coordinator of round r more accurate estimates.

that the decision value contains at least α values.

This simple example gives an idea of how the original protocol is inadequate to cope with mobility. The next Section outlines the proposed solution.

4.4 The Proposed Protocol

The proposed solution is based on the consensus protocol described by Chandra and Toueg in [6]. As in the Chandra-Toueg's protocol, the consensus is obtained after a sequence of asynchronous rounds. A round r is managed by the base station MSS_c such that $c = (r \bmod n) + 1$.

Whereas the Chandra-Toueg's protocol assumes that a process always sends its estimate once per round (phase 1) and changes its estimate only when it adopts the value proposed by the coordinator (phase 3), our protocol partially removes these limitations. A base station is allowed to change its proposed value while this value does not reflect the decision of at least α mobile hosts. In other words, a base station can change its mind when it adds new values to its uncompleted collection of values. So, during phase 1 of a round, a base station may send up to $\alpha + 1$ messages to the coordinator. Furthermore, after receiving a new estimate from the coordinator, a base station can still send a negative acknowledgment, meaning that it does not agree with the proposed outcome result. More precisely, the value proposed by the coordinator is refused if it contains less than α mobile hosts values.

As soon as a base station has gathered α values and sent a positive acknowledgment to a coordinator, its behavior is similar to that of a process as defined in the original protocol of Chandra and Toueg. However, when a base station decides, it is nevertheless in charge of communicating the decision to the mobile hosts located in its cell.

The protocol is structured into three parts. Part A (see figure 2) describes the role of an arbitrary mobile host h_k. Part B presents the protocol executed by a base station MSS_i. It is subdivided in two sub-parts: sub-part B1 (see figure 3) and sub-part B2 (see figure 4). Sub-part B1 is related to the interactions between a base station and its local mobile hosts (on one hand) and the rest of base stations (on the other hand). Sub-part B2 depicts the adapted Chandra-Toueg's protocol. Finally, the third part C of the protocol is the handoff protocol used to handle the change of location of the mobile hosts.

% Mobile host h_k is located in the cell of MSS_i.

(1) **Upon** the program application requires to start a consensus
  send INIT_1 **to** MSS_i

(2) **Upon** receipt of INIT_3 **from** MSS_i
  % The value of variable Initial_Value is provided by the application program.
  send PROPOSE($h_k, Initial\_Value$) **to** MSS_i;

(3) **Upon** receipt of DECIDE($Decided\_Value$) **from** MSS_i
  % The result of the consensus protocol is delivered to the application program

Figure 2: Protocol Executed by a Mobile Host h_k (Part A)

∗ Local Context of a Mobile Host

- $Initial\_Value$: Value provided by the application program running on a mobile host.

∗ Local Context of a Base Station MSS_i

- $Local\_MH_i$: Set containing the identities of the mobile hosts located in the cell of MSS_i.
- $Suspected_i$: Set containing the identities of the base stations suspected to be crashed. This list is managed by the local failure detector module of MSS_i.
- r_i : Sequence number which identifies the current round of the Chandra-Toueg's protocol executed by MSS_i.
- $Phase_i$: Phase number in a round. Before the consensus protocol starts and after it terminates, $Phase_i$: is equal to 0. Otherwise this variable is either equal to 1, 2, 3 or 4.
- $State_i$: State of MSS_i. $State_i$ is set to *decided* if the consensus has terminated. Otherwise it is set to *undecided*.
- ts_i : Sequence number of the last round during which a new estimate sent by a coordinator has been accepted as the new value of V_i.
- P_i : Set containing the identities of the mobile hosts whose initial values are already known by MSS_i. MSS_i collects values of the mobile hosts located in its cell until $|P_i| \geq \alpha$ holds ($End\_collect_i = true$).
- New_V_i : Set containing the initial values collected by MSS_i.
- V_i : Last set of values proposed by MSS_i.
- $Log_i[r]$: Set containing the estimates received by MSS_i during the r^{th} round (this set is empty if MSS_i is not the coordinator of round r).
- $NB\_PA_i[r]$: Number of positive acknowledgments received by MSS_i during the r^{th} round (equal to 0 if MSS_i is not the coordinator of round r).
- $NB\_NA_i[r]$: Number of negative acknowledgments received by MSS_i during the r^{th} round (equal to 0 if MSS_i is not the coordinator of round r).

∗ **Messages**

- INIT_1: Such a message is sent by a mobile host to its current base station to initiate a consensus. See actions 1 and 4.
- INIT_2: When a base station is asked by a mobile host to initiate a consensus, it broadcasts this message to inform the other base stations that a consensus is started. To ensure a reliable broadcast of message INIT_2, each destination base station has to forward it to the other base stations. So, despite failures of base stations, all (or none) correct base stations will be aware that a consensus has been initiated. See action 4.
- INIT_3: This message is sent to a mobile host either when its base station is informed (on receipt of INIT_2 message) that a consensus has started or when the mobile host enters a new cell managed by a base station MSS_i which is not aware of its initial value and has not yet completed its collection of values ($|P_i| < \alpha$). See the handoff procedure and actions 4 and 2.

[8] $Log[r] := Log[r] \oplus (MSS_j, r, V_j, ts_j)$ is equivalent to two successive operations: (1) $Log[r] := Log[r] \cup \{(MSS_j, r, V_j, ts_j)\}$ then (2) if there exists $(MSS_j, r, V_j', ts_j) \in Log[r] \wedge \exists (MSS_j, r, V_j'', ts_j) \in Log[r]$ such that $card(V_j') \leq card(V_j'')$ then remove (MSS_j, r, V_j', ts_j) from $Log[r]$. This operation is used to keep in $Log[r]$ only the most recent estimate sent by MSS_j during round r.

```
Phase_i := 0; End_collect_i := false; New_V_i := φ; V_i := φ; P_i := φ;
State_i := undecided; r_i := 0; ts_i := 0; Majority := |G_MSS|/2;
for all r : Log_i[r] := φ; NB_PA_i[r] := 0; NB_NA_i[r] := 0; endfor;

cobegin
 (4) || Upon receipt of INIT_1 ∨ INIT_2
          if (Phase_i = 0)
            then send INIT_2 to all MSSs except MSS_i; Phase_i := 1;
                 if (((Local_MH_i ∩ G_MH) ≠ φ) ∧ (¬End_collect_i))
                   then W_Broadcast INIT_3
                 endif
          endif

 (5) || Upon receipt of PROPOSE(h_k, v_k)
          if (¬End_collect_i)
            then P_i := P_i ∪ {h_k}; New_V_i := New_V_i ∪ {v_k};
                 if (|P_i| ≥ α) then End_collect_i := true endif;
                 if (Phase_i > 1) then send ESTIMATE(MSS_i, r_i, New_Vi, P_i, ts_i) to MSS_c endif
          endif

 (6) || Upon receipt of DECIDE(V_j)
          if (State_i = undecided)
            then State_i := decided; V_i := V_j;
                 send DECIDE(V_j) to all MSSs except MSS_i;
                 W_broadcast DECIDE(V_j);
                 Phase_i := 0
          endif

 (7) || Upon receipt of ESTIMATE(MSS_j, r, V_j, P_j, ts_j)
          Log_i[r] := Log_i[r] ⊕ ^8 {(MSS_j, r, V_j, ts_j)};
          if (¬End_collect_i)
            then P_i := P_i ∪ P_j; New_V_i := New_V_i ∪ V_j;
                 if (|P_i| ≥ α) then End_collect_i := true endif
          endif

 (8) || Upon receipt of PA(MSS_j, r_j)
          NB_PA_i[r_j] := NB_PA_i[r_j] + 1

 (9) || Upon receipt of NA(MSS_j, r_j)
          NB_NA_i[r_j] := NB_NA_i[r_j] + 1
```

Figure 3: Protocol Executed by a Base Station (Sub-part B1)

- PROPOSE(-): Such a message carries the value proposed by a mobile host to its local base station[9]. A base station MSS_i takes it into account if $|P_i| < \alpha$. See actions 2 and 5.
- ESTIMATE(-): This message carries the estimate proposed by a base station MSS_i to the current coordinator MSS_c. Each estimate is tagged with a timestamp ts_i identifying the round during which MSS_i has updated its estimate for the last time (see action 12). During round r, MSS_i sends a first ESTIMATE message during action 10. Other ESTIMATE messages can be sent during action 5 when MSS_i updates its collection of values. The estimates sent during round r to MSS_c (MSS_c is necessarily the coordinator of round r), are gathered and logged in a local buffer $Log_c[r]$. A base station MSS_i can propose multiple estimates during a round r but the coordinator MSS_c keeps only the most recent estimate (See the definition of the operator $\oplus$ in footnote 8). While the collect is still possible ($End\_collect_c = false$), the coordinator updates the sets New_V_c and P_c each time it receives the local view of another base station. See actions 10, 5 and 7.
- NEW_EST(-): This message carries the estimate proposed by the coordinator to the base stations. When the coordinator of round r has gathered a majority of esti-

[9]The reader can notice that the value proposed by a mobile host is not required to be always the same. This possibility is not discussed in this paper.

```
(10) || Upon Phase_i = 1
          r_i := r_i + 1; MSS_c := (r_i mod n) + 1;
          if ts_i = 0 then V_i := New_V_i endif;
          Send ESTIMATE(MSS_i, r_i, V_i, P_i, ts_i) to MSS_c;
          if (i = c) then Phase_i := 2 else Phase_i := 3 endif

(11) || Upon (Phase_i = 2) ∧ (|Log_i[r_i]| > Majority)
          Let t_max be the largest timestamp such that there exists
          at least one element (MSS_j, r_j, V_j, P_j, ts_max) ∈ Log_i[r_i]
          if (t_max > 0)
            then Select one of those particular elements
                   denoted (MSS_j, r_j, V_j, P_j, ts_max);
                   V_i := V_j
            else   V_i := New_V_i
          endif;
          Send NEW_EST(MSS_i, r_i, V_i, P_i, End_collect_i) to all;
          Phase_i := 3

(12) || Upon receipt of NEW_EST(MSS_c, r_i, V_c, P_c, End_collect_c)
          if (Phase_i = 3)
            then if (End_collect_c)
                   then V_i := V_c; ts_i := r_i; End_collect_i := true;
                        send PA(MSS_i, r_i) to MSS_c
                   else send NA(MSS_i, r_i) to MSS_c;
                        P_i := P_i ∪ P_j; New_V_i := New_V_i ∪ V_j;
                        if (|P_i| ≥ α) then End_collect_i := true endif
                 endif;
                 if (i = c) then Phase_i := 4 else Phase_i := 1 endif
          endif

(13) || Upon (Phase_i = 3) ∧ (MSS_c ∈ Suspected_i)
          Send NA(MSS_i, r_i) to MSS_c; Phase_i := 1

(14) || Upon (Phase_i = 4) ∧ (NB_PA_i[r_i] + NB_NA_i[r_i]) > Majority
          if (NB_PA_i[r_i] > Majority)
            then send DECIDE(V_i) to all MSSs except MSS_i;
                 State_i := decided; Phase_i := 0
            else Phase_i := 1
          endif
coend
```

Figure 4: Protocol Executed by a Base Station (Sub-part B2)

mates, it selects one estimate from its local buffer $Log_i[r]$ and sends it as a new estimate to all base stations. The selected estimate is either the new estimate sent by a previous coordinator which failed to gather a majority of positive acknowledgments or the set of values New_V_c of the current coordinator. While a base station MSS_i is waiting for a new estimate V_c, it asks its failure detector module whether the current coordinator has crashed or not. If the NEW_EST message is received before the coordinator is suspected and if it carries at least α participant mobile hosts, the base station updates its set of values V_i to V_c and replies with a positive acknowledgment. Otherwise it replies with a negative acknowledgment and next updates its sets New_V_i and P_i. See actions 11 and 12.

- PA(-): Positive acknowledgment sent to the coordinator. If the coordinator gathers a majority of positive acknowledgments, the set V_c is locked and broadcasted as the decided set of values to all base stations. Otherwise the coordinator moves to phase 1 and initiates the next round. See actions 12, 8 and 14.
- NA(-): Negative acknowledgment sent to the coordinator. See actions 12, 13, 9 and 14.
- DECIDE(-): This message carries the decided value. A base station MSS_i receives a message DECIDE(V_j) when a coordinator is aware that a majority of base stations agree upon the set of values V_j . MSS_i adopts this value, changes its state to

decided, forwards the decided set of values to local mobile hosts and terminates. To ensure that all correct processes decide, the message is also forwarded to the other base stations (reliable broadcast). See the handoff procedure and actions 14, 6 and 3.

- GUEST(-): Such a message is sent by a mobile host to inform the current base station when it enters a new cell. See the handoff procedure.
- BEGIN_HANDOFF(-): A BEGIN_HANDOFF message is sent by a base station MSS_i to another base station MSS_j when MSS_i learns that a mobile host has moved from MSS_j cell to its own cell. See the handoff procedure.

• **Role of** h_k
Upon entry in MSS_i cell
    send GUEST(h_k, MSS_j) **to** MSS_i

• **Role of** MSS_i
Upon receipt of GUEST(h_k, MSS_j)
    $Local\_MH_i := Local\_MH_i \cup \{h_k\}$; **send** BEGIN_HANDOFF$(h_k, MSS_i)$ **to** MSS_j;
    if $((Phase_i \neq 0) \wedge (h_k \notin P_i) \wedge \neg(End\_collect_i))$ **then send** INIT_3 **to** h_k **endif**;
    if $((Phase_i = 0) \wedge (State_i = decided))$ **then send** DECIDE(V_i) **to** h_k **endif**

• **Role of** MSS_j
Upon receipt of BEGIN_HANDOFF(h_k, MSS_i)
    $Local\_MH_j := Local\_MH_j - \{h_k\}$

Figure 5: Handoff Procedure (Part C)

The correctness proof of the proposed algorithm is given in appendix A.

5 Conclusion

We have recall the interest of an extended consensus problem and shown how to overcome the difficulties induced by mobility when solving this problem. In an environment with m mobile hosts and n base stations, the proposed protocol tolerates up to $f' = m - \alpha$ mobile hosts failures and $f < \lceil n + 1 \rceil / 2$ base stations failures. The communications over wireless links are limited to a few messages (in the best case, two messages: one to propose the initial value and other to get the decided value). The mobile host's CPU time is low since the actual consensus is run by the base stations. The protocol is scalable: it is independent of the overall number of mobile hosts and all needed data structures are managed by the base stations. The threshold parameter α can be interpreted as a measure of the quality of the decided value delivered by the protocol. The value of parameter α is defined depending on the expected amount of mobile hosts failures that can be tolerated by a given application.

References

[1] Alagar S. and Venkatesan S. Causally ordered message delivery in mobile systems. In *Proc. of the Workshop on Mobile Computing Systems and Applications*, pp. 169–174, Santa Cruz, CA, Dec. 1994.

[2] Badache N. Mobility in distributed systems. Technical Report #962 (in French), IRISA, Oct. 1995.

[3] Badache, N, Hurfin, M, and Macedo, R. Solving the Consensus Problem in a Mobile Environment. In *Proc. of the 18th IEEE Int. Performance, Computing, and Communications Conference, IPCCC'99, Feb 10-12, 1999, Phoenix, Arizona, USA.*

[4] Badrinath B.R., Acharya A. and Imielinski T. Impact of mobility on distributed computations, Operating Systems Review, **27***(2), pp. 15–20, April 1993.*

[5] Basu A., Charron-Bost B. and Toueg S. Simulating reliable links with unreliable links in the presence of process crashes. In Proc. of the 10th Int. Workshop on Distributed Algorithms, *Springer-Verlag, LNCS 1151 (. Babaoğlu and K. Marzullo Eds), pp. 105–122, Italy, October 1996.*

[6] Chandra T.D. and Toueg S. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, **43***(2), pp. 225–267, March 1996.*

[7] Chandra T.D., Hadzilacos V. and Toueg S. The Weakest failure detector for solving consensus. Journal of the ACM, **43***(4), pp. 685–722, July 1996.*

[8] Dolev D., Dwork C. and Stockmeyer L. On the minimal synchronism needed for distributed consensus. Journal of the ACM, **34***(1), pp. 77–97, January 1987.*

[9] Dolev D., Friedman R., Keidar I. and Malkhi D. Failure detectors in omission failure environments. In Proc. of the 16th ACM Symp. on Principles of Distributed Computing, *pp. 286 (brief announcement), Santa Barbara, CA, August 1997.*

[10] Fischer M.J., Lynch N. and Paterson M.S. Impossibility of distributed consensus with one faulty process. Journal of the ACM, **32***(2), pp. 374–382, April 1985.*

[11] Guerraoui R. Revisiting the relationship between non-blocking atomic commitment and consensus. In Proc. of the 9th Int. Workshop on Distributed Algorithms, *Springer-Verlag, LNCS 972 (J.-M. Hlary and M. Raynal Eds), pp. 87–100, France, September 1995.*

[12] Guerraoui R. and Schiper A. Consensus service: a modular approach for building fault-tolerant agreement protocols in distributed systems. In Proc. of the 26th IEEE Int. Symp. on Fault-Tolerant Computing Systems, *pp. 168–177, Sendai, Japan, June 1996.*

[13] Hurfin M. and Raynal M. Fast asynchronous consensus with a weak failure detector. Technical Report #1118, IRISA, Rennes, July 1997 (submitted to publication).

[14] Hurfin M., Raynal M. and Tronel F. A practical building block for solving agreement problems. In Proc. of the IEEE Int. Performance, Computing, and Communications Conference, *pp. 25–31, Phoenix, AZ, February 1998.*

[15] Pradhan D.K., Krishna P. and Vaidya N.H. Recoverable mobile environments: Design and trade-off analysis. In Proc. of the 26th IEEE Int. Symp. on Fault-Tolerant Computing Systems, *Sendai, Japan, June 1996.*

[16] Schiper A. Early consensus in an asynchronous system with a weak failure detector. Distributed Computing, **10***(3), pp. 149–157, 1997.*

Appendix A: Correctness Proof

As our protocol is based on the protocol proposed by Chandra and Toueg, some statements of lemmas and theorems that follow are similar to the ones encountered in [6].

Theorem 1 *If a mobile host decides a value V, then V contains only initial values proposed by mobile hosts of G_MH.*

Proof A coordinator collects only values proposed by base stations (action 7) which have previously collected only values (action 5) proposed by mobile hosts (action 2). Hence, the decided value V contains only initial values of mobile hosts belonging to G_MH. $\square_{Theorem\ 1}$

Theorem 2 *No two mobile hosts decide differently.*

Proof A mobile host decides (action 3) only if its base station has decided (action 6): in that case the mobile host adopts the value broadcast by this base station. Consequently, theorem 2 is valid if two base stations never decide differently. Assume that at least one base station has decided (action 6). In that case, a coordinator has previously broadcast a message DECIDE (action 14). This coordinator must have received a majority of positive acknowledgment in phase 4. So, at least a majority of base stations has adopted the estimate sent by the coordinator in phase 2.

Let r be the smallest round number during which a majority of positive acknowledgments is sent to a coordinator. By assumption no base station has decided during a previous round. Consider a base station MSS_i that has sent a positive acknowledgment to the coordinator MSS_c of round r. During execution of action 12, this base station has also adopted the estimate V_c and updated the value of ts_i to the current round number r. Afterwards, at any round r' such that $r' \geq r$, the value of ts_i will remain greater than zero. Therefore, MSS_i will no more execute the statement $V_i := New\_V_i$ neither during action 10 nor during action 11.

We prove that at any round r' such that $r' \geq r$, the coordinator sends a message NEW_EST that contains necessarily V_c. The proof is done by induction on the round number. Obviously, the claim holds for $r' = r$. Assume that the claim holds for all rounds r' such that $r \leq r' < k$. If MSS_k, the coordinator of round k, executes action 11, it must have received a majority of messages ESTIMATE labeled with the current round number k. Among the base stations which have communicated their estimate to MSS_k, there exists at least one base station MSS_i that has received V_c in its phase 2 of round r and has sent a message ESTIMATE(MSS_i, k, V_i, ts_i) to MSS_k during round k. Obviously, the value of ts_i is greater or equal to r. Assume that the new estimate V_j selected by MSS_k was contained in a message ESTIMATE(MSS_j, k, V_j, ts_j). As the coordinator has selected the largest timestamp, ts_j is greater or equal to r (because $ts_j \geq ts_i$). Furthermore, ts_j is at most equal to $k-1$. Due to the induction hypothesis and because $r \leq ts_j < k$, the value V_j which has been previously broadcast by the coordinator of the round ts_j is necessarily equal to V_c. $\square_{Theorem\ 2}$

Lemma 1 *No base station can block forever in phase 1, 2, 3 or 4 of a round.*

Proof Assume that r is the smallest round number during which a base station remains blocked forever.

No base station remains blocked forever at phase 1 of round r: Every base station executes action 10 and moves either to phase 2 or 3.

Let MSS_c be the coordinator of round r. MSS_c is the only base station that enters in phase 2 during round r. When a base station proceeds from phase 1 to phase 2 or 3, it sends an ESTIMATE message. As a majority of base stations is correct and as communication channels are reliable, MSS_c receives at least a majority of ESTIMATE message sent during phase 1 of round r. Consequently, if MSS_c does not crash, it eventually executes action 11, broadcasts a NEW_EST message and proceeds to the next phase (*i.e.*, phase 3).

No base station MSS_i remains blocked forever in phase 3. If the coordinator MSS_c crashes, MSS_i eventually suspects it and executes action 13. This is ensured by the completeness property associated to the class $\diamond\mathcal{S}$ of failure detectors. If the coordinator is correct, MSS_i eventually receives its new estimate and executes action 12 (communication channels are reliable).

The coordinator is the only base station that enters in phase 4 during round r. At least a majority of base stations executes either action 12 or action 13. Each of them sends a (positive or negative) acknowledgment to the coordinator. As communication channels are reliable, the coordinator will crash or receive a majority of acknowledgments. In the last case, the coordinator either decide or proceed to the next round. $\square_{Lemma\ 1}$

Lemma 2 *Eventually, there is a round during which all correct base stations will send positive acknowledgment to the coordinator.*

Proof Due to lemma 1, if no base station decides during a round r' such that $r' < r$ then all correct base station will execute round r. During this round, a base station sends a negative acknowledgment if either it suspects the coordinator MSS_c or if the NEW_EST message broadcast by the coordinator MSS_c contains a boolean value $End\_collect_c$ equal to false. The accuracy property associated to the class $\diamond\mathcal{S}$ of failure detectors ensures that there is a time after which some correct base station MSS_k is never suspected by any correct base station. After this particular instant, MSS_k when it acts as a coordinator during a round is assured to receive a majority of positive acknowledgments if and only if its local variable $End\_collect_k$ is equal to true.

For any base station, $End\_collect_k$ is a variable initially equal to false. This variable may change from false to true only once during the protocol. This may happen either during execution of action 5, or action 7 or action 12.

We prove that $End\_collect_k$ cannot remain equal to false forever. The proof is by contradiction. Since all faulty base stations will crash after some finite time, there is a time after which MSS_k is not suspected by any correct base station. Due to the fact that channels are reliable, MSS_k will eventually receive the ESTIMATE messages sent by all correct base station. Those messages may have been sent during the current round or during previous rounds. In any case, the base station MSS_k will take all of them into account to update its set of mobile hosts identities P_k and its set of values V_k. $End\_collect_k$ remains false if the cardinality of the set P_k is always less than α. It means that globally the base stations have not gathered values from α distinct mobile hosts. While a base station MSS_i has collected less than α values, it continues to gather information from the mobile hosts. The amount of answers will remain insufficient if and only if there exists at least $m - \alpha$ mobile hosts which either (1) remain disconnected or crashed , or (2) remain forever out of the geographical area or (3) remain in a cell managed by a crashed based station. By assumption, all those situations are transient for at least α mobile hosts. Any mobile host that enters a new cell managed by a correct base station will be asked to communicate its value to a correct base station and the identity of the mobile host will eventually be added to the set P_k. $\square_{Lemma\ 2}$

Theorem 3 *Every correct mobile host eventually decides some value.*

Proof There are two possible cases:

If at least one base station decides and does not crash then all correct base stations eventually deliver a DECIDE message. This is due to the fact that a base station forwards the decision value when it delivers such a message (action 6). Consequently, any mobile host will receive the decided value either when its base station decides (action 6) or when it enters in the cell of a base station that has previously decided.

No correct base station decides. Due to Lemma 2, this case is impossible.

$\square_{Theorem\ 3}$