An Eventual α Partition-Participant Detector for MANETs

Léon Lim and Denis Conan
Institut Télécom, Télécom SudParis
UMR CNRS Samovar
Évry, France
firstname.lastname@telecom-sudparis.eu

***Abstract*—With Mobile Ad hoc Networks (MANETs), communication between mobile users is possible without any infrastructure. MANETs are already a necessary part of wireless systems. Due to arbitrary node arrivals, departures, crashes and movements, network partitioning may arise resulting in a degradation of the service, but not necessarily in its interruption. In this paper, we propose a distributed system model for partitionable systems. We then specify and implement a partition-participant detector that captures the liveness of a partition even if the partition is not completely stable. Its role is to detect the minimal stability condition to guarantee that eventually all the stable processes in an α-Set elect the same leader.**

***Keywords*-MANETs; models for dynamic partitionable systems; partitions; partition-participant detector; stability condition.**

I. Introduction

Mobile Ad hoc Networks (MANETs) are self-organizing networks without any fixed infrastructure. Due to node arrivals, departures, crashes and movements, MANETs are very dynamic networks. Dynamic networks are characterized by being subject to topology changes. The topology changes occur both rapidly and unexpectedly. The graph of nodes is not necessarily completely connected. Nodes can only broadcast messages to nodes that are within their transmission range. Hence, MANETs are often referred to as multihop wireless ad hoc networks: It may happen that a message sent by a node should be routed through a set of intermediate nodes. Furthermore, communication links between nodes are considered unidirectional. For instance, a node can receive a message from another node while having insufficient remaining energy to send a message back. Disconnections, failures and the mobility of nodes can isolate a node or a group of nodes from the other participants of the system. Thus, a distributed system built over MANETs may be split into partitions: Nodes that neither crash nor leave the system might not be mutually reachable. Therefore, distributed systems that are built over MANETs must be partition-tolerant so that network partitioning may result in a degradation of the service but not in its interruption.

In this paper, we focus on dynamic systems built over networks that may partition permanently [2], [3] and in which several partitions may evolve concurrently and independently from each other [6], [15], [9]. Partitions can merge into larger partitions when the communication links between them are re-established. Note that in contrast to partitionable systems, in primary-partition systems, such `merge` and `split` operations are not allowed —*i.e.*, only a single partition can exist and all the non-faulty processes are required to agree on the composition of that partition [6], [26]. Collaborative applications [11], resource allocation management [7], and distributed monitoring [23] are examples of applications that support partitioning. Partitions may experiment with some eventual stability so that the liveness of the computation can be guaranteed —*i.e.*, a stability period lasts long enough in order to eventually allow all the participant nodes of the partition to communicate in a timely manner.

In our partitionable system model, processes may leave and join the system. We consider that there are infinitely many processes, but each run has a maximum concurrency level that is finite —*i.e.,* the number of processes that have joined minus the number of processes that have left is finite. This corresponds to the *infinite arrival model with bounded concurrency* defined in [20] and investigated in [1]. In this model, the set of correct processes is not known in advance and runs can have infinitely many processes as time elapses. This uncertainty leads to two different issues: *(1)* discovering the finite set of processes that will be part of a partition and *(2)* dealing with a possibly infinite set of processes that may wake up at any time. In such a context, it would be interesting to be able to detect the existence of stable partitions in which stable processes are able to communicate with each other in a timely manner. Partition-participant detectors are oracles associated with processes that give the set of stable processes that belong to a partition. Like *failure detectors* [14], a partition-participant detector can make mistakes, but it eventually computes the set of processes that belong to the partition [8]. Note that, unlike a failure detector, the specification of a partition-participant detector must be based on the ability of processes to communicate with each other rather than detecting correct processes. The reason is that nodes may enter or leave a partition so that the set of processes is neither fixed nor known in advance.

Another motivation for our work concerns the problem of the specification of a partitionable group membership. Basically, a group membership service specifies the *view* a process has on the current partition it belongs to. Group membership is a basic building block for *partition-aware* applications that are able to make progress in multiple concurrent partitions without blocking [10]. Two prominent dynamic partitionable systems that specify partitionable group membership are presented in [9] and [15]. In these articles, the authors argue that fault-tolerant applications on top of a partitionable system usually rely on such a service. They extend the definition of the eventually perfect failure detector to partitionable systems in order to provide a membership service. These detectors eventually detect all the mutually reachable processes. They differ about their liveness property: *(1)* liveness must only hold in stable partitions [15], and *(2)* liveness must be ensured in every partition [9]. [15] defines a completely stable partition as "a set of processes that are eventually alive and connected to each other, and the link from any process in this set to any process outside the set is down". Contrarily to our work, [15] considers a static and fully connected distributed system. Futhermore, the specification of [15] does not ensure the liveness of the system when processes are forever intermittently mutually reachable. This unstable case disappears in the specification of [9] with the consideration of fair channels, and thus [9] guarantees liveness not only in stable partitions. However, as stated in [24], these two specifications are not satisfactory. For instance, the specification in [15] can be satisfied by a "trivial but useless implementation"[1] and the specification in [9] cannot be implemented without strong synchrony assumptions[2].

As a consequence, there is a critical need for the definition of a dynamic partitionable system model which is implementable and strong enough, and which guarantees the liveness property not only in completely stable partitions. In operational MANETs, the problem becomes even more complex since they can experience a wide amount of churn [27]: Nodes join and leave the system at arbitrary times and arbitrarily fast. Therefore, partitions may never be completely stable —*i.e.*, it is possible that groups of nodes are unable to progress and terminate useful distributed computations such as leader election or consensus. Nodes which stay in a partition during a long enough period of time are said to be stable, and unstable otherwise. Stable and unstable nodes can coexist in the context of a partition. Nevertheless, it is desirable to avoid that unstable nodes prevent the progress of stable ones. Thus, the stability condition should be weakened in order to allow distributed computations to terminate despite of the presence of unstable nodes in the partition. In other words, useful computations should be executed only by a set of α stable nodes as it is shown in [22]. This weak stability condition better fits to dynamic systems. However, the system model in [22] is defined for primary-partition systems in which it is not possible to have several stable partitions that run concurrently and independently.

Our Contributions. In this paper, we propose a model that characterizes the dynamic behavior of MANETs. We define a weak stability condition which guarantees the liveness of partitions even if they are not completely stable. We also propose an eventual α partition-participant detector, denoted $\diamondsuit\alpha\mathcal{PPD}$, whose role is to make trade-offs between agreement and progress by eventually detecting the stability condition. In addition, $\diamondsuit\alpha\mathcal{PPD}$ eventually determines the leader among them.

The rest of the paper is organized as follows. We discuss some related works in Section II. In Section III, we model dynamic partitionable systems and define a weak stability condition. We specify and implement the eventual α partition-participant detector and also provide its correctness proof in Section IV. We conclude the paper in Section V.

II. Related Work

Distributed models that consider dynamic systems with a stability condition (α processes) can be founded in [22], [17], [19]. In [19], the model involves unreliable failure detectors with α denoting the smallest number of stable processes in the system. A stable process is a process that is running and never suspected unless it fails. Every process that fails is eventually permanently suspected by every correct process. The model is designed for primary-partition systems. In [22], the value of α plays the role of the value $(n - f)$ in static models, where n is the total number of processes in the system and f is the maximal number of crashed processes. Like in our work, the authors state that a dynamic distributed system must present some stability period in order to guarantee progress and termination of the computation. However, the distributed system is not partitionable. In [17], the authors extend the QUERY-RESPONSE communication mechanism of [22] by considering the mobility of nodes, and propose a failure detector $\diamondsuit S^M$ 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; a process is stable if after having entered the system, it never departs. The local value of α of a process p is computed as the value of the neighborhood density of p minus the maximum

[1] The terms "useless" and "trivial" are originally highlighted in [24]. They show that their trivial but useless implementation, while satisfying the safety and liveness properties of the specification of [15], does not provide "strong" guarantees to applications. The implementation allows the following scenario: For each process p, each non singleton view is followed by a singleton view.

[2] It was observed in [24] that [9] implements a specification that is based on a time dependent property (definition of reachability) in a system model that is based on time independent property (definition of fair channels). In [9], the reachability relation is defined as follows. If p sends message m to q at time t, then q receives m if and only if q is reachable form p at time t. However, as pointed out in [24], reachability is not time invariant: Process q can be reachable from process p at time t, and unreachable at time $t' > t$.

number of faulty processes in p's neighborhood. Again, the model in [17] is designed for primary-partition systems.

[5] considers sparse MANETs where the node density is relatively low so that disconnections and network partitions are common. In such a context, the end-to-end connectivity between nodes is a temporary feature that emerges at arbitrary intervals of time. Among the set of processes S in the system, a small group [illegible] is formed for the purpose of reaching consensus. Among the $n \geq 3$ nodes of [illegible], at most f processes (with $0 < f < n/2$) can crash over the lifetime of [illegible]. Nodes in $S \setminus$ [illegible] act as routers and cooperate to discover and maintain the connectivity between nodes of [illegible]. Consensus can be solved if a majority of operative nodes exist for a long enough period of time. But, messages cannot be lost during these stable periods and partitioning cannot be permanent —*i.e*, the distributed system is a primary-partition system.

In [29], the authors aim to characterize dynamic distributed systems and consider the infinite arrival model with a maximum concurrency level b —*i.e.*, this is called the infinite arrival model with b-bounded concurrency [1]. They propose the specification of an oracle called HB^* to implement the Ω failure detector that eventually identifies the unique leader in the system. HB^* provides a list called *alive* whose length is the value of the bounded concurrency b containing processes deemed to be up in the system. HB^* embeds a stability property stating that eventually the good processes take fixed positions inside the alive list. However, even if the problem of network partitioning is considered during perturbed periods, this work caters for the eventual connectivity overlay [28] —*i.e.*, eventually, there is no partition. The model in [29] abstracts processes deployed over WANs whereas our model abstracts processes deployed over MANETs. Furthermore, the network graph in [29] is fully connected, a property that is basically assumed when considering that the network does not partition permanently [2], [3]. In contrast, we consider that the network of mobile nodes is not fully connected and that it can permanently partition —*i.e.*, paths between mobile nodes are dynamically built over time.

The notion of *heartbeat failure detector* $\mathcal{HB}$ was generalized in [2] for partitionable networks. The module $\mathcal{HB}$ outputs an array with one non-negative number for each process of the system. The heartbeat sequence numbers of the processes not in the same partition are bounded. Our partition-participant detector is inspired by this work. But, in [2], the system is static, the number of nodes of the system is known, and nodes do neither move nor leave the system.

By analogy with the concept of partition-participant detector, the notion of *participant detector* was introduced in [13] for solving the problem of bootstrapping a MANET. Participant detectors capture the minimal information that a process must have about the other participant processes to reach a consensus with unknown participants (CUP) in a fault-free scenario. The participant detector module is used for detecting the initially unknown set of processes Π in a MANET. Like in our approach, both the identity and the number of processes are initially unknown. [18] extended the work in [13] and identified the minimal synchrony assumption for solving fault-tolerant CUP (FT-CUP) and uniform FT-CUP. However, both in [13] and [18], the authors do not consider permanent partitioning of the network. The model is designed for primary-partition systems where the total number of processes is fixed and the network is always connected. In our work, not only the system is dynamic and partitionable, but the set of processes in the system is unknown in advance and the network is not fully connected.

In a previous work [8], we have proposed an eventual partition-participant detector using dynamic paths which eventually detects the participant nodes of stable partitions in MANETs. Liveness is only guaranteed in completely stable partitions. In this paper, we extend the concept of simple dynamic paths into SADDM paths which are more suitable for MANETs and provide liveness properties even if partitions are not completely stable. A SADDM path combines the lossy property of a fair link[3] [9] and the timeliness property of an eventually timely link[4] [4]. The idea of the combination of the lossy property of a fair link and the timeliness property of an eventually timely link was inspired by [25] through the notion of ADD (Average Delayed/Dropped) link. SADDM paths are dynamically built. With them, an infinite number of messages may be lost or arbitrarily delayed, but some subset of messages not too sparsely distributed in time is guaranteed to be received in a timely manner.

[30], [21] characterize group mobility and predict partitioning in location-aware MANETs. [30] proposes a velocity-based mobility model. The node velocities are assumed to be known to the server which runs a data clustering algorithm. As the authors argue, in real systems, another mechanism is required to efficiently collect the velocities from all the mobile nodes. This is a non trivial task since networks can partition. In addition, node mobility is not the only parameter that is responsible for network partitioning. Other parameters such as node failures and disconnections should also be considered. In [21], a node exchanges its location and its speed to all its one-hop neighbors and calculates the probability that a link will be broken based on distances between nodes. As in [30], only the mobility of nodes is considered. More generally, [30], [21] adopt a probabilistic partition prediction/prevention approach whereas our proposition is based on deterministic partition detection.

[3]If a message is sent from p to q an infinite number of times, and p and q are not permanently unreachable from each other, then q receives m an infinite number of times.

[4]There is a time after which all the messages that are sent are received timely.

III. Model

In this section, we define processes that communicate with each other by passing messages through wireless communication links in Sections III-A and III-B. Then, we distinguish different kinds of links, and define their properties in Section III-C. We illustrate these properties in Section III-D. Afterwards, we introduce the notions of partition and partition-participant detector in Section III-E. Finally, we introduce the stable condition and the stability criterion that ensure progress in Sections III-F and III-G, and present an example of a distributed system configuration with stable partitions in Section III-H.

A. Processes

We consider a dynamic distributed system model made up of mobile uniquely-identified nodes. We consider one process per node executing programs by taking steps. Thus the system consists of an infinite countable set of processes $\Pi = \{\dots p_i, p_j, p_k \dots\}$. Processes are also denoted p, q, r, etc.

We consider the *infinite arrival model with bounded concurrency* [20]: In any bounded period of time, only finitely many nodes take steps; the total number of nodes in a single run may grow to infinity as time passes; however, each run has a maximum concurrency level that is finite but unknown. Contrarily to static systems, in dynamic systems, processes do not know Π, i.e., the processes in Π do not necessarily know each other. A correct process never fails. A faulty process fails by crashing (and as a consequence leaves the system). Correct processes may leave and join the system.

In order to simplify the presentation without loss of generality, we assume the existence of a global clock which is not accessible by the processes. We take the range $\mathcal{T}$ of the clock's tick to be the set of natural numbers $\mathbb{N}$.

B. Links

We assume the MANET communication model in which nodes do not send point-to-point messages but broadcast messages that will be 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 also assume that q receives a message m from p at most once (*no duplication*) and only if p previously broadcast m (*no creation*).

C. 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. Finally, a *forever intermittently up* link can lose messages it transports arbitrarily. In our model, forever intermittently links are the root of the instability of the system. They result in two processes being forever intermittently mutually reachable. In the worst case, no useful computations can be terminated if two processes p and q are not mutually reachable at those times at which they attempt to communicate with each other. To avoid this bad scenario, we assume the fairness property on communication links between processes that are at least forever intermittently reachable (including processes that are eventually forever reachable). Similarly to [9], a fair link is defined as follows.

Definition 1. Fair link. *A link $p \rightsquigarrow q$ is said to be fair if p broadcasts a message m to q an infinite number of times and q is correct, then q receives m from p an infinite number of times.*

Observe that eventually up links and forever intermittently up links are both captured into fair links. Thus, in the sequel, we do not distinguish them.

We denote 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, as a fair path from p_1 to p_n, denoted $\textsc{fair}(p_1 p_2 \dots p_n)$.

By definition, a fair link can lose messages due to communication failures. The *reachability* relation captures these communication failures. With fair links, reachability is defined similarly to [2] and as suggested in [24].

Definition 2. 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 $p \rightharpoondown q$, that is $p \rightharpoondown q \stackrel{def}{\equiv} \textsc{fair}(p \dots q)$.*

If process q is reachable from process p, and vice-versa, we write $p \leftrightharpoons q$. We also say that p and q are mutually reachable. Notice that the definition of fair link is time independent and that our definition of reachability, contrary to the definition of [9], is also time independent.

A link that intermittently loses messages may satisfy the fairness property. However, it is worth pointing out that 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 to compute and terminate a useful computation during that period. Thus, we make the additional assumption that follows. We define the concept of SADDM (Simple Average Delayed/Dropped of a Message) links[5] and SADDM paths where communication delays between two processes are bounded during stable periods. A SADDM link allows messages to be lost or arbitrarily delayed, but guarantees that some subset of the messages sent on the link will be received in a timely manner. In addition, such messages are not too sparsely

[5]This concept of SADDM link is inspired by [25] through the notion of ADD link (named *channel* in that paper).

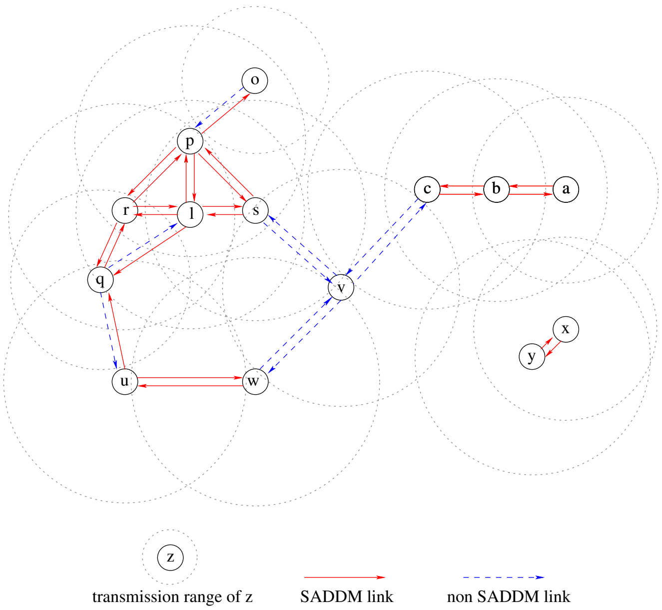

Figure 1. SADDM links and SADDM paths

distributed in time. A SADDM link is defined as follows.

Definition 3. SADDM link. *Let β and δ be two constants, and $[t_1, t_2]$ be a finite time interval during which p broadcasts a message m to q at least β times. A link $p \rightsquigarrow q$ is said to be a SADDM link if q receives m at least once by time $t_1 + \delta$, with $\delta > t_2 - t_1$.*

We now define a SADDM path as a sequence of SADDM links as follows.

Definition 4. SADDM path. *A sequence of processes $(p_1 \dots p_n)$ is a SADDM path if $\forall i \in [1, n-1]$ the link $p_i \rightsquigarrow p_{i+1}$ is SADDM and $i \neq j \implies p_i \neq p_j$.*

This definition means that there exists a constant c such that, for each finite time interval $\phi = [\tau_1, \tau_2]$ during which p_1 broadcasts a message m to p_n at least β^c times, p_n receives at least one of these messages through $\text{SADDM}(p_1 p_2 \dots p_n)$ by time $\tau_1 + \beta^c + c\delta$, with $\beta^c + c\delta > \tau_2 - \tau_1$. The constants β and δ are the same as the ones used in the definition of a SADDM link. Since the concurrency is bounded, $\beta^c + c\delta$ is bounded. A SADDM path from process p to process q is denoted by $\text{SADDM}(p \dots q)_\phi$ with respect to an interval ϕ. The subscript interval ϕ on the path is omitted when it is unambigous.

D. Illustrative example of SADDM paths

The notions of SADDM links and SADDM paths are presented in Figure 1. Dashed circles are used to represent the transmission range of processes. Solid arrows correspond to SADDM links, otherwise arrows are dashed. Processes u and w can communicate with each other in a timely manner since there exist SADDM links from u to w and from w to u. This is the same for processes x and y. In addition, processes p, q, r, s and l can communicate with each other in a timely manner since there exists a SADDM path between any two of them. This is also the case for the set of processes $\{a, b, c\}$. Process u cannot communicate with any process in $\{p, q, r, s, l\}$ since there is no SADDM path from a process in this set to u. This is also the case for processes o and v.

E. Partition and Partition-Participant Detectors

The network is partitionable. Several disjoint sub-sets of processes may co-exist such that the processes in each subset are mutually reachable and processes in two sub-sets are not mutually reachable. $\leftrightharpoons$ is an equivalence relation and *partitions* are defined by equivalence classes of this relation. The partition of process p is denoted $PART_p = \{q \in \Pi | p \leftrightharpoons q\}$.

In order to ensure that a useful computation can progress and terminate, a partition has to satisfy some form of eventual stability. Processes in partitions that present some eventual stability are called *stable processes*. A process is stable in the context of some partition. A *partition pattern* is a function $\mathbb{P} : \Pi \times \mathcal{T} \to 2^\Pi$, where $\mathbb{P}(p, t)$ denotes the set of processes that p believes to be in its partition at time t. A process can join and leave a partition arbitrarily. Thus, the function $\mathbb{P}$ is not necessarily monotonic in time. Similarly to failure detectors [14], partition-participant detectors are distributed oracles associated with each process. The failure detector proposed in [14] is for primary-partition systems in which every pair of processes in the fixed and known set Π is connected by a reliable communication channel. The failure detector is used to state which processes are in Π, that is which processes are correct: A process is correct if it is not suspected to have crashed by any process in a failure pattern. Differently to failure detectors, in partitionable systems, due to processes entering and leaving a partition, and since Π is neither fixed nor known in advance, the specification of a partition-participant detector has to be based on the ability of processes in a partition to communicate with each other rather than individual processes being correct or crashed.

F. Stability Property and Stability Condition

In primary-partition systems made up of $|\Pi|$ processes, if there exists a majority of processes that can communicate with each other during a long enough period of time, then the system is said to be stable for that period [12]. By analogy, in partitionable systems, a partition becomes stable during a period when all the correct processes in that partition can communicate with each other during that period. So far, we have not defined such a period. To do so, as in [22], let us define a time interval $\Delta = [t_b, t_e]$ 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. Practically speaking, the execution of the application may also be divided into phases, the phases then becoming the periods. In every stable period Δ, stable processes can communicate through SADDM paths. The stable partition associated with Δ is denoted by $\Delta PART_p$, and is defined as follows.

Definition 5. Stable Partition Per Period. *The stable partition per period Δ of process p is the set of all the correct processes q, denoted $\Delta PART_p$, such that there exist at least a SADDM path from p to q and a SADDM path from q to p.*

To simplify the presentation, we consider only one period in the rest of the paper. During a period, the stable property is a property that remains true once it holds —*i.e.*, it holds after the *stabilization time* ST_p which is unknown to the processes. A stable partition associated to some process p is then denoted by $\diamondsuit PART_p$ which is defined as follows.

Definition 6. Stable Partition. *The stable partition of process p is the set of correct processes q, denoted $\diamondsuit PART_p$, such that there exists a time ST_p after which* $\textsc{saddm}(p \ldots q)$ *and* $\textsc{saddm}(q \ldots p)$ *exist.*

The validity period of the definition of a stable partition is the duration of an execution —*i.e.*, in practice, a process is stable if the SADDM paths exist long enough for the algorithm to terminate. Then, we define the concept of a stable process in the context of a stable partition as follows.

Definition 7. Stable Process. *For any two correct processes p and q, if there exists a time t after which process $q \in \diamondsuit PART_p$, then q is stable in $\diamondsuit PART_p$.*

In the sequel of the paper, by a short abuse of language, q of the previous definition will be said to be a stable process, without mentioning the name of the partition, such that we will say that q is stable.

A partition may never be completely stable —*i.e.*, $\diamondsuit PART_p$ may never exist. Such a behavior can prevent the progress of processes in a partition, and in the worst case, can block the system. The issue is then to weaken the stability property and allow useful computations to terminate with safety guarantees. Since stable and unstable nodes can coexist in the context of a partition, it is desirable to avoid unstable nodes from preventing the progress of stable ones. So, useful computations should be executed only by a set of α processes among the stable processes. The intuition is that α expresses the trade-off between agreement and progress. It is up to the application to provide an appropriate value of α through another service. α is a parameter of the partition-participant detector module. It is the responsibility of the application to choose a suitable value of α —*i.e.,* the minimum number of participants. Thus, we define the stability condition associated to a process p as follows.

Definition 8. Stability Condition. $|\diamondsuit PART_p| \geq \alpha_p$.

Stable processes are forever mutually reachable after the minimal stabilization time ST_p. Unfortunately, processes cannot know ST_p. Then, since all the nodes have different battery power, bandwidth capability, mobility behavior, etc., only a sub-set of mutually reachable nodes is selected to take part to the computation. The participating members are selected among mutually reachable nodes by some *stability criterion* to form a partition that may possibly satisfy the stability condition. We define in the next section a stability criterion for selecting the nodes that are detected to be "enough stable".

G. Stability Criterion

A stability criterion is a parameter that is used to determine which nodes are the most stable ones, the ones that may be "marked" as stable. We call this kind of set of nodes a *tentative set*. Application designers may choose different stability criteria per application execution. Furthemore, the choice of the appropriate parameter will be influenced by the needs of the application.

In this paper, we choose the time-based stability criterion $hb_p^q \geq threshold_p$, with hb_p^q being a function that depends on the number of heartbeat messages received by p from q (hb_p^q increases if q is present in p's partition and decreases otherwise), and with $threshold_p \geq 1$. q is marked as stable by p if $hb_p^q \geq threshold_p$, and is removed from p's tentative set if $hb_p^q = 0$ —*i.e.*, p does not receive any heartbeat from q anymore. With this stability criterion, we can eliminate a node from participating if it disappears while tolerating sporadic disconnections. In addition, heartbeat counters are also used to state whether processes are currently mutually reachable.

H. Illustrative example of stable partitions

In Figure 2, we complement Figure 1 to illustrate the definitions of stable process, stability condition, and stable partition. Black disks represent unstable nodes whereas white disks depict stable processes. Each stable partition is enclosed by a solid circle. There are eventually fives stable partitions $\diamondsuit PART_o, \diamondsuit PART_p, \diamondsuit PART_w, \diamondsuit PART_a$ and $\diamondsuit PART_x$, with their value of α equals to 1, 4, 2, 3 and 2, respectively. Processes can move inside the stable partition.

Remark that stable partitions are not necessarily isolated from other nodes of the network: All the links from any process in a stable partition to any process outside the partition are not necessarily down. For instance, process u in $\diamondsuit PART_w$ can receive messages broadcast by processes in $\diamondsuit PART_p$ in a timely manner through some SADDM paths, but processes in $\diamondsuit PART_p$ cannot receive messages broadcast by u in a timely manner since there is no SADDM path from any process in $\diamondsuit PART_w$ to q. Therefore, u is unstable in the context of $\diamondsuit PART_p$, but is stable in the context of $\diamondsuit PART_w$.

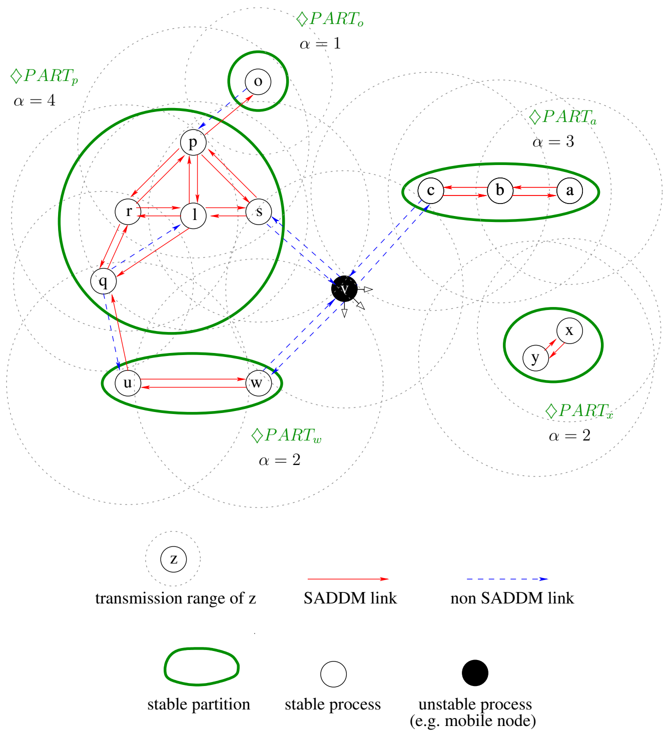

Figure 2. Stable partitions and their stability condition

IV. Eventual α Partition-Participant Detector

In this section, we develop the specification and give an implementation of the eventual α partition-participant detector.

A. Specification

An *eventual α partition-participant detector* $\Diamond\alpha\mathcal{PPD}$ is a distributed oracle that eventually detects the set of stable processes α-Set in a partition. The processes in α-Set are chosen according to the stability criterion $hb_p^q \geq threshold_p$. $\mathcal{T}\forall\in\mathcal{T}\{\mathcal{T}|\Rightarrow\notin\langle\rangle\mathcal{T}\rangle\Rightarrow\notin\Rightarrow\forall\mathcal{D}\Rightarrow\mathcal{T}\notin\forall\not\Rightarrow\mathcal{T}\notin\langle\backslash\rangle\notin\langle\backslash\not\Rightarrow\mathcal{T}\{\langle\Rightarrow\Rightarrow\{\Rightarrow\mathcal{T}\notin\mathcal{T}\mathcal{T}\Rightarrow\Rightarrow\mathcal{T}\Rightarrow\{\Rightarrow\mathcal{T}\notin\mathcal{T}\mathcal{T}\Rightarrow\Rightarrow\backslash$

stable processes according to p's local stability criterion.

Despite the fact that processes may initially have different values of α and αSet, the objective of the algorithm is threefold: *(1)* eventually all the processes in αSet_p have the same value αSet_p, *(2)* the value of αSet_p is eventually αSet_l with l being the leader among αSet_p, and thus *(3)* eventually all the processes in αSet_p elect the same leader $l \in \alpha Set_p$. αSet_p is computed as the set of processes q in $tentative_p$ for which $\alpha_q \leq \alpha_p$. αSet_p serves to compute the output of the algorithm (Line 70). Note that, for all the processes u and v, α_u and α_v do not have to be equal. A process q takes part to the construction of αSet_p if $|\alpha Set_p| > \alpha_q$. The idea is that "potential" leader processes try to convince other processes in their stable partition to agree with their value of α-Set. But, only the stable process with the highest value of α eventually succeeds.

We now describe the five main tasks that the algorithm executes. In Task 1, process p repeatedly broadcasts a HEARTBEAT message with a bootstrap path (p, α_p) —*i.e.,* $\langle \text{HEARTBEAT} \mid (p, \alpha_p)\rangle$, to announce that it is alive and present.

In Task 2, upon expiration of ${}_{part}\mathsf{timer}$, p checks the set of processes that p believes to be stable in its partition. If there are one or more processes in αSet_p which are no more stable ($\alpha Set_p \not\subseteq tentative_p$, Line 21) or if the stability condition is not satisfied ($|\alpha Set_p| < \alpha_p$), then p re-computes αSet_p (Line 22) as the set of processes having reached a given level of stability ($hb_p^q \geq threshold_p$). If the stability condition is reached ($|\alpha Set_p| \geq \alpha_p$, Line 23) and if p believes itself to be the leader (Line 24), then p tries to convince the other processes in its α-Set to agree on its value of αSet_p by broadcasting a message $\langle \text{ALPHASET} \mid p, \alpha Set_p\rangle$ (Line 25). Otherwise, p increments its timer value ${}_{part}timeout$. Finally, p prepares itself for the next execution of Task 2 (Line 28–29).

In Task 3, upon the reception of the message $\langle \text{ALPHASET} \mid q, \alpha Set_q\rangle$, p verifies if *(1)* p and q are mutually reachable, and *(2)* q is the leader of p (Line 34). If this is the case, p adopts the value of αSet_q for its local variable αSet_p (Line 35). As a consequence, p and q both believe that there exist more than $\alpha_q \geq \alpha_p$ stable processes in p's and q's partition. p re-broadcasts the message $\langle \text{HEARTBEAT} \mid q, \alpha Set_q\rangle$ so that the wave can reach the other processes in the partition (Line 36).

In Task 4, upon the reception of a message $\langle \text{HEARTBEAT} \mid path\rangle$, if $path$ begins with the tuple (p, α_p), then p knows that one of its messages $\langle \text{HEARTBEAT} \mid (p, \alpha_p)\rangle$ has passed through a cycle —i.e., each node q in the tuple that appears after (p, α_p) in $path$ is mutually reachable from p (Lines 42). When p sees q for the first time (Line 43), p creates ${}_{proc}\mathsf{timer}_p^q$ and its corresponding timeout value ${}_{proc}timeout_p^q$, and sets ${}_{proc}\mathsf{timer}_p^q$ to ${}_{proc}timeout_p^q$ (Line 53). ${}_{proc}\mathsf{timer}_p^q$ is reset to ${}_{proc}timeout_p^q$ every time a message $\langle \text{HEARTBEAT} \mid path\rangle$ has gone through a cycle from p. If q was previously reachable from p (Line 48), and is not already taking part to the construction of αSet_p ($(q, \alpha_q) \notin tentative_p$, Line 49), then p starts considering q as a potential stable process and adds q to $tentative_p$ with a heartbeat counter assigned to 1 (Line 50). If q already takes part to the construction of αSet_p ($(q, \alpha_q, hb_p^q) \in tentative_p$, Line 51), p increments q's heartbeat counter —i.e., q is getting "more" stable according to the stability criterion (Line 52). If $path$ does not begin with (p, α_p) and if p does not appear in $path$ or appears just once, p appends (p, α_p) to $path$ and broadcasts a HEARTBEAT message with $newpath = path \circ (p, \alpha_p)$ (Lines 55–56). Observe that, as described in [2], p must forward the message even if it already appears once in $path$ since it might be the case that there exists a cycle between q and r where p belongs both to the path from q to r and to the path from r to q.

In Task 5, upon expiration of ${}_{proc}\mathsf{timer}_p^q$, p decrements q's heartbeat counter —i.e., q is getting "less stable" (Line 63). When q's heartbeat counter reaches zero, q is removed from $tentative_p$ —i.e., q can no more be considered as a stable process and should no more participate to the construction of αSet_p (Line 61). ${}_{proc}\mathsf{timer}_p^q$ expires means that the value of ${}_{proc}timeout_p^q$ is not enough for a message $\langle \text{HEARTBEAT} \mid path\rangle$ to travel along a cycle (including q) from p. Therefore, ${}_{proc}timeout_p^q$ is incremented. Observe that, in order to exclude from αSet_p processes that are too unstable, ${}_{proc}timeout_p^q$ can equal to, but cannot exceed ${}_{part}timeout$ (Line 64).

Finally, by querying its local partition-participant detector (Lines 68–71), a client application obtains the identifier of the current leader l in the partition and the set of stable processes, that is $\alpha Set_p = \alpha Set_l$.

C. Proof of Correctness of the Implementation

We now show that Algorithm 1 implements $\diamond\alpha\mathcal{PPD}$.

Lemma 1. *Let p_1 be a stable process and p_n be a process in $\diamond PART_{p_1}$ such that there exists a SADDM path ${}_{\text{SADDM}}(p_1p_2\dots p_n)$ from p_1 to p_n. Eventually one of the β^{n-1} messages $\langle \text{HEARTBEAT} \mid (p_1, \alpha_{p_1})\rangle$ broadcast by p_1 reaches p_n in at most $\beta^{n-1}\eta + (n-1)\delta$ seconds.*

Proof: Let p_1 be a stable process and p_n be a process in $\diamond PART_{p_1}$ such that there exists a SADDM path ${}_{\text{SADDM}}(p_1p_2\dots p_n)$ from p_1 to p_n. Let $\Sigma = {}_{\text{SADDM}}(p_1p_2\dots p_n)$. To simplify the presentation of the proof, a path will be regarded as a sequence of processes —*i.e.*, we don't consider the value of α associated to each process that is present in the variable $path$ of Algorithm 1. By definition of SADDM path, each process p_i, for $i \in [1, n]$, appears at most once in Σ. For $j \in [1, n]$, let $P_j = {}_{\text{SADDM}}(p_i)_{i\in[1,j]}$. To prove the lemma, we show by induction that $\forall j \in [1, n-1]$, at least one of the β^{j-1} messages

Algorithm 1 Implementation of $\Diamond\alpha\mathcal{PPD}$ for process p

init():
Begin
$\alpha_p \leftarrow n$ with $n \geq 1$; {Minimum number of stable processes required by the application}
$\alpha Set_p \leftarrow \{(p, \alpha_p)\}$; {Stable processes with their value of α}
$threshold_p \leftarrow c$; {Minimal value for being stable according to the stability criterion}
$maxhb_p \leftarrow hb$ with $hb \geq threshold_p$; {Maximum value that a heartbeat counter can have}
${}_{part}timeout \leftarrow t \geq 1$; set ${}_{part}\mathsf{timer}$ to ${}_{part}timeout$; {Timer used for checking the stability condition}
${}_{proc}timeout \leftarrow \{\}$; ${}_{proc}\mathsf{timer} \leftarrow \{\}$; {Sets of pairs $(process, timeout)$ and $(process, \mathsf{timer})$, respectively}
$mreachable_p \leftarrow \{(p, \alpha_p)\}$; {Mutually reachable processes with their respective value of α}
$previous_p \leftarrow \{(p, \alpha_p)\}$; {Previous value of $mreachable_p$}
$tentative_p \leftarrow \{(p, \alpha_p, maxhb_p)\}$; {Set of tuples $(process, \alpha, heartbeat\_nb)$ for constituting αSet_p}
End

Task T1: **every** η **seconds** {Broadcasting heartbeats}
Begin
broadcast$_{nbg}$($\langle$HEARTBEAT $\mid (p, \alpha_p)\rangle$);
End

Task T2: **upon expiration of** ${}_{part}$**timer** {Checking the stability of an α-Set}
Begin
If $(\alpha Set_p \nsubseteq tentative_p \vee |\alpha Set_p| < \alpha_p)$ then
$\alpha Set_p \leftarrow \{(q, \alpha_q) | (q, \alpha_q, hb_p^q) \in tentative_p \wedge hb_p^q \geq threshold_p\}$;
If $|\alpha Set_p| \geq \alpha_p$ then
If $p = r \ : \ (r, \alpha_r) \in \alpha Set_p \wedge [\forall (s, \alpha_s) \in \alpha Set_p, \alpha_r > \alpha_s \vee (\alpha_r = \alpha_s \wedge r > s)]$ then
broadcast$_{nbg}$($\langle$ALPHASET $\mid p, \alpha Set_p\rangle$);
Else
${}_{part}timeout \leftarrow {}_{part}timeout + 1$; {There are not enough stable processes, thus increase the detection period}
$previous_p \leftarrow mreachable_p$;
set ${}_{part}\mathsf{timer}$ to ${}_{part}timeout$;
End

Task T3: **upon reception of** $\langle$ALPHASET $\mid q, \alpha Set_q\rangle$ {Verifying the leader's α-Set}
Begin
If $\alpha Set_p \subseteq \alpha Set_q$ then
$\alpha Set_p \leftarrow \alpha Set_q$;
broadcast$_{nbg}$($\langle$ALPHASET $\mid q, \alpha Set_q\rangle$);
End

Task T4: **upon reception of** $\langle$HEARTBEAT $\mid path\rangle$ {Detecting mutually reachable processes}
Begin
If first tuple in $path$ is $(p, *)$ then
For all (q, α_q) : (q, α_q) appears after the first tuple in $path \wedge q \neq p$ do
If $(q, *) \notin mreachable_p$ then
$mreachable_p \leftarrow mreachable_p \setminus \{(q, *)\} \cup \{(q, \alpha_q)\}$;
${}_{proc}\mathsf{timer} \leftarrow {}_{proc}\mathsf{timer} \cup \{(q, {}_{proc}\mathsf{timer}_p^q);\}$ {Dynamic creation of timer and timeout for new process q}
${}_{proc}timeout_p^q \leftarrow 1$; ${}_{proc}timeout \leftarrow {}_{proc}timeout \cup \{(q, {}_{proc}\,timeout_p^q)\}$;
Else
If $(q, \alpha_q) \in previous_p$ then
If $(q, \alpha_q, hb_p^q) \notin tentative_p$ then
$tentative_p \leftarrow tentative_p \cup \{(q, \alpha_q, 1)\}$;
Else
$tentative_p \leftarrow tentative_p \setminus \{(q, *, *)\} \cup \{(q, \alpha_q, \max(hb_p^q{+}1, maxhb_p)\}$;
set ${}_{proc}\mathsf{timer}_p^q$ to ${}_{proc}timeout_p^q$; {Resetting timer when receiving HEARTBEAT}
Else
If (p, α_p) appears at most once in $path$ then {p may belong to $\textsc{saddm}(q \ldots r)$ and $\textsc{saddm}(r \ldots q)$}
broadcast$_{nbg}$($\langle$HEARTBEAT $\mid path \circ (p, \alpha_p)\rangle$);
End

Task T5: **upon expiration of** ${}_{proc}$**timer**$_p^q$ {Computing heartbeats per period}
Begin
If $hb_q^p = 1 \ : \ (q, *, hb_q^p) \in tentative_p$ then $tentative_p \leftarrow tentative_p \setminus \{(q, *, *)\}$;
Else
$tentative_p \leftarrow tentative_p \setminus \{(q, *, *)\} \cup \{(q, *, hb_p^q{-}1)\}$;
${}_{proc}timeout_p^q \leftarrow \min({}_{proc}timeout_p^q + 1, {}_{part}\,timeout)$;
${}_{proc}timeout \leftarrow {}_{proc}timeout \setminus \{(q, *)\} \cup \{(q, {}_{proc}\,timeout_p^q)\}$;
End

Procedure participants()
Begin
Return $(l, \alpha Set_p) \ : \ (l, \alpha_l) \in \alpha Set_p \wedge [\forall (r, \alpha_r) \in \alpha Set_p, \alpha_l > \alpha_r \vee (\alpha_l = \alpha_r \wedge l > r)]$;
End

$\langle \mathsf{HEARTBEAT} \mid P_{j-1} \rangle$ originally broadcast by p_1 reaches p_j in at most $\beta^{j-1}\eta + (j-1)\delta$ seconds.

For the base case ($j = 1$), by Task T1, p_1 permanently broadcasts $\langle \mathsf{HEARTBEAT} \mid P_1 \rangle$ every η seconds to all its neighbours, and thus to p_2. As the path $(p_1 p_2)$ is a SADDM path, then at least one of the β messages broadcast by p_1 is received by p_2 in at most $\beta\eta + \delta$ seconds. This shows the base case. For the induction step, let $j \leq n-1$ and assume that at least one of the β^{j-1} messages $\langle \mathsf{HEARTBEAT} \mid P_{j-1} \rangle$ originally broadcast by p_1 is received by p_j in at most $\beta^{j-1}\eta + (j-1)\delta$ seconds. Since the path $(p_j p_{j+1})$ is a SADDM path, p_{j+1} receives at least one of the $\beta\beta^{j-1} = \beta^j$ messages $\langle \mathsf{HEARTBEAT} \mid P_1 \rangle$ in at most $\beta^j\eta + j\delta$ seconds. Moreover, p_{j+2} appears at most once in P_{j+1} and p_{j+2} is a neighbour of p_{j+1}. So, each time p_{j+1} receives $\langle \mathsf{HEARTBEAT} \mid P_j \rangle$, it re-broadcasts $\langle \mathsf{HEARTBEAT} \mid P_{j+1} \rangle$ to p_{j+2} by appending itself to P_j (Line 56). Therefore, since $(p_{j+1} p_{j+2})$ is a SADDM path, p_{j+2} receives at least one of the $\beta\beta^j = \beta^{j+1}$ messages $\langle \mathsf{HEARTBEAT} \mid P_{j+1} \rangle$ originally broadcast by p_1 in at most $\beta^{j+1}\eta + (j+1)\delta$ seconds. This shows the induction step. Therefore, we conclude that eventually at least one of the β^{n-1} messages $\langle \mathsf{HEARTBEAT} \mid P_1 \rangle$ broadcast by p_1 reaches p_n in at most $\beta^{n-1}\eta + (n-1)\delta$ seconds. ∎

Lemma 2. *Let p be a stable process. There is a time after which $hb_p^q = maxhb_p$ and $hb_q^p = maxhb_q$ remain true, $\forall q \in \Diamond PART_p$.*

Proof: Let p be a stable process and q be a process in $\Diamond PART_p$. Let $\Sigma 1 = {\scriptstyle \text{SADDM}}(p_i)_{i\in[1,k]}$ be a SADDM path from p to q, and $\Sigma 2 = {\scriptstyle \text{SADDM}}(p_i)_{i\in[k,n]}$ be a SADDM path from q to p. We consider now the path $\Sigma = {\scriptstyle \text{SADDM}}(p_i)_{i\in[1,n]}$ which is the concatenation of $\Sigma 1$ and $\Sigma 2$ —*i.e.*, $\Sigma = \Sigma 1 \circ \Sigma 2$.

By definition of SADDM path, $\forall i \in [1,k]$ and $\forall i \in [k,n]$, each process p_i appears at most once in $\Sigma 1$ and $\Sigma 2$, respectively, and at most twice in Σ. By construction, $p_1 = p_n = p$, and $p_k = q$. For $j \in [1,n]$, let $P_j = {\scriptstyle \text{SADDM}}(p_i)_{i\in[1,j]}$. To prove that $hb_p^q = maxhb_p$, we can use the Lemma 1. The proof of $hb_q^p = maxhb_q$ can be done in the same way by considering the SADDM path Σ' equals to $\Sigma 1'$ concatenated with $\Sigma 2'$, where $\Sigma 1' = {\scriptstyle \text{SADDM}}(p_i)_{i\in[k,n]}$ and $\Sigma 2' = {\scriptstyle \text{SADDM}}(p_i)_{i\in[1,k]}$, from q to p. Since the two proofs are quite similar, only one of them ($hb_p^q = maxhb_p$) is described here.

The proof of $hb_p^q = maxhb_p$ is as follows. From Lemma 1, eventually at least one of the β^{n-1} messages $\langle \mathsf{HEARTBEAT} \mid P_{n-1} \rangle$ originally broadcast by p_1 reaches p_n in at most $\beta^{n-1}\eta + (n-1)\delta$ seconds. When p_n receives a message HEARTBEAT, $\forall i \in [2, n-1]$, ${}_{proc}\mathsf{timer}_{p_n}^{p_i}$ is reset to ${}_{proc}timeout_{p_n}^{p_i}$ (Line 53) such that ${}_{proc}\mathsf{timer}_{p_n}^{p_i}$ does not expire. Otherwise, ${}_{proc}timeout_{p_n}^{p_i}$ eventually gets incremented over the value $\beta^{n-1}\eta + (n-1)\delta$. Hence, eventually p always receives heartbeat messages $\langle \mathsf{HEARTBEAT} \mid P_{n-1} \rangle$ with p being the first element of the path P_{n-1} before ${}_{proc}\mathsf{timer}_p^q$ expires. Since q appears after p in P_{n-1}, $(q, *) \in previous_p$ (Line 48) and $(q, *, *) \in tentative_p$ (Lines 49–52), p increments $hb_{p_n}^q$ up to $maxhb_p$. ∎

Lemma 3. *Let p be a stable process. There is a time after which if $q \in \alpha Set_p$ permanently, then $q \in \Diamond PART_p$.*

Proof: Let p be a stable process. We show by contraposition that for every process $q \notin \Diamond PART_p$ there does not exist a time after which $q \in \alpha Set_p$ permanently. From Lemma 2, for every process $q \notin \Diamond PART_p$, there does not exist a time after which $hb_p^q = maxhb_p$ and $hb_q^p = maxhb_q$ hold, so that Lines 61 and 63 of Algorithm 1 do not stop being executed. Therefore, every process $q \notin \Diamond PART_p$ that is added to the set $tentative_p$ keeps being removed from it. αSet_p is computed as the set of the processes in $tentative_p$ (Line 22). Hence, there does not exist a time after which $q \in \alpha Set_p$ permanently. ∎

Lemma 4. *Let p be a stable process and q be a process in $\Diamond PART_p$. There is a time after which $\alpha Set_p \subseteq tentative_p$ remains true.*

Proof: From Lemma 3, $q \in \alpha Set_p$ permanently means that $q \in \Diamond PART_p$. Consider the time after which $hb_p^q = maxhb_p$. Thanks to Lemma 2, such a time exists after which q is added to $tentative_p$ and not removed afterwards. Since the set αSet_p is computed from the set $tentative_p$, $\alpha Set_p \subseteq tentative_p$ eventually remains true. ∎

Lemma 5. *Let p be a stable process such that $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. There is a time after which p is the only process in αSet_p that broadcasts some messages* ALPHASET.

Proof: Let p be a stable process such that $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. By definition of the stability condition —*i.e.*, $|\alpha Set_p| \geq \alpha$— and from Lemma 4, the condition at Line 23 is always true. Hence, p periodically broadcasts messages $\langle \mathsf{ALPHASET} \mid p, \alpha Set_p \rangle$. We can show by contradiction that none of the processes $(q \neq p) \in \alpha Set_p$ eventually keeps broadcasting ALPHASET messages. Let us suppose that there exists a process q in αSet_p such that $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q)$, and there does not exist a time after which q stops broadcasting ALPHASET messages. By Task T2, q broadcasts ALPHASET messages when the conditions at Lines 21, 23 and 24 hold, that is $\alpha Set_q \not\subseteq tentative_q$. Since $p \in tentative_q$ and $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q)$, the condition $\alpha Set_q \not\subseteq tentative_q$ can be true only if $hb_q^p < threshold_q \leq maxhb_q$, leading to a contradiction with Lemma 2. ∎

Lemma 6. *Let p be a stable process such that $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. There is a time after which $\alpha Set_q = \alpha Set_p$ remains true.*

Proof: Let p be a stable process such that $\alpha_p > \alpha_q \vee$

$(\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. By Lemma 5, there is a time after which p is the only process that periodically broadcasts ALPHASET messages. By Task T3, every time q receives the message $\langle \text{ALPHASET} \mid p, \alpha Set_p \rangle$, q adopts the value of αSet_p for its local variable αSet_q since $\alpha Set_q \subseteq \alpha Set_p$ (thanks to Lemma 2 and to $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$). Moreover, thanks to Lemma 4, $\alpha Set_q \subseteq tentative_q$ is eventually always true. Therefore, q eventually keeps its set αSet_q unchanged. Hence, there is a time after which $\alpha Set_q = \alpha Set_p$ remains true. ■

Lemma 7. *Let p be a stable process. There is a time after which $\Diamond\alpha\mathcal{PPD}_q$ at node q always outputs $(l, \alpha Set_l)$ such that $(l, \alpha_l) \in \alpha Set_p \wedge [\forall (r, \alpha_r) \in \alpha Set_p, \alpha_l > \alpha_r \vee (\alpha_l = \alpha_r \wedge l > r)]$.*

Proof: Let p be a stable process. By Lemma 6, there is a time after which $\alpha Set_q = \alpha Set_p$ remains true. Since the leadership function is the same for all the processes, there is a time after which $\Diamond\alpha\mathcal{PPD}_q$ at every process $q \in \alpha Set_p$ outputs $(l, \alpha Set_l)$ such that $(l, \alpha_l) \in \alpha Set_p \wedge [\forall (r, \alpha_r) \in \alpha Set_p, \alpha_l > \alpha_r \vee (\alpha_l = \alpha_r \wedge l > r)]$. ■

Theorem 1. *$\Diamond\alpha\mathcal{PPD}$ satisfies properties P1, P2, and P3.*

Proof: Consider a stable process p such that $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \alpha Set_p$. From Lemma 6, there is a time after which $\alpha Set_q = \alpha Set_p$ remains true. Hence, eventually all the processes in αSet_p have the same set $\alpha Set = \alpha Set_p$. This satisfies P2. From Lemma 7, for each stable process q in αSet_p, the module $\Diamond\alpha\mathcal{PPD}_q$ outputs p. So, p is the leader that is eventually elected by all the stable processes in αSet. This satisfies P3. By properties P2 and P3, P1 is trivially satisfied. ■

V. Conclusion

In this paper, we propose a model that characterizes the dynamic behavior of stable partitions in MANETs. To this means, we have defined a weak stability condition based upon the application-dependent parameter α. α is a threshold value used to capture the liveness property of a partition. In each partition, at least α stable processes execute distributed computations. In order to be part of this set, nodes are selected by using the stability criterion $hb_p^q \geq threshold_p$: A node is removed from participating if it disappears while tolerating sporadic disconnections. In addition, we have presented an eventual α partition-participant detector $\Diamond\alpha\mathcal{PDD}$ whose role is to detect the stability condition and to guarantee that eventually all the processes in an α-Set elect the same leader.

Using $\Diamond\alpha\mathcal{PDD}$ as a building block, we are specifying and designing a group membership service for partionable networks over MANETs. We plan to evaluate by simulation $\Diamond\alpha\mathcal{PDD}$ using different mobility models. We also want to study other stability criteria that may be elicited [16].

Acknowledgments

We are grateful to Miguel Correia, the EDCC-2012 PC Chair, for his accompaniment in the review process and to anonymous reviewers for their comments which helped us to improve this paper.

References

[1] M. K. Aguilera, "A pleasant stroll through the land of infinitely many creatures," *SIGACT News*, vol. 35, no. 2, pp. 36–59, 2004.

[2] M. K. Aguilera, W. Chen, and S. Toueg, "Using the Heartbeat Failure Detector for Quiescent Reliable Communication and Consensus in Partitionable Networks," *Theoretical Computer Science*, vol. 220, no. 1, pp. 3–30, Jun. 1999.

[3] ——, "On Quiescent Reliable Communication," *SIAM Journal of Computing*, vol. 29, no. 6, pp. 2040–2073, Apr. 2000.

[4] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "On implementing omega with weak reliability and synchrony assumptions," in *Proceedings of the 22nd ACM Symposium on Principles of Distributing Computing*, Boston, Massachusetts, USA, 2003, pp. 306–314.

[5] K. Alekeish and P. Ezhilchelvan, "Consensus in Sparse, Mobile Ad Hoc Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 467–474, Jun. 2011.

[6] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg, "On the Formal Specification of Group Membership Services," Department of Computer Science, Cornell University, Ithaca, New-York (USA), Tech. Rep. TR 95-1534, Aug. 1995.

[7] T. Anker, D. Dolev, and I. Keidar, "Fault tolerant video on demand services," in *Proceedings of 19th IEEE International Conference on Distributed Computing Systems*, Austin, TX , USA, 1999, pp. 244–252.

[8] L. Arantes, P. Sens, G. Thomas, D. Conan, and L. Lim, "Partition Participant Detector with Dynamic Paths in MANETs," in *Proceedings of 9th IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, USA, Jul. 2010.

[9] O. Babaoğlu, R. Davoli, and A. Montresor, "Group Communication in Partitionable Systems: Specification and Algorithms," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 308–336, Apr. 2001.

[10] O. Babaoğlu, R. Davoli, A. Montresor, and R. Segala, "System support for partition-aware network applications," *ACM SIGOPS Operating Systems Review*, vol. 32, pp. 41–56, January 1998.

[11] K. Birman, R. Friedman, M. Hayden, and I. Rhee, "Middleware support for distributed multimedia and collaborative computing," *Software: Practice and Experience*, vol. 29, no. 14, pp. 1285–1312, 1999.

[12] R. Boichat, P. Dutta, S. Frølund, and R. Guerraoui, "Deconstructing paxos," *ACM SIGACT News Distributed Computing Column*, vol. 34, no. 1, pp. 47–67, Mar. 2003.

[13] D. Cavin, Y. Sasson, and A. Schiper, “Consensus with Unknown Participants or Fundamental Self-Organization,” in *Proceedings of the 3rd International Conference on AD-HOC Networks & Wireless*, ser. Lecture Notes in Computer Science, no. 3158. Vancouver, British Columbia, Canada: Springer-Verlag, Jul. 2004, pp. 135–148.

[14] T. Chandra and S. Toueg, “Unreliable Failure Detectors for Reliable Distributed Systems,” *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996.

[15] G. Chockler, I. Keidar, and R. Vitenberg, “Group Communication Specifications: A Comprehensive Study,” *ACM Computing Surveys*, vol. 33, no. 4, pp. 427–469, Dec. 2001.

[16] J. C. García, S. Beyer, and P. Galdámez, “A stability criteria membership protocol for ad hoc networks,” in *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems*. Vilamoura, Portugal: Springer-Verlag, 2009, pp. 690–707.

[17] F. Greve, P. Sens, L. Arantes, and V. Simon, “A failure detector for wireless networks with unknown membership,” in *Proceedings of the 17th international conference on Parallel processing*, Bordeaux, France, 2011, pp. 27–38.

[18] F. Greve and S. Tixeuil, “Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks,” in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007, pp. 82–91.

[19] J.-F. Hermant and G. Le Lann, “Fast Asynchronous Uniform Consensus in Real-Time Distributed Systems,” *IEEE Transactions on Computers*, vol. 51, pp. 931–944, Aug. 2002.

[20] M. Merritt and G. Taubenfeld, “Computing with Infinitely Many Processes,” in *Proceedings of the 14th International Conference on Distributed Computing*, Toledo, Spain, Oct. 2000, pp. 164–178.

[21] B. Milic, N. Milanovic, and M. Malek, “Prediction of Partitioning in Location-Aware Mobile Ad Hoc Networks,” in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, vol. 9, 2005, pp. 306–313.

[22] A. Mostefaoui, M. Raynal, C. Travers, S. Patterson, D. Agrawal, and A. El Abbadi, “From Static Distributed Systems to Dynamic Systems,” in *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, Florianpolis, Brazil, Oct. 2005, pp. 109–118.

[23] P. Murray, “A Distributed State Monitoring Service for Adaptive Application Management,” in *Proceedings of IEEE International Conference on Dependable Systems and Networks*, Yokohama, Japan, Jul. 2005, pp. 200–205.

[24] S. Pleish, O. Rtti, and A. Schiper, “On the Specification of Partitionable Group Membership,” in *Proceedings of the 7th European Dependable Computing Conference*, Kaunas, Lithuania, May 2008, pp. 37–45.

[25] S. Sastry and S. Pike, “Eventually Perfect Failure Detectors Using ADD Channels,” in *Proceedings of the 5th International Symposium on Parallel and Distributed Processing and Applications*, Nigara Falls, Canada, 2007, pp. 483–496.

[26] A. Schiper, “Brief announcement: dynamic group communication,” in *Proceedings of the 22nd ACM Symposium on Principles of Distributing Computing*, Boston, Massachusetts, USA, 2003, pp. 113–113.

[27] V. Srivastava, A. Hilal, M. S. Thompson, J. N. Chattha, A. B. Mackenzie, and L. A. Dasilva, “Characterizing Mobile Ad Hoc Networks — The MANIAC Challenge Experiment,” in *Proceedings of of 3rd ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization (WiNTECH)*, Sep. 2008.

[28] S. Tucci-Piergiovanni and R. Baldoni, “Connectivity in Eventually Quiescent Dynamic Distributed Systems,” in *Proceedings of the 3rd Latin-American Symposium on Dependable Computing*, vol. 4746, Morelia, Mexico, 2007, pp. 38–56.

[29] ——, “Eventual Leader Election in Infinite Arrival Message-Passing System Model with Bounded Concurrency,” in *Proceedings of the 8th European Dependable Computing Conference*, Valencia, Spain, 2010, pp. 127–134.

[30] K. Wang and L. Baochun, “Group mobility and partition prediction in wireless ad-hoc networks,” in *Proceedings of IEEE International Conference on Communications*, 2002, pp. 1017–1021.