

# Partitionable Group Membership for Mobile Ad Hoc Networks\*

Léon LIM and Denis CONAN  
Institut Mines-Télécom, Télécom SudParis  
CNRS UMR SAMOVAR,  
Évry, France

June 2013

## Abstract

Group membership is a fundamental building block that facilitates the development of fault-tolerant systems. The specification of group membership in partitionable systems has not yet reached the same level of maturity as in primary partition systems. Existing specifications do not satisfy the following two antagonistic requirements: *i*) the specification must be weak enough to be solvable (implementable); *ii*) it must be strong enough to simplify the design of fault-tolerant distributed applications in partitionable systems. In this article, we propose: *1*) a new distributed system model that takes into account the formation of dynamic paths, *2*) a specification of partitionable group membership for MANETS called  $\mathcal{PGM}$ , and *3*) an implementation of  $\mathcal{PGM}$  designed by adapting a well-known solution of primary partition group membership, namely Paxos. This results in the specification of an abortable consensus as the combination of two abstractions: An eventual  $\alpha$  partition-participant detector and an eventual register per partition which guarantee liveness and safety per partition, respectively. Then, partitionable group membership is solved by transformation into a sequence of abortable consensus.

**keywords:** Partitionable group membership, dynamic partitionable systems, MANETS, abortable consensus.

## 1 Introduction

Mobile Ad hoc NETWORKS (MANETS) are self-organising networks that lack a fixed infrastructure. The task of building distributed systems upon MANETS raises numerous challenges due to operational constraints: variability of wireless communication bandwidth and throughput due to unreliable physic layer, dynamicity of nodes arrivals and departures, heterogeneity of hand-held devices in terms of battery power and bandwidth capability, mobility behaviour of nomad users, etc. As a consequence, topology changes occur both rapidly and unexpectedly and nodes (processes) can dynamically enter and leave the system. In other words, a distributed system built over MANETS is partitionable. According to the CAP (*Consistency, Availability and Partition-tolerance*) theorem, it is impossible for an synchronous distributed system service to provide C, A and P at the same time [Brewer, 2000, Gilbert and Lynch, 2012]: consistency implies that each response to a request is atomic; availability requires that each node receiving a request must respond; partition-tolerance reflects the tolerance of message losses. If the system is partially synchronous then two of the three desirable properties can be achieved. Since

---

\*Submitted for publication.

network partitioning is an intrinsic characteristic in MANETs, P must be provided while ensuring some trade-off between C and A. In other words, network partitioning may result in a degradation of the services, but not necessarily in their unavailability: partitioned groups must continue to operate as autonomous distributed systems. Basically, there is a need to consider a mechanism (or middleware service) that manages partitioned groups explicitly in order to mitigate the effects of network partitions on C and A. Therefore, we tackle in this article the problem of partitionable group membership for MANETs.

Group membership is a middleware service that maintains views of the membership of the group at each process. A view is a list of processes (the members) with an identifier for unique identification. In the literature, two types of group membership services have emerged: primary partition and partitionable [Chockler et al., 2001]. *Primary partition* group membership maintains a single agreed view of the core cluster of processes —*i.e.*, the so-called primary partition. [Guerraoui and Schiper, 2001] shows that primary partition membership can be solved by a sequence of consensus, where each consensus is executed by the processes in the current view and the decision returned by the consensus is a set of processes, and these processes are the members of the next view. *Partitionable group membership* maintains all partitions uniformly. By allowing concurrent views, partitionable group membership does not require strong agreement as the one in primary partition group membership [Chandra et al., 1996], but its specification faces two antagonistic requirements: *i)* the specification must be weak enough to be implementable; *ii)* it must be strong enough to simplify the design of fault-tolerant distributed applications in partitionable systems. Collaborative applications [Birman et al., 1999], resource allocation management [Anker et al., 1999], and distributed monitoring [Murray, 2005] are examples of applications that support permanent partitioning and thus go on running on multiple partitions.

Several partitionable group membership specifications have been proposed and surveyed in [Anceaume et al., 1995, Babaoğlu et al., 2001, Chockler et al., 2001, Pleish et al., 2008]. Two of the prominent specifications are in [Babaoğlu et al., 2001] and [Chockler et al., 2001]. They sketch the two categories of partitionable group membership specifications which differ about their liveness property: *i)* liveness must hold only in completely stable partitions [Chockler et al., 2001], and *ii)* liveness must be ensured in every partition [Babaoğlu et al., 2001]. However, one of the two requirements cited above is not satisfied in these specifications: the specification in [Chockler et al., 2001] requires a strong stability condition that can be satisfied by a trivial but useless implementation; the specification in [Babaoğlu et al., 2001] requires a weak stability condition that cannot be implemented without assuming a model that is somewhat in contradiction with dynamic systems. As a consequence, we focus in this article on the specification of partitionable group membership that is implementable and strong enough.

Since consensus can be used to solve primary partition group membership, we argue that another type of consensus may be used to solve partitionable group membership. One of the well-known consensus protocols in primary partition systems is the Synod algorithm of Paxos [Lamport, 1998, Lamport, 2001]. Since Paxos is adaptable and makes use of a sequence consensus natively, we propose a solution to the partitionable group membership problem by adapting Paxos.

The contribution presented in this article is threefold. Firstly, we define a distributed system model for MANETs with a weak stability condition based upon the application-dependent parameter named  $\alpha$ , which is a threshold value used to capture the liveness property of a partition:  $\alpha$  stable processes are required to execute distributed computations in a partition. We also define a heartbeat-counter-based stability criterion which is a parameter that is used by processes to

determine the most stable nodes among mutually reachable ones. Secondly, we adapt the Paxos protocol by following the methods proposed in [Boichat et al., 2003a, Boichat et al., 2003b]. This results in a specification of a form of consensus, called abortable consensus ( $\mathcal{AC}$ ).  $\mathcal{AC}$  is a combination of two abstractions: An eventual  $\alpha$  partition-participant detector ( $\diamond\mathcal{PPD}$ ) and an eventual register per partition ( $\diamond\mathcal{RPP}$ ). For short,  $\diamond\mathcal{PPD}$  is specified to abstract liveness in a partition whereas  $\diamond\mathcal{RPP}$  encapsulates safety in the same partition. Then, the partitionable group membership problem is solved by a transformation into a sequence of  $\mathcal{AC}$ . Thirdly, we provide algorithms that implement all these abstractions, and proofs of the algorithms (presented in A and B).

The rest of the article is organised as follows. We define a dynamic distributed system model for MANETs in three steps: We present first elements in Section 2; we summarise in Section 3 the problems in existing specifications by focusing on the specifications in [Chockler et al., 2001] and [Babaoğlu et al., 2001]; and, we enrich our system model with fairness, reachability and timeless properties in Section 4. Afterwards, we specify abortable-consensus-based partitionable group membership in Section 5. Then, in Section 6, we implement the specification  $\mathcal{PGM}$ , including  $\mathcal{AC}$  and its two modules  $\diamond\mathcal{PPD}$  and  $\diamond\mathcal{RPP}$ . Finally, we discuss some related works and conclude the article in Sections 7 and 8.

## 2 Distributed system model

In this section, we define the first elements of our distributed system model by characterising the dynamicity of MANETs and presenting preliminary definitions.

### 2.1 Crash-recovery and infinite arrival models

We consider a dynamic distributed system composed of infinitely many mobile nodes. We assume that nodes are uniquely identified, and consider one process per node. Thus, the system consists of an infinite countable set of processes (nodes)  $\mathbb{P} = \{\dots p_i, p_j, p_k \dots\}$ . A process is *correct* in an execution if it does not fail in that execution. We consider the crash-recovery model in which each process has, in addition to its regular volatile memory, a stable storage which allows the process to store part or all of its state via the primitive *store()*. This allows the process to restart upon failure with the same identifier by recovering its state via the primitive *recover()*. We assume the *execution integrity* property stipulating that the recovery of a process is necessarily preceded by its failure.

Nodes can dynamically enter the system or leave it by crashing, recovering, disconnecting, or reconnecting. By definition [Aguilera, 2004, Merritt and Taubenfeld, 2000], the number of processes that have joined the system minus the number of departures is called *concurrency*. We then consider the *infinite arrival model with bounded concurrency* investigated in [Aguilera, 2004, Merritt and Taubenfeld, 2000]: in any bounded period of time, only finitely many nodes take steps; the total number of nodes in a single execution may grow to infinity as time passes; however, each execution has a maximum concurrency that is finite but unknown. In addition, processes do not know  $\mathbb{P}$  —*i.e.*, the processes in  $\mathbb{P}$  do not necessarily know each other.

### 2.2 Asynchronous event-based composition model

We consider the *asynchronous event-based composition model* [Guerraoui and Rodrigues, 2006] to specify interfaces and properties of distributed algorithms and systems. In this model, each process is composed of a set of software modules called components. Each primitive or composite

component is identified by its name and is characterised by an interface presenting different types of events that the component can accept and produce. Distributed algorithms are typically made of a collection of at least one components per process that are supposed to satisfy some properties. We consider  $\mathbb{E}$  to be the set of possible events including at least the events `store()`, `crash()` and `recover()`. Thereafter,  $\mathbb{E}$  is enriched with other events related to group membership and MANETs.

### 2.3 Execution, global history and causal order

The execution of a process is modelled by a sequence of events. The local history of process  $p$  during an execution is a sequence of events or absences of events  $h_p = (e_p^1 e_p^2 e_p^3 \dots)$ , where an absence is denoted by  $\epsilon$  and  $e_p^i$  corresponds to the  $i$ th event of process  $p$ . The index  $p$  and the exponent  $i$  are omitted when the context is clear. In addition, we consider the existence of a discrete global clock that is not accessible to the processes. We take the range  $\mathbb{T}$  of the clock's tick to be the set of natural numbers  $\mathbb{N}$ . The global history of an execution is a function  $H : \mathbb{P} \times \mathbb{T} \rightarrow \mathbb{E} \cup \epsilon$ . If  $p$  executes an event  $e \in \mathbb{E}$  at instant  $t$  then  $H(p, t) = e$ ;  $H(p, t) = \epsilon$  meaning that  $p$  does not execute any event at instant  $t$ . Let  $I \subseteq \mathbb{T}$  be an interval, we write  $e \in H(p, I)$  if  $p$  executes some event  $e$  in  $I$  —i.e.,  $\exists t \in I : H(p, t) = e$ . We also consider the “happened-before” relation between events as defined in [Lamport, 1978].

### 2.4 MANET links

Processes communicate by exchanging messages chosen among the set of possible messages  $\mathbb{M}$ . We assume the MANET communication model in which nodes do not send point-to-point messages but broadcast messages that are received by those nodes that are in their transmission range. If a process  $q$  is within the transmission range of a process  $p$ , we say that there is a link between  $p$  and  $q$  denoted  $p \rightsquigarrow q$ . Links are unidirectional and the network is not necessarily completely connected. Messages are uniquely identified and there is no upper bound on message transmission delays. We assume that  $q$  receives a message  $m$  from  $p$  at most once (*no duplication*) and only if  $p$  previously broadcast  $m$  (*no creation*). We define the event `broadcastnbq(p, m)` which corresponds to the broadcast of  $m$  by  $p$  to its neighbours. By way of exception, the specifications of [Chockler et al., 2001] and [Babaoğlu et al., 2001] introduce the event `send(p, m)` which corresponds to the sending of  $m$  by  $p$ .  $p$  receives a message  $m$  by executing the event `receive(p, m)`.

### 2.5 Groups and views

We adopt the terminology of [Chockler et al., 2001]. A group is a set of processes which are members of the group. A view is a pair  $v = (set, id) \in \mathbb{V}$ , where  $set$  is a set of processes identified by a unique identifier  $id$  and  $\mathbb{V}$  is the set of possible views. Membership reflects the process' current perception of the group. The event by which a process  $p$  considers a new view  $v$  is called view installation and is noted `view_change(p, v, TS)`, where  $p$  is a process,  $v$  is a view, and  $TS$  is a transitional set which corresponds to a sub-set of processes of the intersection between members of the current view and members of the new view. The execution of most of the events occurs in the context of a view. The view in which an event is executed is defined by the following function `viewof` :  $\mathbb{E} \rightarrow \mathbb{V} \cup \perp$ . In this definition,  $\perp \notin \mathbb{V}$  means that the event is not executed in a view —e.g., during the initialisation or upon a recovery.

**Definition 1.** *viewof* [Chockler et al., 2001]. *The view of an event  $e$  executed by process  $p$  is*

the view delivered to  $p$  in the event  $e'' = \text{view\_change}()$  which precedes  $e$  and such that no event  $e'$  (which corresponds to  $\text{view\_change}()$  or  $\text{crash}()$ ) at  $p$  between  $e''$  and  $e$ . The view is  $\perp$  if there does not exist such an event  $e''$ .

### 3 Problems in existing specifications

In this section, we present basic safety and liveness properties of existing specifications of partitionable group membership and their problems.

#### 3.1 Safety properties

The three following safety properties provide basic guarantees about the installation of views. They are satisfied by [Chockler et al., 2001] and [Babaoglu et al., 2001] and most of partitionable group membership specifications.

The self inclusion property requires that partitionable group membership must inform a process about only views which it is a member of. A view reflects process' current perception of the group —*i.e.*, a set of processes with which it can communicate. A process is always able to communicate with itself unless it fails.

**Property 1.** Self inclusion. *If process  $p$  installs view  $v$ , then  $p$  is a member of  $v$ .*

The local monotonicity property requires an increasing order of view identifiers. Two consequences follow: 1) a process installs a view at most once, and 2) if any two processes install the same any two views, then they install these views in the same order. A process that recovers installs the last view stored in its local stable memory.

**Property 2.** Local monotonicity. *If process  $p$  installs view  $v$  after installing view  $v'$ , then the identifier of  $v$  is strictly greater than that of  $v'$ .*

The initial view event property states that the events  $\text{send}()$ ,  $\text{receive}()$ , or  $\text{broadcast}_{nbg}()$  do not appear before the first event  $\text{view\_change}()$ .

**Property 3.** Initial view event. *Each application event  $\text{send}()$ ,  $\text{receive}()$  and  $\text{broadcast}_{nbg}()$  is executed in some view.*

#### 3.2 Liveness properties

Satisfying liveness raises the issue of detecting currently active and connected processes [Chockler et al., 2001]. Since we consider partitionable systems, we introduce a mechanism for detecting the stability of nodes in groups. Nodes may detect false positive stable nodes due to the duration of these detections. Ideally, the partitionable group membership service must provide precise information (views) that correspond to situations of processes that are currently active and that can communicate with each other. Such an ideal behaviour cannot be guaranteed in all the executions, but can only be achieved in some conditions. These conditions correspond to the stability of the network that is external to the implementation of the service. Therefore, the service may deliver different views to members of a group during periods of time with unstable network conditions. What matters is that the same view be delivered to group's members once group's membership stabilises.

The specifications in [Chockler et al., 2001] and [Babaoglu et al., 2001] make use of the failure detector abstraction by extending the *eventually perfect failure detector*

$\diamond\mathcal{P}$  [Chandra and Toueg, 1996] in order to define the liveness properties. This leads to two stability conditions: 1) strong stability [Chockler et al., 2001] in which liveness properties are guaranteed only in eventually completely stable partitions in which all the links are eventually up, and 2) weak stability [Babaoğlu et al., 2001] in which liveness properties are guaranteed in all the partitions in which communication links present some fairness.

**Strong stability condition** In [Chockler et al., 2001], processes are linked by paths made of unidirectional logical links. Links may be up or down. When a link is down, it cannot transport any message —*i.e.*, each transported message is lost. If a link is up and remains forever after instant  $t$  then each message transported through this link after  $t$  is eventually received by the destination node. An eventually completely stable partition is then defined as a set of processes  $S$  such that the processes in  $S$  are eventually active and connected to each other through up links and the links from any process in  $S$  to any process outside  $S$  are down. In these strong stability conditions, the failure detector that is denoted  $\diamond\mathcal{P}$  — *like* in [Chockler et al., 2001] behaves like  $\diamond\mathcal{P}$ . Thus, the stability of the partition containing mutually connected nodes is required eventually forever. However, in practice, this stability is required for a period of time that lasts long enough so that the output of the failure detection module stabilises. The liveness property of partitionable group membership ensures that eventually each process installs a view  $v$  which contains only members of the completely stable partition ( $v.members = S$ ). This view is called a “precise view”.

**Weak stability condition** In [Babaoğlu et al., 2001], a reachability property between processes is defined in order to express the ability of processes to communicate with each other: If process  $p$  sends a message  $m$  to process  $q$  at instant  $t$  then  $q$  receives  $m$  if and only if  $q$  is reachable by  $p$  at instant  $t$ . The adapted version of  $\diamond\mathcal{P}$ , denoted  $\diamond\tilde{\mathcal{P}}$  in [Babaoğlu et al., 2001], is used to specify liveness in all the partitions, and not only in eventually completely stable partitions. For each request by a process  $p$ ,  $\diamond\tilde{\mathcal{P}}$  returns a set of processes which are reachable by  $p$ . To exclude useless solutions in which no messages are delivered, [Babaoğlu et al., 2001] assumes that *i)* a correct process always delivers its own messages and *ii)* if a correct process  $p$  delivers message  $m$  in view  $v$  that includes some other process  $q$  and if  $q$  never delivers  $m$ , then  $p$  eventually installs a new view  $v'$  as the immediate successor to  $v$  (*liveness of message delivery*). Therefore, the liveness property of partitionable group membership can be satisfied in partitions in which reachability relation is persistent. If this weak stability condition captured by  $\diamond\tilde{\mathcal{P}}$  eventually holds, then the partitionable group membership service outputs the value returned by  $\diamond\tilde{\mathcal{P}}$ .

### 3.3 Problems in the specifications

From a practical point of view, the sole usage of partitionable group membership for building distributed applications is limited [Babaoğlu et al., 1998]: View-aware applications require the cooperation of processes in the same group that is facilitated by a reliable multicast service. A basic and well-known property of reliable multicast is *virtual synchrony*, which can be used in conjunction with either the *agreement on successors* property or the *transitional set* property. The virtual synchrony property imposes that two processes which install two consecutive views  $v'$  et  $v$  deliver the same set of messages in  $v'$  [Chockler et al., 2001, Babaoğlu et al., 2001]. The following paragraphs present the problem of capricious view that is present in the specification of [Chockler et al., 2001] and the problem of the definition of fair links in the specification

of [Babaoğlu et al., 2001], thus showing the necessity to propose a new distributed system model and a new specification of partitionable group membership.

**Problem of capricious views** Each view installation must be justified and must reflect system changes as a result of events which occurred in the system (such as real or suspected failure, or request to join or leave the group) [Anceaume et al., 1995]. In particular, installation of a capricious view —*i.e.*, installation of a new view at any instant and without any reason— should not be allowed because it allows for instance arbitrary deletions of correct and mutually connected processes. Now, we follow the argument of [Pleish et al., 2008] to show that the specification of [Chockler et al., 2001] does not meet the requirement stipulating that the specification must be strong enough to simplify the design of fault-tolerant distributed applications in partitionable systems. In [Pleish et al., 2008], the authors provide two trivial but useless algorithms that satisfy the specification of [Chockler et al., 2001] (including self-inclusion, local monotonicity, initial view event and virtual synchrony). In these implementations, a singleton view containing just  $p$  —*i.e.*, a *capricious view*— is installed at  $p$  before every installation of a non-singleton view. Clearly, these implementations do not help  $p$  to get any information on other processes. We therefore consider that the specification of partitionable group membership has to avoid capricious view installation in order to be strong enough: The removal or inclusion of a process is allowed only if this process was suspected to leave or join the partition, respectively; and, if such an event occurs then a new view is eventually installed to reflect the event.

**Problem in the definition of fair links** The problem of the specification of [Babaoğlu et al., 2001] stems from the fact that two processes which are intermittently mutually reachable become unreachable precisely at those times when they attempt to communicate. To avoid this adverse scenarios, [Babaoğlu et al., 2001] enriches the system model with the notion of fair link which is defined as follows [Babaoğlu et al., 2001]: “*Let  $p$  and  $q$  be two processes that are not permanently unreachable from each other. If  $p$  sends an unbound number of messages to  $q$ , then  $q$  will receive an unbound number of these messages*”. However, as stated by [Pleish et al., 2008], the specification in [Babaoğlu et al., 2001] is based on a time-dependent property (Definition of reachability relation) in a system model comprising a time-independent property (Definition of fair link). In [Babaoğlu et al., 2001], the notion of reachability is not time invariant: Process  $q$  can be reachable by  $p$  at instant  $t'$ , but may not be reachable by  $p$  at instant  $t > t'$ . As the consequence, [Babaoğlu et al., 2001] assumes a model that is somewhat in contradiction with dynamic systems. In addition, we believe that communication links enriched with the lossy property of fair links is not enough to capture the dynamicity of MANETs: links are created dynamically and may not last long enough to tolerate infinite message losses that are allowed by fair links. Thus, we argue that another kind of links must be considered in order to model wireless, dynamic, and unstable links of MANETs.

## 4 Enriched distributed system model

In this section, we propose a model that captures the dynamic creation of communication links, and define the stability property and the stability condition of partitions in MANETs.

## 4.1 Fairness, reachability and timeliness

We distinguish three kinds of links: 1) eventually down, 2) eventually up and 3) forever intermittently up. An *eventually up* link eventually transports messages without losing any of them. An *eventually down* link eventually stops transporting messages. A *forever intermittently up* link can arbitrarily lose messages it transports. As stated in [Babaoglu et al., 2001], forever intermittently links are the root of the instability of the system. To avoid bad scenarios, we consider the lossy property of links by defining a fair link of MANETs as follows.

**Definition 2.** Fair link. *The link  $p \rightsquigarrow q$  is fair if, when  $p$  broadcasts a message  $m$  to  $q$  an infinite number of times,  $q$  receives  $m$  from  $p$  an infinite number of times.*

Notice that eventually up links and forever intermittently up links are both captured into fair links. Thus, we do not distinguish them in the sequel. A sequence of processes  $(p_1 p_2 \dots p_n)$  in which the links  $p_1 \rightsquigarrow p_2, \dots, p_{n-1} \rightsquigarrow p_n$  are fair is called a fair path from  $p_1$  to  $p_n$  denoted by  $\text{FAIR}(p_1 p_2 \dots p_n)$ .

By definition, a fair link can lose messages due to communication failures. Our *reachability* relation captures these communication failures and is defined as suggested in [Pleish et al., 2008].

**Definition 3.** Reachability. *Given two processes  $p$  and  $q$ ,  $q$  is reachable from  $p$  if and only if there is a fair path from  $p$  to  $q$ . We denote this relation of reachability of  $q$  from  $p$  as  $p \rightarrow q$ .*

If process  $q$  is reachable from process  $p$ , and vice-versa, we write  $p \rightleftharpoons q$  and we say that  $p$  and  $q$  are mutually reachable. Therefore, both the definitions of fair link and reachability are time independent.

Since fair links may suffer from arbitrary delays and/or losses such that there exists no “finite stable period” in which processes can communicate “fast enough” in order complete useful computation, we define the concept of SADDM (Simple Average Delayed/Dropped of a Message) links in which communication delays between two processes in a fair link are bounded during stable periods. This concept of SADDM links is inspired from the notion of Average Delayed/Dropped channel [Sastry and Pike, 2007]. A SADDM link allows messages to be lost or arbitrarily delayed, but guarantees that some subset of them are received in a timely manner and that such messages are not too sparsely distributed in time. A SADDM link is defined as follows.

**Definition 4.** SADDM Link. *Let  $\delta$  and  $\beta$  be two constants and let  $\phi$  be a finite time interval during which  $p$  broadcasts a message  $m$  to  $q$  at least  $\beta$  times. The link  $p \rightsquigarrow q$  is a SADDM link if  $q$  receives at least one of these messages in at most  $\delta$  seconds.*

A sequence of processes  $(p_1 \dots p_n)$  is a SADDM path in the interval  $\Phi$  if  $\forall i, j \in [1, n - 1] : i \neq j \implies p_i \neq p_j$  and the link  $p_i \rightsquigarrow p_{i+1}$  is a SADDM link in  $\phi_{i,i+1} \subseteq \Phi$ . A SADDM path from  $p_1$  to  $p_n$  is denoted by  $\text{SADDM}(p_1 \dots p_n)$  with respect to an interval  $\Delta$  which is omitted when it is unambiguous.

## 4.2 Stable partition, stability condition and stability criterion

A partition is stable during a period when all the processes in that partition can communicate with each other using SADDM paths during that period. So far, we have not defined such a period. To do so, as in [Mostefaoui et al., 2005], let us define a time interval  $[t_b, t_e]$  as being a period.  $t_b$  and  $t_e$  are defined by the application processes:  $t_b$  is the beginning time of the application whereas  $t_e$  is its ending time. To simplify the presentation, we consider only one

period in the rest of the article. During a period, the stable property is a property that remains true once it holds, that is to say it holds after the *local stabilisation time*  $LST_p$  which is unknown to the processes. A stable partition associated to some process  $p$  is then denoted by  $\diamond PART_p$  and is defined as follows.

**Definition 5.** *Stable Partition.* *The stable partition of process  $p$  is the set of processes such that there exists a time  $LST_p$  after which, for each  $q \in \diamond PART_p$ ,  $SADDM(p \dots q)$  and  $SADDM(q \dots p)$  exist.*

In addition, we say that a process  $q$  is *stable* in the context of  $p$ 's stable partition if  $q \in \diamond PART_p$ . This means that  $q$  is stable in  $p$ 's partition at some unknown time  $t \geq LST_p$ .

Since there may be stable processes and unstable processes that coexist in the same partition, progress is only desirable for stable ones. Hence, we claim that useful computations should be executed only by a set of at least  $\alpha$  stable processes. The value of  $\alpha$  is specified as a requirement by the application. It is the responsibility of the application to choose a suitable value of  $\alpha$  —*i.e.*, the minimum number of participants to the application computations. Therefore, the stability condition associated to a process  $p$  is defined as follows.

**Definition 6.** *Stability condition.*  $|\diamond PART_p| \geq \alpha_p$ .

Stable processes in the same partition are always able to communicate with each other after  $LST_p$  through SADDM paths. Unfortunately, processes cannot know  $LST_p$ . We claim that, since nodes are heterogeneous, only a subset of mutually reachable nodes should be selected by some *stability criterion* to form a partition that may possibly satisfy the stability condition. A stability criterion is a parameter that is used to determine which nodes are the most stable ones.

We propose that nodes detect the current processes in their partitions —*i.e.* the ones that are currently mutually reachable— by periodically broadcasting heartbeat messages to their neighbours. Then, we specify a time-based stability criterion  $minhb_p \leq hb_p^q \leq maxhb_p$ , with  $minhb_p \geq 1$  and  $maxhb_p \geq minhb_p$  being two constants, and with  $hb_p^q$  being a function that depends on the number of heartbeat messages received by  $p$  from  $q$  —*i.e.*,  $hb_p^q$  increases if  $q$  is present in  $p$ 's partition and decreases otherwise.  $q$  is marked as stable by  $p$  if  $hb_p^q \geq minhb_p$ , and  $q$  is removed from  $p$ 's set of stable processes if  $hb_p^q = 0$  —*i.e.*,  $p$  does not receive any heartbeat from  $q$  anymore.  $maxhb_p$  is the maximal value that a heartbeat counter can reach so that the heartbeat counter does not increase indefinitely and the detection time of a departure is not proportional to the duration of the presence of the process in the partition. With this stability criterion while tolerating sporadic disconnections, a node is eliminated from participating if it disappears from the partition.

### 4.3 Illustrative example

Figure 1 depicts an example of a distributed system configuration during a time period with stable partitions and their stability condition. Dashed circles represent the transmission range of processes. Solid arrows correspond to SADDM links and dashed arrows correspond to links that do not have the SADDM property. Black disks represent unstable nodes whereas white disks depict stable processes. Each stable partition is enclosed by a solid circle, and two stable partitions do not intersect. In Figure 1, there are eventually five stable partitions  $\diamond PART_o$ ,  $\diamond PART_p$ ,  $\diamond PART_w$ ,  $\diamond PART_a$  and  $\diamond PART_x$  with their value of  $\alpha$  equals to 1, 4, 2, 3 and 2, respectively.

Observe that stable partitions are not necessarily isolated from the other nodes of the network: Links from processes in a stable partition to processes outside the partition are not

necessarily down. For instance, process  $u$  in  $\diamond PART_w$  can receive messages broadcast by processes in  $\diamond PART_p$  in a timely manner through some SADDM paths, but processes in  $\diamond PART_p$  cannot receive messages broadcast by  $u$  in a timely manner because there is no SADDM path from any process in  $\diamond PART_w$  to  $q$ . Therefore,  $u$  is unstable in the context of  $\diamond PART_p$  and stable in the context of  $\diamond PART_w$ .

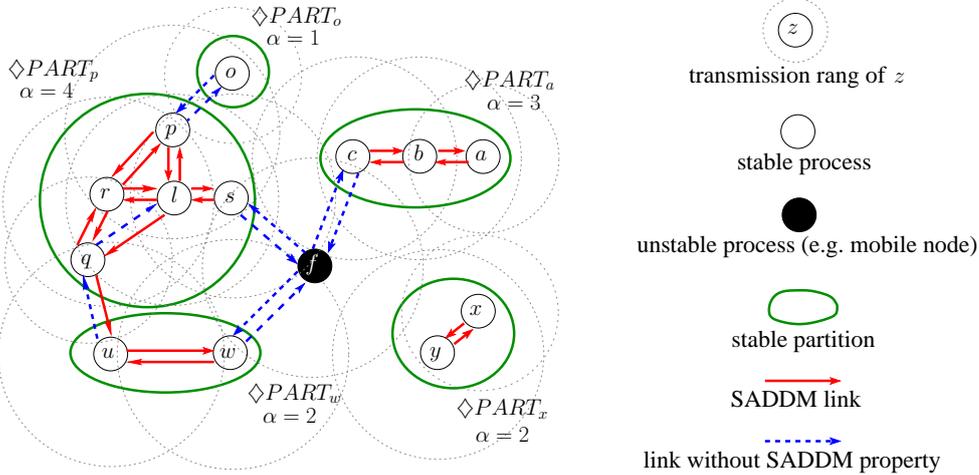


Figure 1: Stable partitions and their stability condition during a period of time.

## 5 Abortable consensus-based partitionable group membership

We now present the architecture and the specification of the abortable consensus-based partitionable group membership.

### 5.1 Architecture

The architecture of partitionable group membership  $\mathcal{PGM}$  is based upon abortable consensus  $\mathcal{AC}$  (cf. Figure 2).  $\mathcal{AC}$  is a combination of two modules: The eventual  $\alpha$  partition-participant detector  $\diamond PPD$  and the eventual register per partition  $\diamond RPP$ .  $\diamond PPD$  and  $\diamond RPP$  correspond to the adapted versions of the eventual leader  $\diamond Leader$  and the eventual register  $\diamond Register$  of [Boichat et al., 2003a], respectively. The role of  $\diamond PPD$  is to construct the set of stable processes  $\alpha Set$  and elect a leader among them. The role of  $\diamond RPP$  is to propose and decide a value among processes in  $\alpha Set$ .  $\diamond RPP$  uses the retransmission module to deal with message losses in SADDM paths.  $\diamond RPP$  also uses  $\diamond PPD$  to decide if a proposed value must be aborted when participant nodes disappear from the partition.

It is the responsibility of the application to propose a new view if needed. Thus, the application layer decides to include or exclude processes by proposing to install a new view by using  $\mathcal{PGM}$ .  $\mathcal{PGM}$  uses the retransmission module to send and receive new views similarly to  $\diamond RPP$ .  $\mathcal{PGM}$  notifies the application that its request can not be satisfied when the installation proposal of a view fails —*i.e.* consensus is abandoned. The set of members of the potential new view must be included in  $\alpha Set$ . To this means,  $\diamond PPD$  notifies the application when it detects a change in the composition of  $\alpha Set$ ; for the sake of simplicity, this notification is supposed to be done implicitly.

## 5.2 Specification of the partitionable group membership

In addition to the basic safety properties self inclusion, local monotonicity and initial view event (cf. Section 3.1),  $\mathcal{PGM}$  satisfies the following safety and liveness properties.

**Property 4.**  $\mathcal{PGM}$ -Validity. *If process  $p$  installs view  $v = (vset, vid)$ , then  $v' = (vset, vid')$  was proposed by a process  $q$  (possibly  $p$ ) in  $v.members$  and such that  $|v.members| \geq \alpha_p$ .*

$\mathcal{PGM}$ -Validity stipulates that the act of installing a view must reflect events that occurred in the environment.

**Property 5.**  $\mathcal{PGM}$ -Agreement on final view. *If there exists a set of stable processes  $S \subseteq \alpha Set \subseteq \diamond PART_p$  that wish to install view  $v_f$  with  $v_f.members = S$ , then all the processes in  $S$  eventually install the same final view  $v_f$ .*

$\mathcal{PGM}$ -Agreement on final view ensures agreement on the installed view among a set of processes  $S$  in a partition.  $\mathcal{PGM}$  must eventually provide the same view to all the processes in  $S$  when the partition stabilises. The members of the final view is included into or equal to the set  $\alpha Set$  that is provided by  $\diamond PPD$ . The last view is obtained if the partition stabilises and if eventually no processes in  $S$  propose any view.

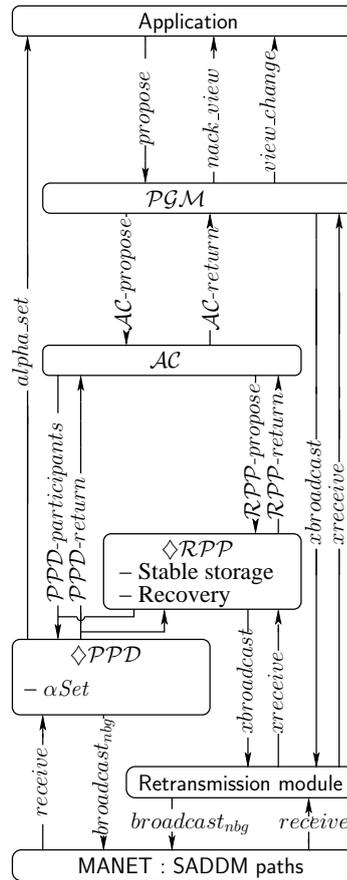


Figure 2: Architecture of partitionable group membership.

It is worth remarking that  $\mathcal{PGM}$ -Agreement on final view guarantees liveness of a partition even when the partition is not completely stable. Recall that stable nodes in a partition are not

necessary isolated from the other nodes in the network —*i.e.*, stable and unstable nodes may coexist in a network partition. The particularity of our approach concerns the detection of the sets  $\alpha Set$  composed of at least  $\alpha$  stable processes. Since view members must form a disjoint subset of nodes  $S \subseteq \alpha Set$  with  $\alpha \leq |S| \leq |\alpha Set|$ , installation of a capricious view is avoided after the local stabilisation time of the partition —*i.e.*, when the stability condition is satisfied

### 5.3 Specification of the abortable consensus

By analogy to the consensus protocol Synod of Paxos, each process in the system is not supposed to propose a value: Only the processes that detect a set constituted of at least  $\alpha$  stable processes and that believe themselves to be the leader of this set propose values. A process  $p$  proposes  $vset$  as the set of the potential next view with the identifier  $vid$  by invoking the primitive  $\mathcal{AC}\text{-propose}(vset, vid)$ . The invocation terminates by returning a value (primitive  $\mathcal{AC}\text{-return}$ ). Returned values  $vset$  and  $vid$  may be  $\perp$ , then meaning that the consensus was aborted. If  $vset \neq \perp \wedge vid \neq \perp$ , then a consensus is said to be reached. Abortable consensus satisfies the three following properties.

**Property 6.** *AC-Termination.* In a partition, let  $p$  be a process which proposes a value. If there exists a set  $\alpha Set$  composed of at least  $\alpha$  processes (including  $p$ ), then  $p$  eventually decide. Otherwise,  $p$  abandons.

**Property 7.** *AC-Agreement.* In a partition, two processes in the set of stable processes  $\alpha Set$  do not eventually decide different values.

**Property 8.** *AC-Validity.* In a partition, if a process  $p$  decides value  $val = (vset, vid)$ , then  $val' = (vset, vid')$  was proposed by a process  $q$  (possibly  $p$ ) in  $vset$ .

### 5.4 Specification of the eventual $\alpha$ partition-participant detector

An *eventual  $\alpha$  partition-participant detector* is a distributed oracle that detects the set of stable processes  $\alpha Set$  in a partition. The processes in  $\alpha Set$  are chosen according to the stability criterion  $minhb_p \leq hb_p^q \leq maxhb_p$ .  $\diamond PPD$  also eventually elects a unique leader among  $\alpha Set$ .  $\alpha Set$  at  $p$  eventually stops changing but there is no knowledge of when the unique leader is elected. Several processes may think they are leaders. However, a unique leader is eventually elected when the stability condition  $|\diamond PART_p| \geq \alpha_p$  holds after some local stabilisation time.  $\diamond PPD$  satisfies the following properties.

**Property 9.** *Eventual  $\alpha Set$  stability.* There is a time after which any two processes of a stable set of processes  $\alpha Set$  have the same  $\alpha Set$ .

**Property 10.** *Eventual leader agreement.* There is a time after which all the processes in a stable set of processes  $\alpha Set$  elect the same process in  $\alpha Set$  as the leader.

### 5.5 Specification of the retransmission module

The retransmission module deals with losses of messages that are broadcast through SADDM paths. A process  $p$  executes the primitive  $xbroadcast(m)$  or  $xreceive(m)$  to broadcast or receive message  $m$ , respectively. We define the retransmission module, which satisfies the following property.

**Property 11.** Reliable broadcast to a set of stable nodes. *Let  $p$  be a stable process and  $dest$  be a set of stable processes included in  $\diamond PART_p$ . If  $p$  broadcast a message  $m$   $\beta^n$  times to  $dest$ , then the message  $m$  originally broadcast by  $p$  is received by each process  $q \in dest$  in at most  $\beta^n \eta + n\delta$  seconds, where  $\eta$  is the maximal period of time that delimits two consecutive broadcasts and  $n$  is the length of the longest SADDM path between  $p$  and  $r \in dest$ .*

To satisfy the reliable broadcast property, all the messages that need be retransmitted must be not too sparsely distributed in time. Recall that  $\beta$  and  $\delta$  are constants that are used in the definition of a SADDM link.

## 5.6 Specification of the eventual register per partition

As in [Boichat et al., 2003a],  $\diamond RPP$  materialises a stable shared memory. However, the eventual register per partition is not necessarily shared by all the processes of the system (including a majority of correct processes), but by the set of processes in partition composed of at least  $\alpha$  stable processes. In addition, the tentative of writing a value in the register may fail in two case: 1) in case of contention or 2) if the stability condition is not satisfied. The first case corresponds to concurrent propositions and is the same as in [Boichat et al., 2003a]: A proposer may abandon a proposition if there exists another proposer in the same partition which began to propose a value concurrently. In the second case, the proposer does not only abandon its proposition, but also the consensus instance, because there does not exist  $\alpha$  stable processes in the partition. In case of success during a stable period of time, the execution model may execute as follows: When the stability condition is satisfied, and if only one stable process keeps proposing a value, then the value is eventually persistently written in the register of the partition.

Process  $p$  attempts to write a value  $val = (vset, vid)$  persistently in the register by invoking  $RPP-propose(p, vset, vid)$ .  $p$  decides on a value when it returns from the invocation of  $RPP-return(p, vset, vid)$ . The returned value may be  $(p, \perp, \perp)$ , then meaning that the consensus is aborted. Otherwise, the consensus is reached.  $\diamond RPP$  satisfies the following properties.

**Property 12.**  *$RPP$ -Non triviality of abandon. In a partition, if there exists a set  $\alpha Set$  composed of at least  $\alpha$  stable processes and if  $p \in \alpha Set$  proposes a value  $val = (vset, vid)$  an infinite number of times, with  $vset \subseteq \alpha Set \wedge |vset| \geq \alpha$ , then  $p$  eventually decides  $val' = (vset, vid')$ . If there does not exist a set composed of at least  $\alpha$  stable processes, then  $p$  abandons.*

**Property 13.**  *$RPP$ -Agreement per partition. Idem  $AC$ -Agreement.*

**Property 14.**  *$RPP$ -Validity per partition. Idem  $AC$ -Validity per partition.*

## 6 Implementation

In this section, we complete our contribution by presenting an implementation of  $\mathcal{PGM}$ . The proofs are provided in Appendices A and B.

### 6.1 Eventual $\alpha$ partition-participant detector and retransmission module

We already have presented an algorithm that implements  $\diamond PPD$  in a previous work [Lim and Conan, 2012]. So, we do not give the algorithm here, but we recap the idea. The algorithm is based on the periodic exchange of heartbeat messages to identify the current processes that are mutually reachable. Each process uses its local stability criterion to determine  $\alpha$  processes which are the most stable ones —*i.e.*, the ones that have exchanged the largest

number of heartbeats. Eventually, in a partition, all the processes in a stable set  $\alpha Set$  composed of at least  $\alpha$  stable processes have the same  $\alpha Set$  and elect the same process in  $\alpha Set$  as the leader.

The retransmission module is used to deal with losses of messages sent through SADDM paths. The implementation of this module and its proof can be found in [Lim, 2012]. We present the idea of the algorithm in this paragraph. Each message  $m$  that must be transmitted is associated with a set of destination nodes that is included in the current partition. The original sender  $p$  of  $m$  keeps broadcasting  $m$  periodically as long as it does not receive an acknowledgement from all the destination nodes, as done in [Guerraoui et al., 1997, Guerraoui and Rodrigues, 2006] to implement stubborn links. All the nodes of the partition participate by playing the role of “followers”. Without any special treatment,  $p$  may continue broadcasting  $m$  infinitely when destination processes disappear from  $p$ ’s partition. The retransmission module invokes  $\diamond PPD$  to address this issue and processes stop re-broadcasting  $m$  if some destination process disappears from the partition. Thus, the algorithm is “quiescent” [Aguilera et al., 1999, Aguilera et al., 2000].

## 6.2 Eventual register per partition

Algorithm 1 implements the eventual register per partition  $\diamond RPP$  for process  $p$ . The goal of the algorithm is to ensure that if  $p$  keeps proposing a value, then the value is eventually persistently written in the register. Each proposed value is composed of a set of processes  $vset_p \subseteq 2^{\mathbb{P}}$  and a unique identifier  $vid_p \in \mathbb{V}$  with  $vset_p$  and  $vid_p$  being the members and the identifier of the potential new view. To ensure uniqueness of view identifiers,  $p$  proposes values with increasing identifiers that have not already been chosen by another process in the partition. This is done by having  $p$  increment its identifier  $vid_p$  such that  $vid_p > lastProposal_p.id$  (Line 7) with  $lastProposal_p$  being the last proposal that  $p$  has proposed or accepted.  $lastProposal_p$  is stored in  $p$ ’s local stable memory so that  $p$  can retrieve the value when recovering from failure (Task 1).  $p$  broadcasts its proposal in a READ message by using the retransmission module in the procedure  $RPP$ -propose (Line 9). This broadcast corresponds to the beginning of the two phases of the procedure: A read phase (Lines 14–27) and then a write phase (Lines 29–41). We describe these two phases in the rest of this section.

**Read phase** The goal of this phase is to prepare the set of processes  $vset_p$  to accept  $p$ ’s proposal  $(vset_p, vid_p)$ . It consists of Tasks 2 and 3. Task 2 handles the reception of ACKREAD or NACKREAD messages of  $p$ ’s READ message whereas Task 3 treats the reception of READ messages originally broadcast by other processes. We present the two tasks in the following paragraphs.

**Task 2** While proposing a value,  $p$  checks that no process in  $vset_p$  has encountered a proposal with the same set  $vset_p$  and a greater identifier. If a process in  $vset_p$  with such a proposal exists then  $p$  aborts because this means that there exists at least some process  $q$  which proposes concurrently. In such a case,  $p$  receives a NACKREAD message containing  $vid_p$  and  $vset_p$  from  $q$  (Line 33).  $p$  also aborts when one or several processes in  $vset_p$  are no longer considered to be stable by  $p$  (Line 16). When  $p$  receives an ACKREAD message with the identifier  $vid_p$  from all the processes in  $vset_p$  and if these processes are included in  $p$ ’s set of stable processes provided by  $ppd$ ,  $p$  chooses the response with the highest identifier (Line 20), and updates  $vid_p$  and  $lastProposal_p$  (Line 21). Afterwards,  $p$  begins the write phase by broadcasting a WRITE message containing its proposal  $(vset_p$  and  $vid_p)$ .

**Task 3** Process  $p$  checks whether the proposal just received is eligible —*i.e.*, the condition  $vid_p < vid_q \wedge vset_q \not\subseteq \alpha Set_p$  provided by  $ppd$  (Line 25) does not hold. The eligibility of a proposal reflects the fact that all the members in the proposed view are stable and that the identifier of the proposal is strictly greater than the identifier of  $p$ 's last proposal. When  $q$ 's proposal is not eligible,  $p$  refuses  $q$ 's proposal and replies to  $q$  by broadcasting a NACKREAD message containing  $vid_p$  (Line 26). Otherwise,  $p$  verifies that it is included in  $vset_p$  and then  $p$  broadcasts an ACKREAD message containing  $vid_q$  and  $vid_p$  to  $q$  (Line 27).

**Write phase** The aim of this phase is to convince processes in  $vset_p$  to write  $p$ 's value in the persistent register. Like in the read phase, this attempt can fail in case of concurrent accesses. The write phase consists of Tasks 4 and 5. Task 4 handles the reception of ACKWRITE and NACKWRITE messages, and the changes of  $\alpha Set$ . Task 5 treats write requests. We present the two tasks in the following.

**Task 4** Process  $p$  abandons its current proposal upon the reception of a NACKWRITE message containing the identifier  $vid_p$ . This means that some process  $q$  proposed a value with the same set  $vset_p$  but the associated identifier is greater than the identifier  $vid_p$ : The NACKWRITE message is broadcast by  $q$  at Line 41 when the test at line 38 fails. Like in the read phase,  $p$  also abandons its current proposal when one or several processes in  $vset_p$  are excluded from the set of stable processes  $\alpha Set_p$  provided by  $ppd$ . Otherwise, when  $p$  receives an ACKWRITE message with the identifier  $vid_p$  from all the processes in  $vset_p$  then  $p$  decides on its proposed value.

**Task 5** Upon the reception of a WRITE message from process  $q$  with  $vset_q$  and  $vid_q$ ,  $p$  checks whether it can accept  $q$ 's value. This is done by having  $p$  verify whether its last proposal contains the same set  $vset_q$  with a smaller identifier. If so,  $p$  changes the value of  $lastProposal_p$  to  $(vset_q, vid_q)$  (Line 34). Otherwise,  $p$  refuses  $q$ 's proposal by broadcasting a NACKWRITE message with  $vid_p$  to  $q$ .

### 6.3 Abortable consensus

Algorithm 2 implements abortable consensus  $\mathcal{AC}$  for process  $p$ . The algorithm combines an instance of the module  $\diamond\mathcal{RPP}$  with an instance of the module  $\diamond\mathcal{PPD}$ . The variable *decision* is used to store the decision value.

After proposing a value  $val = (set, id)$  by calling the procedure  $\mathcal{AC}\text{-propose}$ ,  $p$  verifies 1) if the set of processes  $\alpha Set$  provided by  $ppd$  is still composed of at least  $\alpha$  processes, 2) if  $p$  is the leader of  $\alpha Set$ , and 3) if  $p$  has not decided yet. If so,  $p$  keeps continuing to propose its value. The value returned by abortable consensus corresponds to the value returned by  $rpp$  (Line 8). Therefore, a proposal is abandoned by  $rpp$  when there exists another proposer that proposes concurrently, but the abortable consensus instance is not abandoned as long as  $p$  detects via  $ppd$  more than  $\alpha$  stable processes and believes itself to be the leader. Otherwise, the proposer abandons the abortable consensus instance.

### 6.4 Partitionable group membership

Algorithm 3 implements partitionable group membership  $\mathcal{PGM}$  for process  $p$ . The goal of the algorithm is twofold: 1) to check whether there exists a set of stable processes  $\alpha Set$  in  $p$ 's partition and 2) to install the same view at all the processes in  $\alpha Set$ . Variable *ac* represents an instance of abortable consensus. The potential successor view is stored in variable *decision*.  $\alpha_p$  is the

---

**Algorithm 1** Implantation of eventual register per partition for process  $p$ 


---

```

1  Init():
2  |  $lastProposal_p \leftarrow (vset, vid);$  { $p$ 's last proposal}
3  |  $ppd \leftarrow connect\ to\ \diamond PPD;$ 
4
5  Procedure  $\mathcal{RPP}\text{-propose}(vset_p, vid_p)$ 
6  | If  $lastProposal_p.id \geq vid_p$  then
7  | |  $vid_p \leftarrow lastProposal.id + 1;$ 
8  | |  $lastProposal_p \leftarrow (vset_p, vid_p); store(lastProposal_p);$ 
9  | |  $xbroadcast(p, \langle READ \mid vid_p, vset_p \rangle);$ 
10
11 Task T1 : upon recovery
12 |  $retrieve(lastProposal_p);$ 
13
14 Task T2: upon  $[\forall q \in vset_p : xreceive(m) \text{ with } m = (q, \langle \{ACKREAD \mid vid_p, vid_q\}, \{p\})$ 
15  $\vee m = (q, \langle \{NACKREAD \mid vid_p\}, \{p\})] \vee [\alpha Set_p \text{ provided by } ppd \text{ has changed } \wedge \exists r \in vset_p : r \notin \alpha Set_p]$ 
16 | If  $(\alpha Set_p \text{ provided by } ppd \text{ has changed } \wedge \exists r \in vset_p : p \notin \alpha Set_p)$  then
17 | | generate  $\mathcal{RPP}\text{-return}(\perp, \perp);$ 
18 | | If received at least one  $(q, \langle \{NACKREAD \mid vid_p\}, \{p\})$  then
19 | | | generate  $\mathcal{RPP}\text{-return}(\perp, \perp);$ 
20 | | Else
21 | | | select the response  $(q, \langle \{ACKREAD \mid vid_p, vid_q\}, \{p\})$  with the highest  $vid_q;$ 
22 | | |  $vid_p \leftarrow \max(vid_p, vid_q) + 1; lastProposal_p \leftarrow (vset_p, vid_p); store(lastProposal_p);$ 
23 | | |  $xbroadcast(p, \langle WRITE \mid vid_p, vset_p \rangle);$ 
24
25 Task T3: upon  $xreceive(m)$  with  $m = (q, \langle READ \mid vid_q \rangle, vset_q)$ 
26 | If  $(vid_p < vid_q \wedge vset_q \not\subseteq \alpha Set_p)$  provided by  $ppd$  then
27 | |  $xbroadcast(p, \langle \{NACKREAD \mid vid_p\}, \{q\} \rangle)$ 
28 | | Else If  $p \in vset_q$  then  $xbroadcast(p, \langle \{ACKREAD \mid vid_q, vid_p\}, \{q\} \rangle);$ 
29
30 Task T4: upon  $[\forall q \in vset_p : xreceive(m) \text{ with } m = (q, \langle \{ACKWRITE \mid vid_p\}, \{p\})$ 
31  $\vee m = (q, \langle \{NACKWRITE \mid vid_p\}, \{p\})] \vee [\alpha Set_p \text{ provided by } ppd \text{ has changed } \wedge \exists r \in vset_p : r \notin \alpha Set_p]$ 
32 | If  $(\alpha Set_p \text{ provided by } ppd \text{ has changed } \wedge \exists r \in vset_p : p \notin \alpha Set_p)$  then
33 | | generate  $\mathcal{RPP}\text{-return}(\perp, \perp);$ 
34 | | If received at least one  $(q, \langle \{NACKWRITE \mid vid_p\}, \{p\})$  then
35 | | | generate  $\mathcal{RPP}\text{-return}(\perp, \perp);$ 
36 | | Else generate  $\mathcal{RPP}\text{-return}(vset_p, vid_p);$ 
37
38 Task T5: upon  $xreceive(m)$  with  $m = (q, \langle WRITE \mid vid_q \rangle, vset_q)$ 
39 | If  $(p \in vset_q)$  then
40 | | If  $(vid_q \geq vid_p \vee vid_p = \perp)$  then
41 | | |  $lastProposal_p \leftarrow (vset_p, vid_q); store(lastProposal_p);$ 
42 | | |  $xbroadcast(p, \langle \{ACKWRITE \mid vid_q\}, \{q\} \rangle);$ 
43 | | Else  $xbroadcast(p, \langle \{NACKWRITE \mid vid_q\}, \{q\} \rangle);$ 

```

---

same variable as the one used in the algorithm that implements  $\diamond PPD$  [Lim and Conan, 2012]. All the variables are initialised in the phase Init.

Process  $p$  proposes a successor view  $v_{new}$  with  $v_{new} = (set, id) \wedge |vset_{new}| \geq \alpha_p \wedge id > v.id$  by executing an instance of abortable consensus with  $v_{new}$  being its proposal for the consensus decision. The returned value of  $ac$  is not necessarily a value that was proposed by some process —*i.e.*,  $decision$  may equal  $(\perp, \perp)$  meaning that  $p$ 's proposal for installing  $v_{new}$  has been aborted. Algorithm 3 notifies the application layer about this abandon by generating a `nack_view_change` event (Line 11). If  $decision \neq (\perp, \perp)$  then  $decision$  corresponds to the successor view of  $p$ 's current view and  $p$  uses the retransmission module to broadcast this decision to the members of this new view.

In Task 1, upon the reception of a DECISION message from  $q$  containing the decided value  $decision_q$ ,  $p$  accepts to install the new view  $decision_q$  only if  $p$  is a member of this view and the identifier of  $p$ 's current view is not greater than  $decision_q.id$  (Line 14). If these conditions hold then Algorithm 3 notifies the application layer about the new view by generating a `view_change`

event (Line 16).

---

**Algorithm 2** Implementation of abortable consensus for process  $p$

---

```

1 Init():
2   |  $rpp \leftarrow \text{create } \diamond\mathcal{RPP}$ ;
3   |  $ppd \leftarrow \text{create } \diamond\mathcal{PPD}$ ;
4   |  $decision \leftarrow (\perp, \perp)$ ;
5
6 Procedure  $\mathcal{AC}\text{-propose}(vset, vid)$ 
7   | While ( $|ppd.participants().\alpha Set| \geq \alpha \wedge ppd.participants().l = p \wedge decision = (\perp, \perp)$ ) do
8   |   |  $decision \leftarrow rpp.\mathcal{RPP}\text{-propose}(vset, vid)$ ;
9   |   | generate  $\mathcal{AC}\text{-return}(decision)$ ;

```

---



---

**Algorithm 3** Implementation of partitionable group membership for process  $p$

---

```

1 Init():
2   |  $ac \leftarrow \text{create } \mathcal{AC}$ ;                                     {Abortable consensus instance}
3   |  $\alpha_p \leftarrow n \geq 1$ ;                                 {Same as  $\alpha_p$  in  $\diamond\mathcal{PPD}_p$ }
4   |  $decision \leftarrow (\perp, \perp)$ ;                             {Tuple  $(members, id)$ }
5
6 Procedure  $\text{propose}(v_{new})$  with  $v_{new} = (set, id) \wedge |set| \geq \alpha_p \wedge id \geq decision.id \vee decision.id = \perp$ 
7   |  $decision \leftarrow ac.\text{propose}(v_{new}.set, v_{new}.id)$ ;       {Abortable consensus decision}
8   | If  $decision \neq (\perp, \perp)$  then
9   |   |  $x\text{broadcast}(p, \langle \text{DECISION} \mid decision \rangle, decision.members)$ ;
10  | Else
11  |   | generate  $nack\_view\_change(v_{new})$ ;
12
13 Task 1: upon  $x\text{receive}(m)$  with  $m = (q, \langle \text{DECISION} \mid decision_q \rangle, *)$ 
14  | If ( $p \in decision_q.members \wedge (decision.id = \perp \vee decision_q.id > decision.id)$ ) then
15  |   |  $decision \leftarrow decision_q$ ;
16  |   | generate  $view\_change(decision)$ ;                         {Notify application about current view}

```

---

## 7 Related works

In this section, we discuss some works related to partition-participant detectors in MANETs, registers in dynamic systems and partitionable group membership services dedicated to MANETs.

### 7.1 Participant detectors in MANETs

[Aguilera et al., 1999] presents the heartbeat failure detector  $\mathcal{HB}$  for partitionable networks.  $\mathcal{HB}$  provides at each process an array with one entry for each process of the system. The heartbeat sequence of a process not in the partition is bound. The notion of the eventual  $\alpha$  partition-participant is inspired from this work. The authors also consider that links are fair. However, fair links may suffer from arbitrary delays and/or losses. We extend the concept of simple dynamic paths [Mostefaoui et al., 2005] into SADDM paths. A SADDM path combines the lossy property of a fair link and the timeless property of an eventually timely link [Aguilera et al., 2003]. A SADDM link is weaker than a timely link because it allows messages to be lost or arbitrarily delayed. It is stronger than a fair link because it guarantees that some subset of the broadcast messages are received in a timely manner and that such messages are not too sparsely distributed in time. The system in [Aguilera et al., 1999] is considered to be a fully-connected static one; the number and the identity of processes are known in advance; and nodes do not move or leave the system.

[Cavin et al., 2004] introduced the concept of participant detector that is used to solve the problem of bootstrapping in MANETs. The authors consider that the identity and the number of processes in the network are initially unknown. However, processes are always connected through reliable bidirectional links and do not crash. Participant detectors are defined for discussing about the minimal information that processes must have about the other participants in order to make the problem of consensus with unknown participants solvable.

[Nesterenko and Schiper, 2007] proposes an eventual reachability failure detector  $\diamond\mathcal{R}$  that provides to each process a set of processes called quorum. The authors define the concept of reachability graph  $R$  that is a directed multi-graph. Nodes in  $R$  correspond to participants of the system and there exists a path from process  $p$  to process  $q$  if the output value of  $p$ 's quorum contains  $q$ . According to the authors,  $\diamond\mathcal{R}$  can be extended and adapted to partitionable systems. However, the system is not dynamic, the number and the identify of the processes in the system are known by all the processes, and all the links are considered to be reliable. In addition, no implementation of  $\diamond\mathcal{R}$  is provided.

[Greve et al., 2011] extends the QUERY-RESPONSE communication mechanism of [Mostefaoui et al., 2005] by considering the mobility of nodes and proposes a failure detector  $\diamond S^M$  that eventually detects the set of *known* and *stable* processes: A process is known if it has joined the system and has been identified by a stable process and a process is stable if it never departs after having entered the system. Similarly to our approach, the authors define a parameter  $\alpha$  which corresponds to the expected number of processes that can communicate with each other despite of node movements, failures, etc. Then, they propose to compute the value of  $\alpha$  as the neighbourhood density of the process minus the maximum number of faulty processes in process's neighbourhood, which is supposed to be known. In our solution,  $\alpha$  is specified as a requirement by the application. In addition, the models in [Greve et al., 2011, Mostefaoui et al., 2005] target primary partition systems.

[Conan et al., 2008] proposes an eventually perfect unreliable partition detector for wireless networks which is composed of a heartbeat failure detector and a vector-based disconnection detector. The partition detector is able to detect partitions provoked by disconnections and failures. However, the total number of nodes of the system is known and the proposed solution is neither based on the definition of stable periods nor on the definition of dynamic paths.

## 7.2 Registers for dynamic systems

[Baldoni et al., 2009a] implements a regular register in dynamic systems with infinite arrival with  $b$ -bounded concurrency [Merritt and Taubenfeld, 2000, Aguilera, 2004]. The authors assume that the churn rate is constant: at every unit of time, the number of processes that leave the system equals the number of processes that join the system. They propose an adapted version of the regular register for dynamic systems and give two implementations: for synchronous systems and for eventually partially synchronous systems. These implementation are however designed for primary partition systems. [Baldoni et al., 2009b] extends the model proposed in [Baldoni et al., 2009a] for building dynamic registers when the churn rate is not constant.

## 7.3 Partitionable group membership for MANETs

[Filali et al., 2006] proposes a partitionable group membership algorithm for MANETs in which nodes and communication links continuously appear and disappear. Partitions correspond to cliques of the undirected network graph. The graph is specified as a set of *Node* and installed views are subsets of *Node*. Each process eventually installs a view that contains the members of

its current clique. A stability criterion called maximal criterion is used to build non-extensible disjoint sets of nodes when the system stabilises. The authors do not give any definition of stability, but note that a system that remains relatively stable is needed (which does not rule out unexpected failures or disconnections). In addition, members of the group are limited to nodes at a 2-hops distance.

According to [Boulkenafed et al., 2005], partitionable group membership for MANETs can be defined according to a functional property  $f$  to be realised.  $f$  refers to some interest attributes supported by the nodes that form the group —*e.g.*, localisation, security domain, constraints related to service quality and connectivity of the network. A node is included in a group if it is eligible according to the attributes required by the group. The proposed solution is similar to our solution in the sense that the interest attributes can be used as stability criteria. The stability criterion that we propose is based on heartbeat counters per period in order to capture the connectivity via the reachability relation and the stability of the network via the concept of SADDM paths.

## 8 Conclusion

In this article, we have presented a distributed system model adapted to the dynamic characteristics of MANETs, and then a specification called  $\mathcal{PGM}$  and an implementation of partitionable group membership for MANETs. This specification satisfies the two antagonistic requirements. Firstly,  $\mathcal{PGM}$  is strong enough because it avoids capricious views installation during stable periods because the removal or inclusion of a process is allowed only if this process was suspected to leave or join the partition, respectively. In addition, a new view is eventually installed in order to reflect the event. Secondly,  $\mathcal{PGM}$  is weak enough to allow an implementation, an example of which we provide and prove. In short, we have defined a system model that takes into account the formation of dynamic paths in MANETs. Then, the partitionable group membership problem is solved by transformation into a sequence of abortable consensus  $\mathcal{AC}$ .  $\mathcal{AC}$  is specified as the combination of two abstractions: An eventual  $\alpha$  partition-participant detector  $\diamond PPD$  and an eventual register per partition  $\diamond RPP$ .  $\diamond PPD$  captures liveness in a partition by detecting  $\alpha$  stable processes whereas  $\diamond RPP$  encapsulates safety by materialising a distributed register in the same partition.

## References

- [Aguilera et al., 2003] Aguilera, M., Delporte-Gallet, C., Fauconnier, H., and Toueg, S. (2003). On implementing Omega with weak reliability and synchrony assumptions. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing*, pages 306–314, New York, USA.
- [Aguilera, 2004] Aguilera, M.-K. (2004). A Pleasant Stroll through the Land of Infinitely Many Creatures. *Distributed Computing Column of ACM SIGACT News*, 35(2):36–59.
- [Aguilera et al., 1999] Aguilera, M.-K., Chen, W., and Toueg, S. (1999). Using the Heartbeat Failure Detector for Quiescent Reliable Communication and Consensus in Partitionable Networks. *Theoretical Computer Science*, 220(1):3–30.
- [Aguilera et al., 2000] Aguilera, M.-K., Chen, W., and Toueg, S. (2000). On Quiescent Reliable Communication. *SIAM Journal on Computing*, 29(6):2040–2073.

- [Anceaume et al., 1995] Anceaume, E., Charron-Bost, B., Minet, P., and Toueg, S. (1995). On the Formal Specification of Group Membership Services. Technical Report TR 95-1534, Department of Computer Science, Cornell University, New York, USA.
- [Anker et al., 1999] Anker, T., Dolev, D., and Keidar, I. (1999). Fault tolerant video on demand services. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 244–252.
- [Babaoğlu et al., 2001] Babaoğlu, O., Davoli, R., and Montresor, A. (2001). Group Communication in Partitionable Systems: Specification and Algorithms. *IEEE Transactions on Software Engineering*, 27(4):308–336.
- [Babaoğlu et al., 1998] Babaoğlu, O., Davoli, R., Montresor, A., and Segala, R. (1998). System Support for Partition-Aware Network Applications. *ACM SIGOPS Operating Systems Review*, 32:41–56.
- [Baldoni et al., 2009a] Baldoni, R., Bonomi, S., Kermarrec, A.-M., and Raynal, M. (2009a). Implementing a Register in a Dynamic Distributed System. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, pages 639–647, Montreal, Canada.
- [Baldoni et al., 2009b] Baldoni, R., Bonomi, S., and Raynal, M. (2009b). Regular Register: an Implementation in a Churn Prone Environment. In *Proceedings of the 16th international conference on Structural Information and Communication Complexity*, pages 15–29.
- [Birman et al., 1999] Birman, K., Friedman, R., Hayden, M., and Rhee, I. (1999). Middleware support for distributed multimedia and collaborative computing. *Software—Practice and Experience*, 29(14):1285–1312.
- [Boichat et al., 2003a] Boichat, R., Dutta, P., Frølund, S., and Guerraoui, R. (2003a). Deconstructing Paxos. *Distributed Computing Column of ACM SIGACT News*, 34(1):47–67.
- [Boichat et al., 2003b] Boichat, R., Dutta, P., Frølund, S., and Guerraoui, R. (2003b). Reconstructing Paxos. *Distributed Computing Column of ACM SIGACT News*, 34:42–57.
- [Boulkenafed et al., 2005] Boulkenafed, M., Sacchetti, D., and Issarny, V. (2005). Using Group Management to Tame Mobile Ad Hoc Networks. In *Proceedings of IFIP TC 8 Working Conference on Mobile Information Systems*, pages 245–260.
- [Brewer, 2000] Brewer, E. (2000). Towards robust distributed systems. In *Invited Talk, Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*.
- [Cavin et al., 2004] Cavin, D., Sasson, Y., and Schiper, A. (2004). Consensus with Unknown Participants or Fundamental Self-Organization. In *Proceedings of the 3rd*, number 3158 in Lecture Notes in Computer Science, pages 135–148, Vancouver, British Columbia, Canada. Springer-Verlag.
- [Chandra et al., 1996] Chandra, T.-D., Hadzilacos, V., Toueg, S., and Charron-Bost, B. (1996). On the Impossibility of Group Membership. In *Proceedings of the 15th ACM Symposium on Principles of Distributing Computing*, Philadelphia, USA.
- [Chandra and Toueg, 1996] Chandra, T.-D. and Toueg, S. (1996). Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225–267.

- [Chockler et al., 2001] Chockler, G.-V., Keidar, I., and Vitenberg, R. (2001). Group Communication Specifications: A Comprehensive Study. *ACM Computing Surveys*, 33(4):427–469.
- [Conan et al., 2008] Conan, D., Sens, P., Arantes, L., and Bouillaguet, M. (2008). Failure, Disconnection and Partition Detection in Mobile Environment. In *Proceedings of the 7th IEEE International Symposium on Network Computing and Applications*, pages 119–127.
- [Filali et al., 2006] Filali, M., Issarny, V., Mauran, P., Padiou, G., and Quéinnec, P. (2006). Maximal Group Membership in Ad Hoc Networks. In *Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics*, pages 51–58, Poznan, Poland.
- [Gilbert and Lynch, 2012] Gilbert, S. and Lynch, N.-A. (2012). Perspectives on the CAP theorem. *IEEE Computer*, 45(2):30–36.
- [Greve et al., 2011] Greve, F., Sens, P., Arantes, L., and V., S. (2011). A Failure Detector for Wireless Networks with Unknown Membership. In *Proceedings of the 17th*, pages 27–38, Berlin, Heidelberg. Springer-Verlag.
- [Guerraoui et al., 1997] Guerraoui, R., Oliveira, R., and Schiper, A. (1997). Stubborn Communication Channels. Technical Report TR97, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.
- [Guerraoui and Rodrigues, 2006] Guerraoui, R. and Rodrigues, L. (2006). *Introduction to Reliable Distributed Programming*. Springer-Verlag, NJ, USA.
- [Guerraoui and Schiper, 2001] Guerraoui, R. and Schiper, A. (2001). The Generic Consensus Service. *IEEE Transactions on Software Engineering*, 27(1):29–41.
- [Lamport, 1978] Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565.
- [Lamport, 1998] Lamport, L. (1998). The Part Time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169.
- [Lamport, 2001] Lamport, L. (2001). Paxos Made Simple. *Distributed Computing Column of ACM SIGACT News*, 32(4):18–25.
- [Lim, 2012] Lim, L. (2012). *Partitionable Group Membership for Mobile Ad Hoc Networks (in French)*. PhD thesis, Institut Mines-Télécom, Télécom SudParis, France.
- [Lim and Conan, 2012] Lim, L. and Conan, D. (2012). An eventual alpha partition-participant detector for manets. In *Proceedings of the Ninth European Dependable Computing Conference*, pages 25–36, Sibiu, Romania.
- [Merritt and Taubenfeld, 2000] Merritt, M. and Taubenfeld, G. (2000). Computing with Infinitely Many Processes Under Assumptions on Concurrency and Participation. In *Proceedings of the 14th*, pages 164–178.
- [Mostefaoui et al., 2005] Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and El Abbadi, A. (2005). From Static Distributed Systems to Dynamic Systems. In *Proceeding of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 109–118, Florianopolis (Brazil).

- [Murray, 2005] Murray, P. (2005). A Distributed State Monitoring Service for Adaptive Application Management. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, pages 200–205, Washington, DC, USA.
- [Nesterenko and Schiper, 2007] Nesterenko, M. and Schiper, A. (2007). On Properties of the Group Membership Problem. Technical Report TR-KSU-CS-2007-01, Kent State University.
- [Pleish et al., 2008] Pleish, S., Rütli, O., and Schiper, A. (2008). On the Specification of Partitionable Group Membership. In *Proceedings of the 7th European Dependable Computing Conference*, pages 37–45, Kaunas, Lithuania.
- [Sastry and Pike, 2007] Sastry, S. and Pike, S. (2007). Eventually Perfect Failure Detectors Using ADD Channels. In *Proceedings of the 5th international Symposium on Parallel and Distributed Processing and Applications*, pages 483–496, Niagara Falls, Canada.

## A Formal definitions, predicates, properties, and assumptions

The predicates, properties, and definitions are inserted in the order of their appearance in the article.

### A.1 Preliminary definition and predicates for presenting properties and definitions of partitionable group membership

In the following definition, the function  $pid : \mathbb{E} \rightarrow \mathbb{P}$  returns the identifier of the process associated to a given event.

**Definition 1.** *viewof*.

$$viewof(e) \stackrel{def}{=} \begin{cases} v & \text{if } \exists TS \exists t \exists t'' \exists e'' \nexists t' \nexists e' : \\ & H(pid(e), t) = e \\ & \wedge H(pid(e), t'') = e'' = \text{view\_change}(pid(e), v, TS) \\ & \wedge t'' < t' < t \\ & \wedge \left( H(pid(e), t') = e' = \text{crash}(pid(e)) \right. \\ & \quad \vee \exists TS' \exists v' : H(pid(e), t') = e' \\ & \quad \left. \wedge e' = \text{view\_change}(pid(e), v', TS') \right) \\ \perp & \text{otherwise} \end{cases}$$

$$alive(p) \stackrel{def}{=} \nexists t : H(p, t) = \text{crash}(p)$$

$$alive\_after(p, t) \stackrel{def}{=} \nexists t : H(p, t) = \text{crash}(p) \\ \vee \exists t'' \leq t \nexists t' > t'' : H(p, t'') = \text{recover}(p) \wedge H(p, t') = \text{crash}(p)$$

$$receive(p, m) \stackrel{def}{=} \exists t : H(p, t) = \text{receive}(p, m)$$

$$send(p, m) \stackrel{def}{=} \exists t : H(p, t) = \text{send}(p, m)$$

$$last\_view(p, v) \stackrel{def}{=} \exists t \nexists t' > t \exists v \exists TS \nexists v' \nexists TS' : H(p, t) = \text{view\_change}(p, v, TS) \\ \wedge H(p, t') = \text{view\_change}(p, v', TS')$$

### A.2 First set of properties and definitions of partitionable group membership

**Property 1.** Self inclusion.

$$\text{view\_change}(p, v, TS) \Rightarrow p \in v.members$$

**Property 2.** Local monotonicity.

$$\left( H(p, t) = \text{view\_change}(p, v, TS) \right. \\ \left. \wedge H(p, t') = \text{view\_change}(p, v', TS') \wedge t > t' \right) \Rightarrow v.id > v'.id$$

**Property 3.** Initial view event.

$$\left( e = \text{send}(p, m) \vee e = \text{receive}(p, m) \vee e = \text{broadcast}_{nbg}(p, m) \right) \Rightarrow viewof(e) \neq \perp$$

**Definition 2.** Fair link (for MANETs).

$$\left( \forall t \exists t_1 \geq t : H(p, t_1) = \text{broadcast}_{nbg}(p, m) \wedge alive\_after(q, t_1) \right) \\ \wedge \left( \forall t \exists t_2 \geq t : \wedge H(q, t_2) = \text{receive}(q, m') \right) \\ \wedge \text{broadcast}_{nbg}(p, m') \in H(p, [t, +\infty])$$

**Definition 3.** Reachability.

$$reachable(p, q) \stackrel{def}{=} FAIR(p \dots q)$$

**Definition 4.** SADDM Link.

$$\begin{aligned} SADDM(p, q) &\stackrel{def}{=} \exists I = [t_1, t_2] : \\ &\left( \forall t_i \in [1, \beta] \in I : H(p, t_i) = \text{broadcast}_{nbq}(p, m) \right. \\ &\wedge \text{alive\_after}(q, t_1) \\ &\left. \wedge \exists t' \leq t_1 + \delta : H(q, t') = \text{receive}(q, m) \wedge \delta > (t_2 - t_1) \right) \end{aligned}$$

**Definition 5.** Stable Partition.

$$\begin{aligned} \diamond PART_p &\stackrel{def}{=} \left\{ q \in S \subseteq \mathbb{P} \mid \exists LST_p \forall t' \geq LST_p \exists t'' > t' : \right. \\ &\left. SADDM(p \dots q)_{[t', t'']} \wedge SADDM(q \dots p)_{[t', t'']} \right\} \end{aligned}$$

**Definition 6.** Stability condition.

$$|\diamond PART_p| \geq \alpha_p$$

### A.3 Comparison, addition and subtraction operations on view identifiers

In order to ensure uniqueness of the view identifiers, we consider that each view identifier  $vid$  is taken from the set  $\mathbb{VID}$  that is a pair  $(p, c)$  with  $p$  being the process identifier and  $c \in \mathbb{N}$  the value of  $p$ 's local counter. We trivially define the comparison, addition and subtraction operations on the view identifiers as follows. Let  $vid' = (p, i)$  and  $vid = (q, j)$  be two view identifiers in  $\mathbb{VID}$  and  $k$  be an integer in  $\mathbb{N}$ :

$$\begin{aligned} vid' = vid &\stackrel{def}{=} (p = q \wedge i = j) \\ vid' > vid &\stackrel{def}{=} (p > q \vee p = q \wedge i > j) \\ vid' < vid &\stackrel{def}{=} (p < q \vee p = q \wedge i < j) \\ vid' \geq vid &\stackrel{def}{=} (p > q \wedge p = q \wedge i \geq j) \\ vid' \leq vid &\stackrel{def}{=} (p < q \wedge p = q \wedge i \leq j) \\ vid + k &\stackrel{def}{=} vid = (p, i + k) \\ vid - k &\stackrel{def}{=} vid = (p, i - k) \end{aligned}$$

### A.4 Properties of $\mathcal{PGM}$ , $\mathcal{AC}$ , $\diamond PPD$ , the retransmission module, and $\diamond RPP$

**Property 4.**  $\mathcal{PGM}$ -Validity.

$$\begin{aligned} &\left( \exists t \exists p : H(t, p) = \text{view\_change}(p, v) \right) \\ \Rightarrow &\left( \exists t' < t \exists q \in v.\text{members} : H(q, t') = \text{propose}(q, v') \right. \\ &\left. \wedge |v.\text{vset}| \geq \alpha_p \wedge v.\text{vid} \geq v'.\text{vid} \right) \end{aligned}$$

**Property 5.**  $\mathcal{PGM}$ -Agreement on final view.

$$\begin{aligned} \exists S \subseteq \alpha Set \subseteq \diamond PART_p \exists p \forall q \in S \exists t' \forall t \geq t' \exists v_f : \\ H(p, t') = \text{propose}(p, v_f) \Rightarrow \text{last\_view}(q, v_f) \end{aligned}$$

**Property 6.**  $\mathcal{AC}$ -Termination.

$$\begin{aligned} \exists p \exists vset \exists vid \exists t'' : \\ H(p, t'') = \mathcal{AC}\text{-propose}(p, vset, vid) \end{aligned}$$

$$\begin{aligned}
& \wedge \left[ \left( \forall t' \geq t'' : (\exists \alpha Set \subseteq \diamond PART_p : |\alpha Set| > \alpha) \Rightarrow \exists vset' \exists vid' \exists t > t'' : \right. \right. \\
& H(p, t) = \mathcal{AC}\text{-return}(p, vset', vid') \wedge vset' \neq \perp \wedge vid' \geq vid) \\
& \left. \vee \left( \forall t' \geq t'' : (\exists \alpha Set \in \diamond PART_p : |\alpha Set| > \alpha) \Rightarrow \exists t > t'' : \right. \right. \\
& \left. \left. H(p, t) = \mathcal{AC}\text{-return}(p, \perp, \perp) \right) \right]
\end{aligned}$$

In property  $\mathcal{AC}$ -Termination, observe that the identifier of the returned value is  $vid'$  with  $vid' \geq vid$ . The reason is the following. Proposer  $p$  may have to propose more than once before the consensus is reached. In order to ensure uniqueness of view identifiers and ignore old proposals,  $p$  issues a sequence of proposals with increasing identifiers. As a consequence, the identifier  $vid'$  that is eventually returned by  $\mathcal{AC}$  is greater or equals to  $vid$ . This reasoning also applies to the following properties.

**Property 7.**  $\mathcal{AC}$ -Agreement.

$$\begin{aligned}
& \exists vset \exists vset' \exists vid \exists vid' \forall p, q \in \alpha Set \subseteq \diamond PART_p \\
& \exists t' \forall t \geq t' \exists t_1 \geq t \exists t_2 \geq t : \\
& \left( H(p, t_1) = \mathcal{AC}\text{-return}(p, vset', vid') \wedge H(q, t_2) = \mathcal{AC}\text{-return}(p, vset, vid) \right) \\
& \Rightarrow (vset = vset' \wedge vid = vid')
\end{aligned}$$

**Property 8.**  $\mathcal{AC}$ -Validity.

$$\begin{aligned}
& \exists vset \exists vid \exists p \exists t : \\
& H(p, t) = \mathcal{RPP}\text{-return}(p, vset, vid) \\
& \Rightarrow \left( \exists q \in vset \exists t' < t : H(q, t') = \mathcal{RPP}\text{-propose}(q, vset, vid') \wedge vid' \leq vid \right)
\end{aligned}$$

**Property 9.** Eventual  $\alpha Set$  stability.

$$\begin{aligned}
& \forall p, q \in \alpha Set \subseteq \diamond PART_p \exists t' \forall t \geq t' \exists t_1 \geq t \exists t_2 \geq t : \\
& \left( H(p, t_1) = \mathcal{PPD}\text{-return}(p, l, \alpha Set) \wedge H(q, t_2) = \mathcal{PPD}\text{-return}(q, l', \alpha Set') \right) \\
& \Rightarrow \alpha Set = \alpha Set'
\end{aligned}$$

**Property 10.** Eventual leader agreement.

$$\begin{aligned}
& \forall p, q \in \alpha Set \subseteq \diamond PART_p \forall t \geq t' \exists t_1 \geq t \exists t_2 \geq t : \\
& \left( H(p, t_1) = \mathcal{PPD}\text{-return}(p, l, \alpha Set) \wedge H(q, t_2) = \mathcal{PPD}\text{-return}(q, l', \alpha Set') \right) \\
& \Rightarrow l = l'
\end{aligned}$$

**Property 11.** Reliable broadcast to a set of stable nodes.

$$\begin{aligned}
& \exists I = [t_1, t_2] \exists dest \forall q \in dest \exists r \in dest \exists m \exists n : \\
& \left( m = (p, \langle \text{TYPE} \mid p, \dots \rangle, dest) \right. \\
& \wedge n = |\text{SADDM}(p \dots r)| \geq |\text{SADDM}(p \dots q)| \\
& \left. \wedge \forall t_i \in [1, \beta^n] \in I \exists \eta = (\max(t_{i+1} - t_i) : \forall i \in [1, \beta^n]) : \right. \\
& \left. H(p, t_i) = \text{broadcast}_{nbq}(m) \right) \\
& \Rightarrow \left( \forall q \in dest \exists t \leq t_1 + \beta^n \eta + n\delta : H(q, t) = \text{receive}(q, m) \right)
\end{aligned}$$

**Property 12.**  $\mathcal{RPP}$ -Non triviality of abandon.

$$\begin{aligned}
& \exists p \exists vset \exists t' : \\
& H(p, t') = \mathcal{AC}\text{-propose}(p, vset, vid) \\
& \wedge \left[ \left( \forall t'' \geq t' \exists t_1 \geq t'' : \right. \right. \\
& \quad \left. \left( vset \subseteq \alpha Set \subseteq \diamond PART_p \exists vid : \right. \right. \\
& \quad \quad \left. |vset| \geq \alpha \wedge \mathcal{RPP}\text{-propose}(p, vset, vid) \in H(p, [t'', \infty]) \right) \\
& \quad \Rightarrow \exists t_2 \geq t_1 : H(p, t_2) = \mathcal{AC}\text{-return}(p, vset, vid') \wedge vset \neq \perp \wedge vid' \geq vid \Big) \\
& \vee \left( \forall t'' \geq t' : (\exists \alpha Set \in \diamond PART_p : |\alpha Set| > \alpha) \Rightarrow \exists t > t' : \right. \\
& \quad \left. H(p, t) = \mathcal{AC}\text{-return}(p, \perp, \perp) \right) \Big]
\end{aligned}$$

**Property 13.**  $\mathcal{RPP}$ -Agreement per partition. *Idem*  $\mathcal{AC}$ -Agreement.

**Property 14.**  $\mathcal{RPP}$ -Validity per partition. *Idem*  $\mathcal{AC}$ -Validity per partition.

## B Proofs of correctness

In this section, we show that Algorithms 1, 2 and 3 implement  $\diamond\mathcal{RPP}$ ,  $\mathcal{AC}$ , and  $\mathcal{PGM}$ , respectively.

### B.1 $\diamond\mathcal{RPP}$ —Algorithm 1

**Lemma 1.** *If process  $p$  decides value  $val = (vset, vid)$ , then  $val = (vset, vid')$  was proposed by some process  $q$  (possibly  $p$ ) in  $vset$ .*

*Proof.* Let  $p$  be a process that decides value  $val = (vset, vid)$  at instant  $t$ . Consider by contradiction that there does not exist some process  $q$  in  $vset$  which proposed  $val' = (vset, vid')$  at instant  $t_s < t$ .  $p$  deciding  $val$  means that the event  $\mathcal{RPP}\text{-return}(p, vset, vid)$  is executed (Line 34): —i.e.,  $p$  has received an ACKWRITE message from all the processes in  $vset_p$  and this ACKWRITE message is an acknowledgement of its broadcast WRITE message that contains  $vset_p$  and  $vid_p$ . The WRITE message is only broadcast at the end of a READ phase that in turn was launched by having  $p$  call the procedure  $\mathcal{RPP}\text{-propose}(p, vset_p, vid')$ . This is true because communication links do neither duplicate nor create messages.  $\square$

**Lemma 2.** *If  $p$  is the only process among the set of stable processes  $\alpha Set \subseteq \diamond PART_p$  that keeps proposing value  $val = (vset_p, vid_p)$  with  $vset_p \subseteq \alpha Set \wedge |vset_p| \geq \alpha$ , then  $p$  eventually decides  $val' = (vset_p, vid'_p)$ .*

*Proof.* Let  $p$  be a process among the set of stable processes  $\alpha Set \subseteq \diamond PART_p$  that keeps proposing an infinite of times its value  $val = (vset_p, vid_p)$  with  $vset_p \subseteq \alpha Set \wedge |vset_p| \geq \alpha$ . Since  $p$  is eventually the only process that keeps proposing its value, there must exist an instant  $t_1$  after which no process  $q \neq p$  in  $\alpha Set$  proposes a value.

Consider by contradiction that there does not exist an instant  $t_2 > t_1$  at which  $p$  decides  $val' = (vset_p, vid'_p)$ . After  $t_1$ ,  $p$  proposes its value infinitely often. Since  $p$  proposes values with increasing identifiers (Line 7), there must exist an instant  $t_3 > t_1$  after which  $vid_p$  becomes greater than all the identifiers encountered by any process  $q \in vset_p \wedge q \neq p$ . The retransmission module used by  $p$  guarantees that there exists an instant  $t_4 > t_3$  at which  $p$  eventually succeeds in broadcasting its message  $(p, \langle \text{READ } |vid_p\rangle, vset_p)$  to all the processes in  $\alpha Set_p$  with  $vset_p \subseteq \alpha Set_p$  (Property 11). The only scenario that prevents  $p$  from broadcasting its READ message occurs when  $p$  has received a message  $(r, \langle \text{NACKREAD } |vid_r\rangle, \{p\})$  (Line 29) from some process  $r \in vset_p$

such that  $vid_r > vid_p$ . This contradicts the fact that  $p$  has the highest identifier after  $t_3$ . Therefore, processes in  $vset_p$  eventually receive the READ message of  $p$  containing  $vid_p$  and reply to  $p$  with an ACKREAD message that is tagged with  $vid_p$ . Then,  $p$  eventually succeeds in terminating the read phase tagged with  $vid_p$  when  $p$  receives an ACKREAD message tagged  $vid_p$  from all the processes in  $vset_p$ . After  $t_4$ ,  $p$  starts the write phase by broadcasting a WRITE message containing  $vset_p$  and  $vid'_p > vid_p$  (Lines 21–22). Since, all the processes in  $vset_p$  are stable, they eventually receive a message  $(p, \langle \text{WRITE} \mid vid'_p \rangle, vset_p)$  and reply to  $p$  with an ACKWRITE message that contains  $vid'_p$ . Afterwards,  $p$  terminates the write phase of its proposal upon the reception of these ACKWRITE messages.  $p$  eventually decides  $val' = (vset_p, vid'_p)$  after  $t_4 > t_3 > t_1$ . This contradicts the fact that there does not exist an instant  $t_2 > t_1$  at which  $p$  decides  $val'$ .  $\square$

For the following Lemma, we use the Lemma 6 of [Lim and Conan, 2012] stipulating that for stable process  $p$  such that  $\forall q \in \diamond PART_p : \alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q)$ , there exists a time after which  $\alpha Set_q = \alpha Set_p$  remains true.

**Lemma 3.** *Let  $p$  and  $q$  be any two processes in  $vset \subseteq \alpha Set \subseteq \diamond PART_p$  with  $|vset| \geq \alpha$ . There exists an instant after which  $p$  and  $q$  do not decide differently.*

*Proof.* Let  $p$  and  $q$  be any two processes in  $\alpha Set \subseteq \diamond PART_p$  with  $|\alpha Set| \geq \alpha$ . From Lemma 6 of [Lim and Conan, 2012], there exists an instant  $t$  after which for each process  $q$  in  $\alpha Set_p$ ,  $\alpha Set_l = \alpha Set_p = \alpha Set_q$  with  $l$  being the leader of  $\alpha Set$ . Let  $val = (vset_l, vid_l)$  be the value that was decided by  $l$  after  $t$ . To show that  $p$  and  $q$  do not eventually decide different values, we show that each process  $q \in vset_l \subseteq \alpha Set_l$  eventually decides the same value as  $l$ 's decision value.

Since  $l$  is the eventual leader, there must exist an instant  $t_1$  after which  $l$  is the only process that keeps proposing values. From Lemma 2,  $l$  eventually decides the value  $val' = (vset_l, vid'_l)$  — *i.e.* read and write phases associated to  $l$ 's proposal that contains  $vset_l$  and  $vid_l$  with  $vid_l \leq vid'_l$  were terminated with success.  $l$  decides the value  $val'$  and broadcasts the WRITE message containing the decision value to all the processes in  $vset_l$  by using the retransmission module (Line 22). Processes in  $vset_l$  eventually receive  $l$ 's WRITE message containing value  $val = (vset_l, vid'_l)$  (Property 11). This value is accepted by all the processes in  $vset_l$  (Line 39).  $\square$

**Theorem 1.** *Algorithm 1 implements  $\diamond RPP$  by satisfying  $RPP$ -Validity per partition,  $RPP$ -Agreement per partition and  $RPP$ -Non triviality of abandon properties.*

*Proof.* Consider a stable process  $p$ . From Lemma 1, if  $p$  decides value  $val = (vset, vid)$ , then  $val' = (vset, vid')$  was proposed by some process  $q$  (possibly  $p$ ) in  $vset$ . This satisfies the  $RPP$ -Validity per partition property. From Lemma 2, if  $p$  is the only stable process that keeps proposing, then  $p$  eventually decides its proposed value. This satisfies the  $RPP$ -Non triviality of abandon property. From Lemma 3, if  $p$  decides value  $val = (vset, vid)$ , then for each process  $q \in vset \subseteq \diamond PART_p$ ,  $q$  eventually decides  $val$ . This satisfies the  $RPP$ -Agreement property.  $\square$

## B.2 AC—Algorithm 2

**Lemma 4.** *If process  $p$  decides value  $val = (vset, vid)$ , then  $val' = (vset, vid')$  was proposed by some process  $q$  (possibly  $p$ ) in  $vset$ .*

*Proof.* Consider that process  $p$  decides value  $val = (vset, vid)$  at instant  $t$ . From Line 8,  $val$  was decided by the module  $\diamond\mathcal{RPP}$ . From property  $\mathcal{RPP}$ -Validity per partition,  $val' = (vset, vid')$  was proposed by some process  $q$  (possible  $p$ ) in  $vset$  at instant  $t' < t$ .  $\square$

**Lemma 5.** *Let  $p$  and  $q$  be any two processes in  $\alpha Set \subseteq \diamond PART_p$ . There exists an instant after which  $p$  and  $q$  decide the same value.*

*Proof.* Let  $p$  and  $q$  be any two processes in  $\alpha Set \subseteq \diamond PART_p$ . From Line 8,  $val$  was decided by  $rpp$ , and from property  $\mathcal{RPP}$ -Agreement per partition, there exists an instant after which  $p$  et  $q$  decide the same value.  $\square$

**Lemma 6.** *If process  $p$  proposes a value and if there exists a set  $\alpha Set$  constituted of at least  $\alpha$  stable processes that elect  $p$  as being the leader of  $\alpha Set$ , then  $p$  eventually decides. Otherwise,  $p$  abandons.*

*Proof.* Let  $p$  be a process that proposes a value. If there does not exist a set  $\alpha Set$  constituted of at least  $\alpha$  stable processes that include  $p$ , then  $p$  abandons (Lines 9). Consider that there exists a set  $\alpha Set$  constituted of at least  $\alpha$  stable processes that elect  $p$  as being the leader. We show that  $p$  eventually decides. From property  $\mathcal{RPP}$ -Non triviality of abandon, if  $p$  is the only process in  $\alpha Set \subseteq \diamond PART_p$  with  $|\alpha Set| \geq \alpha$  that keeps proposing value  $val = (vset, vid)$  with  $vset \subseteq \alpha Set \wedge |vset| \geq \alpha$ , then  $p$  eventually decides  $val' = (vset, vid')$ . If there does not exist a set  $\alpha Set$  constituted of at least  $\alpha$  stable processes, then  $p$  stops proposing (Line 8). Since  $p$  has not previously decided,  $decision$  equals to  $(\perp, \perp)$  meaning that  $p$  abandons.  $\square$

**Theorem 2.** *Algorithm 2 implements  $\mathcal{AC}$  by satisfying  $\mathcal{AC}$ -Validity,  $\mathcal{AC}$ -Termination and  $\mathcal{AC}$ -Agreement.*

*Proof.* Consider a process  $p$ . From Lemma 4, if  $p$  decides value  $val = (vset, vid)$ , then  $val' = (vset, vid')$  was proposed by some process  $q$  (possible  $p$ ) in  $vset$  and  $vset$  is constituted of at least  $\alpha_p$  processes. This satisfies the  $\mathcal{AC}$ -Validity property. From Lemma 6, if  $p$  proposes a value and there exists a set  $\alpha Set$  constituted of at least  $\alpha$  stable processes which elect  $p$  as the leader, then  $p$  eventually decides. This satisfies the  $\mathcal{AC}$ -Termination property. Let  $p$  and  $q$  any two processes in  $\alpha Set \subseteq \diamond PPD$ . From Lemma 5, there exists an instant after which  $p$  and  $q$  decide the same value. This satisfies the  $\mathcal{AC}$ -Agreement property.  $\square$

### B.3 Partitionable group membership—Algorithm 3

**Lemma 7.** *If process  $p$  installs view  $v$ , then  $p$  is a member of  $v$ .*

*Proof.* Let  $p$  be a process that installs view  $v$ . Since DECISION messages which contains  $v$  are broadcast to the processes in  $v.members$  (Line 9),  $v$  is included in the decision message that  $p$  receives (Line 13). Upon the reception of the decision message  $(q, \langle \text{DECISION} \mid decision_q \rangle, *)$ ,  $p$  installs view  $v$  with  $v.id = decisions_q.id$  (Lines 15–16) only if  $v.members$  (Line 14).  $\square$

**Lemma 8.** *If process  $p$  installs view  $v$  after installing view  $v'$ , then the identifier of  $v$  is strictly greater than  $v'$ 's identifier.*

*Proof.* Let  $p$  be a process and  $v$  its current view.  $p$  installs new view  $v$ , the successor of  $v'$ , only if  $v.id > v'.id$  and  $p \in v.members$  (Line 14).  $\square$

**Lemma 9.** *If process  $p$  decides value  $val = (vset, vid)$ , then  $val' = (vset, vid')$  was proposed by some process  $q$  (possibly  $p$ ) in  $vset$  that contains at least  $\alpha_p$  processes.*

*Proof.* Let  $p$  be a process that decides value  $val = (vset, vid)$  at instant  $t$ . From Lemma 4,  $val' = (vset, vid')$  was proposed by some process  $q$  (possibly  $p$ ) in  $vset$  at instant  $t' < t$ . Consider that  $l$  is the process that proposed value  $val'$  and is the first process that decides  $val'$ —i.e.  $l$  is the first process that broadcast DECISION message containing value  $val$ . This means that  $l$  terminated its consensus instance with the proposed value  $val = (vset, vid)$  and the event  $\mathcal{AC}\text{-return}(p, decision_l)$  with  $decision_l = (vset, vid)$  was generated. This is only possible if  $l$  considers itself to be leader of  $\alpha Set \supseteq vset$  (because only proposers can propose values) and if  $|vset| \geq \alpha_l$  (Line 6) at instant at which  $l$  proposes value  $val = (vset, vid)$ . Therefore, for each process  $q$  in  $vset$ ,  $\alpha_l \geq \alpha_q$ . Processes in the set of stable processes  $vset$  eventually receive  $l$ 's decision containing  $val$  with  $|val.vset| \geq \alpha_l \geq \alpha_q$  for each process  $q$  in  $vset$ .  $\square$

**Lemma 10.** *If there exists a set of stable processes  $S \subseteq \alpha Set \subseteq \diamond PART_p$  that wish to install view  $val' = (S, vid')$ , then all the processes in  $S$  eventually install the same final view  $val$ .*

*Proof.* Let  $S \subseteq \alpha Set \subseteq \diamond PART_p$  be a set of stable processes that wish to install view  $val' = (S, vid')$ , and  $p$  and  $q$  be any two processes in  $S$ . From Lemma 3, there exists an instant after which  $p$  and  $q$  decide the same value  $val = (S, vid)$ . From Lemma 9,  $val' = (S, vid')$  was proposed by some process  $q$  (possibly  $p$ ) in  $S \subseteq \alpha Set_p$  containing at least  $\alpha_p$  processes.  $\square$

**Theorem 3.** *Algorithm 3 implements  $\mathcal{PGM}$ : It satisfies self inclusion, local monotonicity,  $\mathcal{PGM}$ -Validity and  $\mathcal{PGM}$ -Agreement on final view.*

*Proof.* Let  $p$  be a process. From Lemma 7, if  $p$  installs view  $v'$  then  $p$  is a member of  $v$ . This satisfies the self inclusion property. From Lemma 8, if  $p$  installs view  $v$  after installing view  $v'$  then  $v.id > v'.id$ . This satisfies the local monotonicity. From Lemma 9, if a process  $p$  decides value  $val = (vset, vid)$ , then  $val' = (vset, vid')$  was proposed by a process  $q$  (possibly  $p$ ) in  $vset$  and  $vset$  contains at least  $\alpha_p$  processes. This satisfies the  $\mathcal{PGM}$ -Validity property. From Lemma 10, if there exists a set of stable processes  $S \subseteq \alpha Set \subseteq \diamond PART_p$  that wish to install view  $val = (S, vid)$ , then all the processes in  $S$  install eventually the same final view. This satisfies the  $\mathcal{PGM}$ -Agreement of final view.  $\square$