Solving the Consensus Problem in a Mobile Environment

Nadjib Badache[a*] **Michel Hurfin**[b] **Raimundo Macedo**[c]
badache@ist.cerist.dz *hurfin@irisa.fr* *macedo@ufba.br*
[a]USTHB, Institut d'informatique, BP 32 El-Alia 16111 Bab-Ezzouar, Alger, Algeria
[b]IRISA, Campus de Beaulieu, 35042 Rennes Cedex France
[c]Universidade Federal da Bahia, Campus de Ondina, Salvador, Bahia, Brasil

The consensus paradigm can be used as a building block in many practical problems, like atomic broadcast, atomic commitment, group membership and leader election. It is well known that consensus cannot be solved deterministically in an asynchronous distributed system that is subject to even a single crash failure. Mobile systems are more prone to failures than conventional distributed systems. Solving consensus in such an environment requires from a set of mobile hosts to agree on a common decision based on their initial states despite failures of both mobile and/or fixed hosts. In this paper, we describe a solution to the consensus problem for mobile computing systems, based on the Chandra and Toueg protocol which uses $\diamond\mathcal{S}$ failure detectors.

I. Introduction

The wide use of small portable computers and the advances in wireless networking technologies have made mobile computing today a reality. There are different types of wireless media: cellular (analog and digital phones), wireless LAN, and unused portions of FM radio or satellite services. A mobile host can interact with the three different types of wireless networks at different point of time. Mobile systems are more often subject to environmental adversities which can cause loss of messages or data [12]. In particular, a mobile host can fail or disconnect from the rest of the network. Designing fault-tolerant distributed applications in such environment 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. Consensus is among the most noticeable, particularly since the results of Chandra and Toueg on Failure Detectors [4]. Consensus allows processes to reach a common decision, which depends on their initial inputs, despite failures. Consensus algorithms can be used to solve many problems. The most popular examples are: atomic commitment, group membership, leader election and atomic broadcast. In distributed transactions, all processes need to agree on whether a transaction commits or aborts. In group membership, a set of processes cooperating to provide a highly available service need to agree on which processes are currently functioning as members of the group. Electing a leader requires a group of processes to agree on a leader without knowing exactly what the members are [9]. With the communication primitive called Atomic Broadcast, each process can broadcast messages and all non-crashed processes must deliver those messages in the same order. All the agreement problems (atomic commitment, atomic broadcast, group membership) are related to the abstract consensus problem [8].

Given its wide applicability, consensus has been extensively studied [4, 5, 6, 7, 8, 11, 14, 13]. However, despite its usefulness, to our knowledge, there is no work which has been devoted to this problem in a mobile computing environment.

*This work was performed during a stay at IRISA, Campus de Beaulieu 35042 Rennes Cedex France.

The aim of this paper is to propose a solution to the consensus problem in a specific mobile computing environment. This solution is based on the Chandra and Toueg protocol which uses $\diamond\mathcal{S}$ failure detectors [4]. The rest of this paper is organized as follows: in Section II. , a solution to the consensus problem in a conventional asynchronous distributed system, is presented. Section III. describes the mobile system model we use. A protocol to solve the consensus problem in a mobile environment is presented in Section IV. We conclude at Section V.

II. Consensus in a Static System

We consider an asynchronous distributed system composed of a finite set of processes $\Pi = \{p_1, p_2, \dots, p_n\}$ completely connected. Communication between processes is ensured through reliable links. A process may fail solely by crashing and 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 has to 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 among 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 to 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.
- **Uniform agreement:** No two processes decide differently.
- **Uniform validity:** If a process decides v, then v was proposed by some process.

Solving the consensus problem in distributed asynchronous systems is a subject that has been extensively studied in the last decade. Informally a distributed system is asynchronous if there is no bound on message delay, clock drift or the time necessary to execute a step [4]. In [7], 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 of determining whether a process has actually crashed or is only very slow. To overcome this difficulty, Chandra and Toueg propose to augment the asynchronous model of computation with the concept of *unreliable failures detectors*.

A. The 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 $Suspected_i$ of those it currently suspects to have crashed. The failure detector is called unreliable in the sense that each module can make mistakes by erroneously adding processes to its list of suspects. If this module later believes that suspecting a process p was a mistake, it can remove it from its list.

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.

B. The $\diamond\mathcal{S}$ Failure Detector

In [4], Chandra and Toueg define eight classes of failure detectors, depending on the nature of both completeness and accuracy properties. In this paper, we are interested in the $\diamond\mathcal{S}$ class of failures detectors. This class of failures detectors which has been proved to be sufficient to solve consensus [4], and even to be the weakest one [5], 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.*

C. An Algorithm to Solve Consensus

Chandra and Toueg have proposed in [4] an algorithm that solves consensus using $\diamond\mathcal{S}$ failure detectors provided that a majority of processes are correct. This algorithm is based on the rotating coordinator paradigm, and proceeds by asynchronous rounds. Each round r is coordinated by a predetermined process p_c defined by $c = (r \bmod n)+1$. During round r, a process communicates only with the coordinator p_c. The number of rounds performed by each process is arbitrary: it depends on the occurence of failures and also on the behavior of the failure detector modules. 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. During the first round, a process p_i proposes its initial value v_i.
2. The second phase is only executed by the coordinator. It collects a majority of estimates[1], and selects the estimate whose timestamp is the greatest one[2]. Then the coordinator suggests this estimate to all the processes.
3. In the third phase each process waits for a new estimate from the coordinator, and either suspects it to have crashed or receives and adopts the new estimate. In the former case, a process sends a negative acknowledgment to the coordinator and runs into the next round. 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[3]. 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: after, 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.

III. The Mobile System Model

A distributed mobile system consists of two distinct set of entities : a large number of mobile hosts (MH) and a set of fixed hosts, some of which act as mobile support stations (MSSs) or base station. The non MSS fixed hosts can be viewed as MSSs whose cells are never visited by any mobile host. All fixed hosts and all communication paths connecting them form the static network. Each MSS is able to communicate directly with mobile hosts located within its cell via a wireless medium. A cell is the geographical area covered by a MSS. A MH can directly communicate with a MSS (and vice versa) only if the MH is physically located within the cell serviced by the MSS. At any given instant of time, a MH can belong to one and only one cell. In order to send messages to another MH that is not in the same cell, the source MH must contact its local MSS which forwards the messages to the local MSS of the target MH over the wired network. 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, a *Handoff procedure* is executed by the MSSs of the two cells. Message propagation delay on the wired network is arbitrary but finite and channels between a MSS and each of its local mobile hosts ensure fifo delivery of messages.

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

[2] Initially every estimates is timestamped with 0.

[3] 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.

A. 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, mobile hosts 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 algorithms for mobile systems need to minimize communication over wireless links. Furthermore, mobile hosts are less powerful than fixed hosts and have less memory and disk storage. Hence, while designing distributed algorithms for mobile systems, the following factors should be taken into account [1, 3]:

- the amount of computation performed by a mobile host should be kept low,
- the communication overhead in the wireless medium should be minimal,
- algorithms should be scalable with respect to the number of mobile hosts,
- algorithms should be able to easily handle the effect of mobile hosts disconnections and connections.

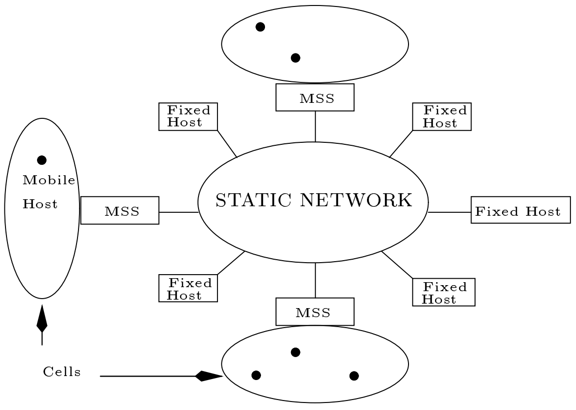

Figure 1: Mobile System Model

IV. Consensus in a Mobile System

In the following, we consider a broadcast group $G = (G\_MSS, G\_MH)$ of communicating mobile hosts, where G_MH and G_MSS are respectively a set of *m* mobile hosts roaming in a geographical area (like a campus area) covered by a fixed set of *n* MSSs. In so far, local mobile hosts of base station MSS_i will refer to mobile hosts that belong to group G and currently residing in MSS_i cell.
A mobile host can move from one cell to another. If its current base station fails, the connection between the mobile host and the rest of the system is broken. To recover its connection, a mobile host must move into another cell covered by an operational or correct base station. So, unless it crashes, a mobile host can always reconnect to the network. A mobile host may fail or voluntarily disconnect from the system. When a mobile host fails, its volatile state is lost. However, disconnections can be treated as planned failures which can be anticipated and prepared for [2].
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 set of values V. Due to the resources constraints of mobile hosts and the limited bandwidth of the wireless links, the distributed algorithm to solve consensus is executed by the set of MSSs on behalf of the set G_MH of mobile hosts. MSSs have to agree in a first phase on a subset of values amongst the set of values proposed by the mobile hosts roaming in their respective cells. In the second phase, each MSS forwards the decided set of values to the mobile hosts that currently reside in its cell.

A. Principle

The consensus protocol proposed in this paper is based on the solution described by Chandra and Toueg in [4]. The outlines of their protocol have been described in Section II. . In this Section, we give an overview of our protocol and identify the major differences with the original protocol of Chandra and Toueg.

We assume that the consensus is initiated by a mobile host[4] which requests its current base station to launch the consensus. The contacted base station forwards the request to all other base stations. At the end of this initialization phase, either all the correct base stations or none of them execute the rest of consensus protocol.

During the consensus, each base station on one hand interacts with the mobile hosts located in its cell to exchange information and on the other hand interacts with the other base stations to reach a common decision on some value V. In our approach, a base station which participates to 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. While the consensus is not completed, a base station builds up its collection till it contains values from at least α distinct mobile hosts. The meaning and the interest of this parameter are discussed later. At a given time, the value proposed by a mobile host h_k may be known by several different base stations. Otherwise, a base station may also act on the behalf of no mobile hosts: this does not prevent the base station from participating to the consensus by proposing an estimate equal to the empty set. An important part of the protocol executed by the base stations is dedicated to the gathering of the mobile host's values. Each base station MSS_i manages two variables, namely New_V_i and P_i. The first variable contains the set of values gathered by the base station. The second variable contains the identities of the mobile hosts whose value is already known. The collect begins when the base station is informed that the consensus has been launched. It ends when the cardinality of the set P_i is greater or equal to α. To build up its collection of values, a base station may ask a mobile host for its own value if (1) the base station is not yet aware of its value and (2) the mobile host is currently in its cell. The base station can also take into account the information gathered by other base sta-

[4]The reader can notice that the proposed protocol behaves correctly even if several mobile hosts located at different base stations initiate the consensus.

tions. Actually, ESTIMATE messages sent to the coordinator of a round and NEW_EST messages sent by the coordinator carry the current collection of values of the sender.

As in the protocol of Chandra and Toueg, 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 protocol of Chandra and Toueg 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), we remove partially those limitations in our protocol. A base station is allowed to change its initial proposed value if this value did 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 $\alpha + 1$ ESTIMATE messages to the coordinator. Furthermore, after receiving an NEW_EST message from the coordinator, a base station can still send a negative acknowledgment, meaning that it has not agreed with the proposed outcome result. More precisely, the value proposed by the coordinator is refused if it contains less than α mobile hosts values.

The parameter α is used to ensure that the decided outcome contains the values of at least α mobile hosts. Obviously, this parameter has to be greater or equal to 1. Assume that it is not the case: as a base station can propose a value equal to the empty set, the coordinator can receive a majority of estimates equal to the empty set and suggest this value to all the base stations by broadcasting a NEW_EST message. By setting α to 1, our solution can be used to solve the basic consensus problem. More generally, this protocol can be used as a building block to solve other agreement problems. In this case, the α parameter is defined by the upper layer application program when it requires that a minimal number of mobile hosts participate to the undergoing consensus protocol. Of course, the liveness property of our protocol is no more ensured if more than $m - \alpha$ mobile hosts are crashed forever.

As soon as a base station has gathered α values and sent a positive acknowledgment to a coordinator, its behavior is similar to the behavior of a process as defined by the original algorithm 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.

B. Protocol

The protocol is composed of three parts and each part contains a defined set of actions. 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, in one hand and the rest of base stations in the other hand. Sub-part B2 depicts the Chandra and Toueg protocol adapted to our mobile environment. Finally, the third part C of the protocol is the handoff protocol destined to handle mobility of hosts between different cells.

• Local Context of a Mobile Host

$Initial\_Value$: value provided by the application program.

• Local Context of a Base Station MSS_i

$Local\_MH_i$: Set of mobile hosts located in the cell managed by MSS_i. We assume that this list contains at least the identities of all the mobil hosts currently in the cell,
$Suspected_i$: List of bases stations suspected to be crashed. This list is managed by the local failure detector module of MSS_i,
r_i : current round of the Chandra-Toueg protocol executed by MSS_i,
$Phase_i$: phase number in a round,
$State_i$: state of MSS_i. $State_i$ is set to $decided$ if the consensus has terminated else $State_i$ is set to $undecided$,
ts_i : denotes the last round during which a new estimate sent by the coordinator has been accepted as the new value of V_i,
P_i : set of mobile hosts whose value are already known by MSS_i,
New_V_i : set of values collected by the base station,
V_i : last set of values proposed by MSS_i,
$Log_i[r]$: set of 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 (this set is empty 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 (this set is empty if MSS_i is not the coordinator of round r).

• Messages

INIT_1: this message is sent by a mobile host to its current base station to initiate a consensus,
INIT_2: this message is broadcast by a base station to inform the other base stations that a consensus is started,
INIT_3: this message is broadcast by a base station to query the initial values of all local mobile hosts that belong to group G,
PROPOSE(-): carries the value proposed by a mobile host to its local base station,
ESTIMATE(-): carries the estimate proposed by a base station to the current coordinator,
NEW_EST(-): carries the estimate proposed by the coordinator to the base stations,
DECIDE(-): carries the decided value
PA(-): positive acknowledgment,
NA(-): negative acknowledgment,
GUEST(-): when a mobile host enters a new cell, it sends a GUEST message to identify itself to the current base station,
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 cell.

In figure 2, the three actions performed by an arbitrary mobile host are:

(1) A mobile host executes this action when it receives a request from an upper application program to initiate a con-

% Mobile host h_k is located in MSS_i cell %
(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
% Let the value of variable Initial_Value be a value 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)

sensus.

(2) Message INIT_3 is sent to a mobile host either when its local base station is informed (on receipt of INIT_2 message) that a consensus has started or when the mobil host enters a cell managed by a base station which is not yet aware of its value and has not yet completed its collection of values. 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.

(3) When the consensus protocol terminates, the decided value is forwarded to each mobile host belonging to group G.

Actions of the protocol in figure 3, numbered from (4) to (14) have the following meaning:

(4) When a base station is asked by a mobile host to initiate a consensus, it sends an INIT_2 message to inform the other base stations that a consensus has 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 bases stations, all correct bases stations will be aware that a consensus has been initiated. Next, each base station starts to collect values from local mobile hosts, if any, and until the predicate $|P_i| \geq \alpha$ holds ($End\_collect_i = true$). Testing if $Phase_i = 0$ ensures the unicity of the undergoing consensus.

(5) Each base station gathers the values proposed by its local mobile hosts, while the minimal number of mobile hosts required by the application program to participate to the consensus is not reached ($|P_i| < \alpha$). Moreover, if MSS_i has already sent a first ESTIMATE message during the current round ($Phase_i > 1$), it proposes again its updated collection of values to the coordinator of the round. If the coordinator has not yet received a majority of ESTIMATE messages, it will consider this new message rather than the previous ones (action 7).

[5] $Log[r] := Log[r] \oplus (MSS_j, r, V_j, ts_j)$ 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.

$Phase_i := 0$; $End\_collect_i := false$;
$New\_V_i := \phi$; $V_i := \phi$; $P_i := \phi$;
$State_i := undecided$; $r_i := 0$; $ts_i := 0$;
$Majority := |G\_MSS|/2$;
for all r : $Log_i[r] := \phi$;
for all r : $NB\_PA_i[r] := 0$; $NB\_NA_i[r] := 0$;

cobegin
(4) || **Upon** receipt of INIT_1 $\vee$ INIT_2
if $Phase_i = 0$
then send INIT_2 **to** all $MSSs$ except MSS_i;
$Phase_i := 1$;
if $(Local\_MH_i \cap G\_MH) \neq \phi \wedge \neg End\_collect_i$
then W_Broadcast INIT_3
endif
endif

(5) || **Upon** receipt of PROPOSE(h_k, v_k)
if $\neg End\_collect_i$ **then** *% Values collection %*
$P_i := P_i \cup \{h_k\}$;
$New\_V_i := New\_V_i \cup \{v_k\}$;
if $|P_i| \geq \alpha$ **then** $End\_collect_i := true$ **endif**;
if $Phase_i > 1$ **then** *% MSS_i proposes a new estimate %*
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)
$Log_i[r] := Log_i[r] \oplus$ [5] $\{(MSS_j, r, V_j, ts_j)\}$;
if $\neg End\_collect_i$
then $P_i := P_i \cup P_j$;
$New\_V_i := New\_V_i \cup V_j$
if $|P_i| \geq \alpha$ **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)

(6) 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 ($Phase_i = 0$). To ensure that all correct processes decide, the message is also forwarded to all the other base station (reliable broadcast).

(7) The estimates sent during round r by base stations to MSS_i (MSS_i is necessarly the coordinator of round r), are gathered and logged in a local buffer $Log_i[r]$. r is the round number at the sending base station when the estimate was sent and is not necessarly equal to r_i. Each estimate is tagged with a timestamp ts_j representing the round number when MSS_j updated its estimate for the last time (see action 12). An arbitrary base station can propose multiple estimates during a round r_i but the coordinator keeps only the most recent estimate. While the collect is still possible ($End\_collect_i = false$), the coordinator updates the sets New_V_i and P_i each time it receives the local view of another base station.

(8) and **(9)** Acknowledgments counting.

```
(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;
                  send PA(MSS_i, r_i) to MSS_c;
                  End_collect_i := true
             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)

The Sub-part B2 of the protocol in figure 4 shows the Chandra and Toueg protocol adapted to our environment. The protocol proceeds in asynchronous rounds and each round is divided into four phases. It is assumed that all base stations have an a priori knowledge that during round r, the coordinator is base station $c = (r \ mod \ n) + 1$. Protocol computation is performed through the following actions:

(10) Each base station MSS_i sends its estimate V_i timestamped with the round number r_i. While MSS_i has not received a new estimate from the current coordinator (*i.e.* $ts_i = 0$), it can propose a more complete set of values to the coordinator. If MSS_i is the coordinator in round r_i then it moves to phase 2 and executes action (11) otherwise it moves to phase 3 and executes action (12) or action (13).

(11) Whenever a coordinator has gathered a majority of estimates, it selects one estimate from its local buffer $Log_i[r_i]$ 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_i of the current coordinator.

(12) and **(13)** While a base station MSS_i is waiting for a new estimate V_c, it queries its local failure detector if 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.

(14) If the coordinator gathers a majority of positive acknowledgments, the set V_c is locked and broadcast as the decided set of values to all base stations. Otherwise the coordinator moves to phase 1 and initiates the next round.

```
• 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 ∪ {h_k};
    send BEGIN_HANDOFF(h_k, MSS_i) to MSS_j;
    if Phase_i ≠ 0∧ ∄h_k ∈ P_i ∧ ¬End_collect_i
      then send INIT_3 to h_k
    endif;
    if (Phase_i = 0) ∧ (State_i := decided)
      then send 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)

As it is shown in figure 5, the handoff protocol is reduced. When a mobile host h_k moves from MSS_j cell to MSS_i cell, the handoff protocol execution is triggered. Mobile host h_k has to identify itself to its new base station by sending a message GUEST(h_k, MSS_j). On receiving this message, MSS_i learns that a new mobile host h_k, coming from MSS_j cell has entered in its cell. MSS_i informs MSS_j which removes h_k from the set of mobile hosts that reside in its cell and eventually transfers information about the last state of h_k to MSS_i. MSS_i queries h_k to send back its value, if a consensus has already started and the values collection is still possible.

C. 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 [4].

Theorem 1 *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 1 is valid if no two base stations 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 are 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. 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. $\square_{Theorem\ 1}$

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 are 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 execute 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.

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 2 *Every correct mobile host eventually decides some value.*

Proof There are two possible cases:

1. 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.

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

$\square_{Theorem\ 2}$

V. Conclusion

In an environment with m mobile hosts and n base stations, our 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 and the mobile hosts CPU time is low. The protocol is then more energy efficient. The protocol is also independent from the overall number of mobile hosts and all needed data structures are managed by the base stations. So, the protocol is scalable and cannot be affected by mobile hosts failures.
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 considered application.
Another interesting characteristic of the protocol is to allow a base station to propose continuous estimates while the number of participant mobile hosts is not reached. Hence, the protocol is then more suitable for a mobile environment due to the various operating modes of mobile hosts, particularly their ability to frequently connect and disconnect. Furthermore, if necessary, the protocol can be easily adapted so that all mobile hosts decide on the same value, including these crashed ones whenever they recover.

References

[1] Alagar S., Venkatesan S., Causally ordered message delivery in mobile systems, *in Proc. of Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., (dec.1994).

[2] Badache N., Mobility in Distributed Systems (in french), Technical Report #962, IRISA (Rennes), (October 1995).

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

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

[5] 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).

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

[7] 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, (1985).

[8] Guerraoui R. and Schiper A., Consensus service: a modular approach for building agreement protocols in distributed systems. *IEEE Int. Symp. on Fault-Tolerant Computing Systems (FTCS)*, pp:168–177, Jendai(Japan), (June 1996).

[9] Lampson B.W., How to build a highly available system using consensus.*10th International Workshop on Distributed Algorithms (WDAG'96)*, pp:9–11, (October 1996), Bologna, Italy.

[10] Krishna P., Vaidya N.H. and Pradhan D.K., Recovery in Distributed Mobile Environments. *IEEE Workshop on Advances in Parallel and Distributed Systems*, pp:83–88, (October 1993).

[11] Neiger G., Distributed consensus revisited. Technical Report GIT-CC-93 (october 1993), College of Computing, Georgia Institute of Technology, Atlanta.

[12] Pradhan D.K., Krishna P. and Vaidya N.H., Recoverable mobile environments: Design and trade-off analysis. *FTCS-26*, (June 1996).

[13] Schiper A., Early Consensus in an Asynchronous System with a Weak Failure Detector. *Distributed Computing*, **10**(3), pp:149–157, (1997).

[14] Hurfin M., Raynal M., Fast Asynchronous Consensus with a Weak Failure Detector. Technical Report #118, IRISA, Rennes, (July 1997)(submitted to publication).