Consensus-Free Atomic Broadcast Protocol for Mobile Distributed Systems

Nadjette Rebouh, and Louiza Bouallouche - Medjkoune

***Abstract*—The atomic broadcast problem, which consists in delivering messages atomically to multiple processes, is critical for maintaining consistency in distributed systems. This paper introduces a new atomic broadcast protocol for mobile distributed systems. Existing consensus-based protocols have drawbacks such as performance dependencies and specific consensus requirements. The proposed protocol addresses these limitations by allowing processes to cooperate through consecutive rounds to agree on the message delivery sequence without additional building blocks. It tolerates crash failures and uses an unreliable failure detector for fault-tolerance. The protocol simplicity is enhanced by the use of the $\diamondsuit S$ failure detector to select a decentralized round leader. Performance evaluation carried out by simulation considers message overhead, latency, energy consumption, and additionally examines the impact of the consensus block on the atomic broadcast protocol.**

***Index Terms*—Mobile Distributed Systems, Unreliable Failure Detectors, Atomic Broadcast, Consensus, Rotating Coordinator, Blockchain.**

I. Introduction

THE atomic broadcast problem is a critical challenge in mobile distributed systems, as it requires all messages to be delivered atomically to multiple processes. Therefore, atomic broadcast provides a powerful mechanism for the implementation of fault-tolerant applications, especially when it is necessary to ensure that all processes have the same view of the system state. Overall, the atomic broadcast is an important concept in the design and implementation of distributed systems and is essential for achieving reliable, consistent behavior in complex, multi-processes environments. Achieving atomic delivery of messages is complicated by various factors, such as failures, message loss, and delays, which can cause inconsistencies and incorrect behaviors in the system.

A. Mobile Distributed Systems

A mobile distributed system is a complex networked framework where multiple wireless processes (or nodes) collaboratively operate to achieve tasks, share resources, and provide services [1]. These systems exhibit the fundamental characteristic of decentralization, allowing processes to communicate and coordinate without reliance on a central entity. The wireless nature introduces challenges stemming from varying signal strengths, potential interference, and mobility of processes, all of which need to be addressed for effective communication, seamless data sharing, and coherent execution of operations across the system.

Mobile distributed systems showcase distinctive characteristics. Processes mobility allows processes to shift within the system, posing connectivity and communication challenges. Limited resources, comprising processing power and energy, drive the need for resource-efficient strategies. These systems adopt decentralized approaches, fostering autonomous process collaborations while necessitating synchronization solutions. The dynamic system topology, influenced by mobility and wireless conditions, demands adaptive mechanisms for routing, data dissemination, and fault-tolerance [2]. In conclusion, these characteristics shape mobile distributed systems into complex entities, necessitating innovative solutions for effective communication, prudent resource utilization, and cohesive coordination in a dynamic system environment.

Mobile distributed systems encounter multifaceted challenges. Energy optimization and load balancing are pivotal, while system fluctuations and mobility demand steady connectivity. Data synchronization poses complexities in maintaining consistency, while security concerns amplify in open settings. Scalability and fault-tolerance are vital for growing systems, and interoperability issues arise from processes diversity [3]. Overcoming these challenges necessitates innovative solutions that harmonize energy efficiency, reliable communication, secure data handling, and system scalability within mobile distributed systems.

One of the biggest challenges in mobile distributed systems is preserving data consistency in the face of process mobility, failures, and constraints emerging as a major hurdle. A typical remedy is an atomic broadcast protocol, especially valuable for realizing fault-tolerant services through software-based replication [4]. This foundational technique ensures that all accurate service replicas receive and apply identical updates in a uniform sequence, thus upholding service coherence effectively.

B. Atomic Broadcast

Atomic broadcast is, also, known as a typical example of agreement problems encountered in the design and imple-

Manuscript received November 6, 2025; revised January 12, 2026. Date of publication April 14, 2026. Date of current version April 14, 2026. The associate editor prof. Andrej Hrovat has been coordinating the review of this manuscript and approved it for publication.

N. Rebouh was with Research Unit LaMOS (System Modeling and Optimization), Faculty of Exact Sciences, University of A.Mira, Bejaia, Algeria (e-mail: nadjette.rebouh@univ-bejaia.dz).

L. Bouallouche - Medjkoune was with Research Unit LaMOS (System Modeling and Optimization), Faculty of Exact Sciences, University of A.Mira, Bejaia, Algeria (e-mail: louiza.medjkoune@univ-bejaia.dz).

Digital Object Identifier (DOI): 10.24138/jcomss-2025-0225

mentation of fault-tolerant distributed systems. An agreement problem, involves a set of processes, is characterized by the fact that these processes have to agree on a common value. For example, in the atomic broadcast problem, processes have to agree on a single delivery order for a set of messages [5], [6].

However, solving the agreement problems, respectively the atomic broadcast problem in asynchronous systems is impossible. This is due to the difficulty of distinguishing a crashed process from an extremely slow process. The problem is known as the FLP [1] impossibility [7], [8]. In their seminal paper [9], the authors proposed a solution to this problem by augmenting the system with modules that give information about the processes states (*i.e.*, if they are correct or crashed). Hence, the notion of unreliable failure detectors is introduced.

In addition to the definition of unreliable failure detectors, their classification, and their classes, the authors also proposed solutions to different agreement problems such as consensus, reliable broadcast, and atomic broadcast. Moreover, they showed that agreement problems could be solved by reduction. In other words, at first, a solution to an agreement problem is provided and then this solution is reduced to solve another agreement problem. Similarly, atomic broadcast and consensus could be solved by reduction.

Therefore, a solution to the consensus problem could be reduced to a solution to the atomic broadcast problem and vice versa. Furthermore, an agreement problem solution can be used as a building block of another agreement problem solution.

C. Motivation

Several solutions to the atomic broadcast problem have been proposed according to the reduction property (as a transformation solution or as a building block). Since consensus is the most basic agreement problem [10], [11] many consensus-based atomic broadcast protocols have been presented.

The first consensus-based atomic broadcast protocol has been introduced in [9]. The protocol shows that the atomic broadcast can be, easily, reduced to consecutive executions of the consensus. Each consensus execution tries to define and reaches a sequence of broadcast messages to be atomically delivered by all the processes. Therefore, the idea of consensus-based atomic broadcast has been investigated more in other works and generates more efficient protocols in different ways. The contribution of these solutions is twofold. All the communication patterns are encapsulated in a box named the consensus. Then, processes in the atomic broadcast protocol do not have to cope directly with tasks other than the messages transmission. Furthermore, the failure detection mechanism is also managed implicitly by the consensus box.

However, consensus-based atomic broadcast protocols have many drawbacks. Since the consensus is the building block of the messages sequence definition, then the performance of the atomic broadcast protocol is directly related to the performance of the underlying consensus protocol [12] [13]. Moreover, the performance includes also the performance of the underlying failure detector. Moreover, some consensus-based atomic broadcast protocols require specific consensus protocols which are neither practical nor efficient in some environments.

This paper presents a new atomic broadcast protocol in mobile asynchronous systems. The approach used is completely different from the already presented approach. Processes have to cooperate, in consecutive asynchronous rounds, to agree on the sequence of messages to be atomically delivered without requiring any further building blocks. The protocol tolerates $f < n/2$ crash failures (where f denotes the maximum number of processes that may crash and n denotes the total number of processes) and requires the unreliable failure detector $\diamondsuit S$ [9]. Therefore, it is considered efficient in addition to its simplicity. Moreover, the protocol benefits from the ultimate output of the $\diamondsuit S$ failure detector to choose the leader of the round (*i.e.*, the coordinator of the round). Hence, the processes have only to agree on the coordinator proposition in a decentralized scheme. As a result, the protocol performance is significantly impacted.

The contributions of this paper can be summarized as follows:

- a new atomic broadcast protocol designed specifically for mobile asynchronous systems.
- a cooperative mechanism allowing processes to agree on message ordering without relying on additional building blocks.
- fault-tolerance supporting up to $f < n/2$ crash failures.
- utilization of the unreliable failure detector $\diamondsuit S$ to ensure both efficiency and simplicity.
- a decentralized leader selection mechanism based on the ultimate output of $\diamondsuit S$.
- improved performance compared to existing approaches due to reduced coordination overhead.

These contributions collectively demonstrate the novelty, efficiency, and practical relevance of the proposed protocol, while the limitations highlighted above justify the need for an alternative that avoids consensus dependency; the next section reviews existing approaches in detail to position our contribution within the current state of the art.

The rest of the paper is organized as follows. It is composed of seven main sections. Section II presents the works in relation to our topic. Section III defines the system model, the class $\diamondsuit S$ of failure detectors, the consensus, the reliable broadcast, and the atomic broadcast problems. Then, Section IV exhibits the atomic broadcast protocol. Section V gives the correctness proof of the proposed protocol. Section VI discusses and analyses the protocol cost, and compares it with other related works. Finally, Section VII concludes the paper.

II. Related Work

Consensus-based atomic broadcast approach has been introduced for the first time in [9] (subsequently denoted as CCBABP)due to the reduction property. The authors showed that it is possible to reduce the atomic broadcast problem to a sequence of consensus executions. In each consensus invocation, processes try to reach an agreement on the sequence

[1]The impossibility result of Fisher, Lynch, and Paterson.

of messages to be atomically delivered. Since then, several solutions have been proposed according to this approach in different system models. In [14] (subsequently denoted as DCBABP), authors exploit the same approach in the same system model, but in a different way. They rely on the consensus module only in specific scenarios; *i.e.*, when the processes are not enabled to agree on the same set of messages but they could share the same prefix of this set. Hence, the prefix will be the entry of the consensus module and the processes have to agree on this entry. Both protocols are compatible with any consensus protocol.

The atomic broadcast protocol of [9] is transformed in [15] to support the indirect consensus module. In the indirect consensus, processes have to agree on the set of messages identifiers. In other words, processes exchange the messages identifiers instead of the complete messages. Hence, the consensus module is totally dissociated from the messages sizes. Other solutions have been proposed to cope with the messages ordering in other system models. [16] and [17] use the reduction approach in the crash recovery system where the processes may recover after crashes. In a probabilistic system, authors of [18], [19], [20] decide, through the consensus module, optimistically or conservatively on the sequence of messages to be delivered according to the satisfaction or not of some order properties. However, in these solutions, the atomic broadcast protocols can not function with any consensus module. Hence, they require specific consensus protocols to handle the systems properties. In [21], authors present solutions to the atomic broadcast problem in fully anonymous distributed systems, where processes have no identifiers.

Although the reduction approach relieves the atomic broadcast protocol of the management of failures and consensus task (*i.e.*, the communication task between the processes in order to reach a decision about the sequence of messages to be delivered), it has many drawbacks. The dependency between the consensus protocol and the atomic broadcast protocol in some works makes the atomic broadcast problem unsolvable in some circumstances. Also, since the atomic broadcast protocol is based on the consensus protocol, the performance of the former is heavily related to the performance of the latter (and the performance of the underlying failure detector). In [12], [13], and [22], the authors show the impact of the performance of the underlying consensus on the atomic broadcast protocol performance. Therefore, the choice of the underlying consensus protocol may increase or decrease the atomic broadcast protocol performance in terms of message complexity and latency. Hence, the protocol throughput is affected. In the current paper, we use a consensus-free approach that relieves the atomic broadcast protocol from the overhead caused by the consensus protocol in the previous works.

Recent research on byzantine atomic broadcast explores both consensus-based and consensus-free protocols, particularly in blockchain systems. Otter [23] uses sharded BFT consensus with Abortable Fork Detection to support scalable atomic broadcast across hundreds of processes. Other approaches, like Fides [24] and TenderTee [25], rely on trusted components or asynchronous DAG structures to reduce consensus overhead while preserving reliability. These protocols are designed for high-throughput, fault-tolerant blockchain architectures. The trend reflects a move toward modular, efficient atomic broadcast in decentralized environments [26], [27].

More recently, several studies have advanced this direction by improving atomic broadcast efficiency under asynchronous or partially synchronous conditions. Sony et al. [28] proposed a committee-based optimal asynchronous byzantine agreement protocol that reduces message complexity while maintaining robustness. Sui et al. [29] introduced a signature-free atomic broadcast achieving optimal $O(n^2)$ message cost and constant expected latency, representing a major step toward scalable reliable broadcast in large networks. Gupta et al. [30] explored novel tail-carrying mechanisms in consensus pipelines to reduce latency in broadcast execution, while Amores-Sesar et al. [31] designed a DAG-based consensus framework leveraging asymmetric trust for efficiency in heterogeneous environments.

Complementary to these works, Ruchel et al. [32] proposed a leaderless hierarchical atomic broadcast protocol that eliminates leader bottlenecks and scales efficiently across large systems. Bravo et al. [33] introduced a vertical atomic broadcast that separates ordering from replication, improving scalability in modular architectures. The recent work of Amores-Sesar et al. [34] investigated latency reduction using DAG-based message ordering without common-coin primitives, offering new perspectives for asynchronous message sequencing.

To better highlight the novelty of our approach with respect to existing work, Table I summarizes key properties of representative atomic-broadcast solutions discussed above. The table focuses on the system model, fault model, whether the approach is consensus-based, mobility support, key assumptions (or failure-detector requirements), and brief notes on limitations or distinguishing traits.

Having examined the limitations of consensus-based methods, we now turn to formalizing the system assumptions and building blocks used throughout the paper.

III. System Model and Underlying Blocks

In this section, we define the system model and present the underlying building blocks used throughout the paper.

A. System Model

We consider a mobile asynchronous distributed system composed of a finite set of processes $\Pi = \{p_1, p_2, \ldots, p_n\}$ (Fig. 1). Processes communicate with each other by exchanging messages through reliable channels. A reliable channel does not create or duplicate messages, and every message sent is eventually received. Processes can fail only by crashing: a process behaves correctly according to its specification until it (possibly) crashes. The system tolerates $f < n/2$ crash failures and requires any unreliable failure detector of the class $\diamondsuit S$.

TABLE I
COMPARISON OF THE PROPOSED PROTOCOL WITH REPRESENTATIVE EXISTING ATOMIC-BROADCAST APPROACHES

Work (ref)	*System model*	*Fault model*	*Consensus-based?*	*Mobility support*	*Main assumptions / Notes*
Consensus-based atomic broadcast (CCBABP) [9]	Async	Crash	Yes (reduction to repeated consensus)	No (static model in original)	Reduces atomic broadcast to a sequence of consensus instances; performance depends on the underlying consensus and failure detector.
DCBABP [14]	Async	Crash	Partially (consensus used as a module in specific scenarios)	No (static)	Uses consensus only when processes cannot agree locally; relies on consensus on a prefix to ensure progress. Compatible with any consensus protocol.
Indirect-consensus transformation [15]	Async	Crash	Yes (indirect consensus on identifiers)	No (static)	Exchanges message identifiers instead of full messages so consensus is independent of message sizes; decouples consensus from message payloads.
Crash-recovery variants [16], [17]	Crash-recovery model (processes may recover)	Crash-recovery	Yes (reduction-based)	No (static)	Adapt reduction approach to recovery semantics; require consensus protocols designed for crashes + recovery.
Probabilistic approaches [18], [19], [20]	Probabilistic / randomized models	Crash (probabilistic guarantees)	Yes (optimistic/conservative consensus choices)	No (typically static)	Use consensus optimistically or conservatively depending on order properties; often require specialized consensus and do not interoperate with arbitrary consensus modules.
Anonymous systems [21]	Fully anonymous networks (no IDs)	Crash	Varies / specialized	No	Solutions for systems without identifiers; require different techniques and are not directly comparable with ID-based protocols.
Blockchain / BFT systems (Otter [23], Fides [24], TenderTee [25], etc.)	Large-scale / blockchain (often partially synchronous or specialized async)	Byzantine	Mostly consensus-based (BFT variants, sharded BFT, trusted components)	Typically No (designed for static / large decentralized networks)	Target high-throughput, Byzantine fault-tolerance; often rely on sharding, trusted components, DAG structures, or specialized primitives to reduce consensus overhead.
Recent BFT / asynchronous improvements (Sony [28], Sui [29], Gupta [30], Amores-Sesar [31], [34], Ruchel [32], Bravo [33])	Async or partially sync, heterogeneous settings	Crash or Byzantine (paper-dependent)	Varies: BFT or optimized agreement protocols	Mostly No (focus on scaling / latency)	Advances include signature-free constructions, optimal $O(n^2)$ message cost claims, DAG-based ordering, leaderless hierarchies, and pipeline/tail-carry mechanisms to reduce latency and improve scalability.
This paper	Async mobile distributed system	Crash	*Consensus-free*	*Yes (mobile)*	Uses $\diamondsuit S$ (weak failure detector), rotating coordinator, tolerates $f < n/2$. Eliminates consensus overhead—improved modularity and potential performance/throughput benefits in mobile settings.

1) Design Assumptions: the design of the proposed protocol relies on the following explicit assumptions:

- the system is *fully asynchronous*, meaning there are no bounds on message transmission delays or process execution speeds.
- communication channels are *reliable*, delivering all messages exactly once without reordering or duplication.
- failures are restricted to *crash failures*, with crashed processes not recovering.
- a majority of correct processes is assumed (*i.e.*, $f < n/2$) to ensure that at least one correct process participates in each round and that a majority of proposals can be collected.
- the protocol relies on the unreliable failure detector $\diamondsuit S$, which eventually identifies at least one correct process that is never suspected by others. This enables the rotating-coordinator scheme and guarantees progress despite asynchrony.

B. Underlying Blocks

In this paper, we present a solution to the atomic broadcast problem that relies on the unreliable failure detector $\diamondsuit S$ and the reliable broadcast primitives. In the following, we define the underlying concepts.

1) Unreliable Failure Detectors: In [9], Chandra and Toueg introduced the notion of unreliable failure detectors. Depending on two abstract properties, namely *Completeness* property and *Accuracy* property, they define eight classes of failure detectors. In this paper, we consider the class of

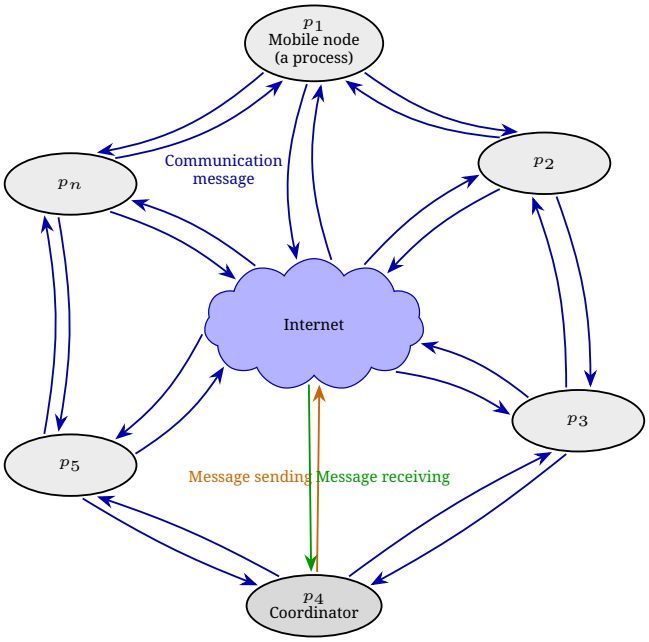

Fig. 1. The system model.

failure detectors denoted $\diamondsuit S$. This class has been proven to be sufficient and, also, the weakest to solve the atomic broadcast problem in an asynchronous system prone to even one failure [35]. The class includes all the failure detectors satisfying the two following properties:

- *strong Completeness:* eventually, every process that crashes is permanently suspected by every correct process.
- *eventual Weak Accuracy:* there is a time after which some correct process is never suspected by correct processes.

C. The Consensus

In the consensus problem, a group of processes has to agree on a common value. This value was previously proposed by one of the processes.
The consensus problem is defined by the following properties [9], [7]:

- *validity*: if a process decides a value *v* then *v* was proposed by at least one process.
- *integrity*: each process decides at most once.
- *agreement*: two correct processes should not decide differently.
- *termination*: each correct process eventually decides a value.

1) The Reliable Broadcast: guarantees reliable delivery of messages to processes (a message sent is received by all correct processes or by any of them) [36]. It respects the policy of "all or nothing". The implementation of the reliable broadcast can be achieved through two primitives, namely, *R_broadcast* and *R_deliver*. The first primitive allows a process to send a message to the whole set of processes. The second primitive allows a process to deliver a message sent by the invocation of *R_broadcast*.
The reliable broadcast is defined by the following properties:

- *validity*: if a correct process R_broadcasts a message *m*, then it eventually R_delivers *m*.
- *integrity*: for any message m, every process R_delivers m at most once.
- *agreement*: if a correct process R_delivers a message m, then all correct processes eventually R_deliver *m*.

2) The Atomic Broadcast: is an extension of the reliable broadcast problem. It is defined by two primitives, namely, $A\_broadcast$ and $A\_deliver$. The first primitive allows a process to send a message to the whole set of processes. The second primitive allows a process to deliver a message sent by the invocation of $A\_broadcast$.

The atomic broadcast problem is defined, in addition to the reliable broadcast properties, by the *total order* property that ensures that processes deliver the same sequence of messages [36]:

- *total order*: if two correct processes p_i and p_j $A\_deliver$ two messages m and m', then p_i $A\_delivers$ *m* before *m'* if and only if p_j $A\_delivers$ m before m'.

With these definitions established, we are now ready to present the proposed consensus-free atomic broadcast protocol (CFABP) and describe its operations.

IV. The Atomic Broadcast Protocol

This section is devoted to CFABP and its description.

A. The Protocol Overview

Algorithm 1 solves the atomic broadcast in asynchronous mobile systems using any failure detector of the class $\diamondsuit S$. In other words, it works with any failure detector that satisfies strong completeness and eventual weak accuracy. Moreover, it tolerates a majority of correct processes ($n \geq 2f + 1$).

The accuracy property of $\diamondsuit S$ failure detector states that there exists a correct process and a time after which this process is never suspected by any other correct process. To benefit from this property, the rotating coordinator principle is employed.

Algorithm 1 uses the rotating coordinator paradigm [9] and proceeds in asynchronous consecutive rounds (processes do not necessarily execute the same round at a given time). In each round and for all processes, the coordinator is chosen according to the current round number. Hence, all processes have a priori knowledge of the current coordinator identity. By "current coordinator", we mean that the role of the coordinator rotates through all processes during the algorithm execution rounds.

Algorithm 1 behaves in a decentralized scheme. In other words, the current coordinator is only used as a launcher of the round. Therefore, it tries to impose its message sequence proposition on processes as A *_delivered*. Then, the processes have to cooperate to reach a decision on the message sequence. Therefore, they share their received coordinator

proposal, if any. They do so by exchanging the coordinator proposal sequence of messages. Moreover, they exchange the appropriate set of messages in case they do not suspect the current coordinator to be crashed (by requesting their $\diamondsuit S$ failure detectors); otherwise, this set is empty. Hence, once a correct process receives a majority of messages (more than $f/2$) handling the coordinator proposal (*i.e.*, the set is not empty), it decides accordingly to $A\_deliver$ the underlying message sequence. In addition, it shares its decision with all other correct processes.

B. The Protocol Workflow

To clarify Algorithm 1 execution, we provide a step-by-step description of how it operates. The protocol progresses through asynchronous rounds during which each process performs four main tasks:

1) *collection of newly delivered messages*: Each process maintains the set $R\_delivered$, containing messages reliably broadcast but not yet atomically delivered. At the beginning of each round, the process gathers the messages that still need ordering.
2) *coordinator proposition*: the process designated as the coordinator of the current round (based on the rotating rule) proposes a message set composed of all messages in $(R\_delivered - A\_delivered)$. This set represents the process's view of undelivered messages that must be ordered.
3) *exchange and validation of proposals:* all processes exchange their coordinator's proposal. According to the $\diamondsuit S$ failure detector, if a process does not suspect the coordinator, it adopts the received proposal; otherwise, it keeps an empty set. Each process waits until it receives a majority of proposals for the same round.
4) *decision and delivery:* when a correct process receives more than $(n - f)$ identical proposals, it decides on that set. It then broadcasts a $DEL(seq, S)$ message and atomically delivers the proposed messages according to their sequence numbers. This guarantees agreement among all correct processes.

C. The Protocol Variables

For readability, we briefly define the main variables used by each process:

- $R\_delivered_i$: a set of messages received via reliable broadcast but not yet atomically delivered.
- $A\_delivered_i$: a set of messages already atomically delivered.
- seq_num: a local sequence number used to impose total order among message sets.
- $coord\_prop$: the current coordinator's proposed set of messages.
- msg_prop: the set of pending messages locally available for ordering.
- r_i: a local round counter for process p_i.
- $MSG(r, S)$: a message used to disseminate a proposal S for round r.
- $DEL(seq, S)$: message used to notify the final decision on sequence S with sequence number seq.

D. The Protocol Description

The complete atomic broadcast protocol is described in Algorithm 1. To solve the atomic broadcast problem, a process goes through different independent tasks. Therefore, processes may execute the same task at the same time without any conflict or overlap. Moreover, the protocol begins by invoking the atomic broadcast primitive $A\_broadcast(m)$ at *line* 6 and ends by executing the atomic broadcast primitive $A\_deliver(m)$ at *line* 9. Upon invoking $A\_broadcast(m)$, a process p_i executes the reliable broadcast primitive $R\_broadcast(m)$, and *m* is received by all correct processes due to the agreement property of the reliable broadcast. Hence, the message *m* is added to the set of reliably delivered messages $R\_delivered_i$ at *line* 11. Thus, the set $R\_delivered$ is destined to be the coordinator proposal for an unknown future round.

The main task of the protocol is expressed by *lines* 18 to 34. As mentioned previously, Algorithm 1 proceeds in consecutive asynchronous rounds. Each process manages different local variables (initialized in *lines* 1 to 5) in addition to a special variable that represents the current coordinator proposal $coord\_prop$. For a given round *r*, the process p_i chooses the coordinator round at *line* 19 (*i.e.*, it becomes the p_c). The processes, then, exchange their $coord\_prop$ set associated with the current round number. Therefore, any other old rounds messages are automatically excluded.

Moreover, only processes that do not suspect the current coordinator update their $coord\_prop$ sets, the others keep them empty. This is due to the fact that a process that doesn't have the coordinator in its failure detector list has already received the coordinator proposal messages set (*i.e.*, one technique used to implement unreliable failure detectors in asynchronous systems). Hence, it updates its own set $coord\_prop$ accordingly. In this task, a given process p_i has three blocking instructions.

The process p_i can be blocked at *line* 20 until it has some $R\_delivered$ messages but not yet $A\_delivered$ ones. Also, p_i has to wait at *line* 24 until it receives the current coordinator proposal set or receives a suspicion message of the current coordinator from its associated $\diamondsuit S$ failure detector. According to the message p_i received, it may update its $coord\_prop$ set. Finally, p_i has to wait for the reception of a majority of $MSG(r, coord\_prop)$ messages at *line* 28. If the set $coord\_prop$ is not empty (*i.e.*, it is the correct current coordinator proposal set), then p_i decides to $A\_deliver$ all the messages in the set $coord\_prop$ (*lines* 15 to 17) according to sequence numbers associated with them after broadcasting its decision at *line* 31 to ensure the protocol safety. In this situation, there is a majority of processes that do not suspect the current coordinator to be failed. Otherwise, p_i proceeds to a new round.

Algorithm 1 *Every process p_i executes the following:*

```
1: Initialization
2:     R_delivered_i ← ∅;
3:     A_delivered_i ← ∅;
4:     seq_num_i ← 0;
5:     r_i ← 0;

6: upon A-broadcast(m):
7:     R − broadcast(m);

8: upon A-deliver(m):
9:     return(m);

10: upon R-deliver(m):
11:    R_delivered_i ← R_delivered_i ∪ {m};

12: upon receiving DEL(seq_num, msg_prop) from p_j:
13:    if (seq_num = seq_num_i) then
14:        ∀k ≠ j: send DEL(seq_num, msg_prop) to p_k ;
15:        A_delivered_i ← A_delivered_i ∪ msg_prop;
16:        seq_num_i ← seq_num_i + 1;
17:        ∀m ∈ A_delivered_i: A − deliver(m);
18: While true do
19:    c ← (r_i mod n) + 1; coord_prop_i ← ∅;
20:    wait until ((R_delivered_i − A_delivered_i) ≠ ∅);
21:    msg_prop_i ← (R_delivered_i − A_delivered_i);
22:    if (i = c) then coord_prop_i ← msg_prop_i;
23:    else if (i ≠ c) then
24:        wait until ((MSG(r_i, msg_prop) is received from p_c) ∨
       (p_c ∈ D_{p_i}) );
25:        if (MSG(r_i, msg_prop) has been received) then
26:            coord_prop_i ← msg_prop;
27:    ∀j: send MSG(r_i, msg_prop_i) to p_j ;
28:    wait until(MSG(r_i, coord_prop)
       has been received from (n-f) processes);
29:    Let rec_msg_i = {(coord_prop)|MSG(r_i, msg_prop) is re-
       ceived from p_j};
30:    if (rec_msg_i = {coord_prop}) then
31:    ∀j do send DEL(seq_num_i, msg_prop_i) to p_j ;
32:    else
33:        r_i ← r_i + 1;
34: endWhile
```

Finally, *lines* 12 to 17 are executed when a participating process p_i receives an $A\_delivery$ message from another process p_j. It means that p_j has already decided on the coordinator sequence of messages at *line* 14 or at *line* 31. Hence, p_i broadcasts, in its turn, the $A\_delivery$ message and $A\_delivers$ the messages sequence if possible (*i.e.*, depending on the appropriate sequence number). This task ensures the agreement property of the protocol. In other words, if one correct process $A\_delivers$ a message, then all correct processes eventually $A\_deliver$ it.

Algorithm 1 must be proven correct. The following section provides the formal proof that it satisfies all atomic broadcast properties.

V. Correctness Proof

In this section, we prove that our protocol satisfies the *validity*, *agreement*, *integrity*, and *total order* properties of the atomic broadcast algorithm. The proof assumes a majority of correct processes (*i.e.*, $f < n/2$), also, the system is augmented with $\diamondsuit S$ failure detector.

1) validity:

Lemma V.1. If a correct process $A\_broadcasts$ a message *m*, then it eventually $A\_delivers$ *m*.

Proof. When a process p_i $A\_broadcasts$ a message *m* at *line* 7, then *m* is automatically added to the set $R\_delivered$ of all processes at *line* 11, according to the agreement property of the reliable broadcast. Hence, the coordinator of the round has *m* in its $R\_delivered$ set. Thus, *m* obviously belongs to the messages proposition of the current coordinator at *line* 21, since it has not been $A\_delivered$ yet.
Then, after some time *t*, a correct process p_c will be trusted by all the correct processes, according to the *eventual weak accuracy* of the $\diamondsuit S$ failure detector. Hence, a process p_j will eventually receive a majority of messages trusting the current coordinator at *line* 28 and decides, accordingly, to $A\_deliver$ the coordinator proposition set at *line* 17. Therefore, *m* is $A\_delivered$. However, before $A\_delivering$ *m*, p_j broadcasts its decision at *line* 31. The process p_i receives p_j decision at *line* 12 and $A\_delivers$ *m*. □

2) Agreement:

Lemma V.2. If a correct process $A\_delivers$ a message m, then all correct processes eventually $A\_deliver$ *m*.

Proof. A process p_i $A\_delivers$ a message *m* at *line* 17. In this case and to ensure the protocol safety, p_i broadcasts the $DEL(-,-)$ message (and also for itself), which contains the messages sequence including *m* and the appropriate sequence number. By the *agreement property* of the reliable broadcast, all processes eventually receive the $DEL(-,-)$ message, respectively, *m*. Thus, they eventually $A\_deliver$ *m* at *line* 17. Moreover, by *Lemma* V.1, each process $\_delivers$ its own broadcast messages. □

3) integrity:

Lemma V.3. For any message m, every process $A\_delivers$ m at most once.

Proof. At the beginning of each new round, the current coordinator p_c must ensure that the set $(R\_delivered_c - A\_delivered_c)$ is not empty. It means that every message *m* in this set msg_prop_c (*line* 21) has been $R\_delivered$ but not yet $A\_delivered$.
When deciding on a messages sequence, a process p_i assigns a sequence number seq_num to this messages sequence. The sequence number is (1) initialized to zero at *line* 4, (2) automatically incremented after each sequence assignment (at *line* 16) and has to be unique. Thus, the uniqueness of the sequence number ensures the message integrity. Then, the process p_i broadcasts the $DEL(-,-)$ message with seq_num as a parameter. Every process p_j that receives p_i message can only $A\_deliver$ *m* if it has the same sequence number seq_num as p_i. □

4) Total Order:

Lemma V.4. If two correct processes p_i and p_j $A\_deliver$ two messages m and m', then p_i $A\_delivers$ *m* before *m'* if and only if p_j $A\_delivers$ m before m'.

Proof. According to Lemma V.3, each message is $A\_delivered$ in a predefined order following a sequence number assigned by a certain process. Let's consider two

messages *m* and *m'* with their assigned sequence numbers seq_num1 and seq_num2, respectively.
If seq_num1 is equal to seq_num2, then *m* and *m'* belong to the same messages sequence. Then, p_i and p_j $A\_deliver$ *m* and *m'* according to their positions in the messages sequence $A\_delivered$ (*line* 17).
If *m* and *m'* belong to different messages sequence, then either ($seq\_num1 < seq\_num2$) or ($seq\_num1 > seq\_num2$). When $seq\_num1 < seq\_num2$, p_i and p_j $A\_deliver$ *m* before *m'*, otherwise, they $A\_deliver$ *m'* before *m*. □

VI. Simulation and Performance Evaluation

In this section, we assess the performance of the proposed CFABP and draw comparisons with two state-of-the-art consensus-based protocols, namely CCBABP and DCBABP ([9] and [14], respectively). These protocols ensure the total order delivery of broadcast messages within a mobile distributed system. All three protocols depend on unreliable failure detectors to ensure fault-tolerance, and they share the common requirement of a majority of correct processes. However, while CCBABP and DCBABP are consensus-based, CFABP diverges as a consensus-free approach.

A. Simulation

In order to evaluate the performance of our CFABP protocol, we employed the C++ programming language. Table II specifies the parameters used in the simulation, including mobility model (random waypoint), topology (mesh), number of processes (5–50), and the number of broadcast messages.
Figures Fig. 2, Fig. 3, and Fig. 4 depict the executions of the protocols (for a system of 5 processes).

TABLE II
Simulation Parameters

Parameter	value
Number of processes	Variable
Number of broadcast messages	Variable
Mobility model	Random
Topology	Mesh

In order to illustrate the impact of the consensus protocol on the efficiency of the performance of the atomic broadcast protocol, we incorporate two distinct consensus protocols: the Chandra-Toueg consensus algorithm (CCP), based on a centralized communication pattern [9] and the Mostefaoui-Raynal consensus algorithm (DCP), based on a decentralized communication pattern [37], within our comparative analysis. The operational executions of these two protocols are depicted through figures Fig. 5 and Fig. 6 respectively (for a system of 5 processes).

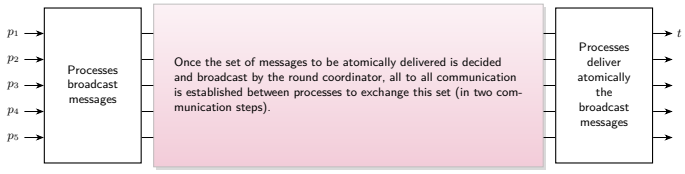

Fig. 2. The CFABP execution.

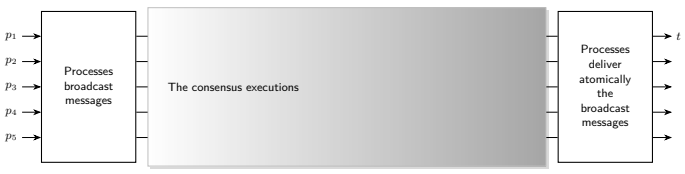

Fig. 3. The CCBABP execution.

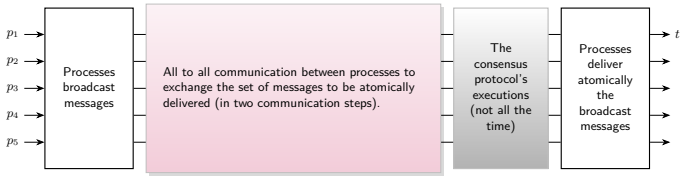

Fig. 4. The DCBABP execution.

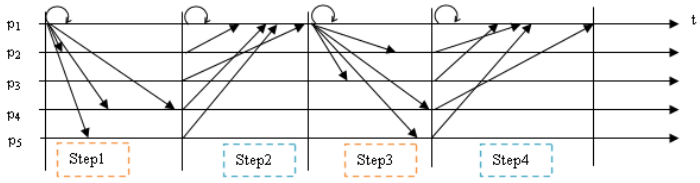

Fig. 5. The CCP execution.

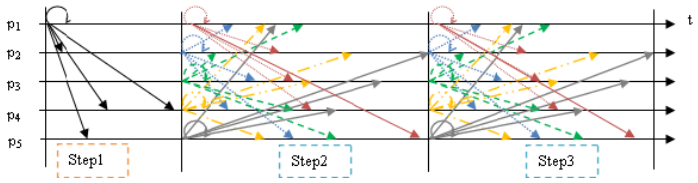

Fig. 6. The DCP execution.

In the evaluation of the protocols, the main parameter is the system size *n* (indicating the number of processes in the system). Additionally, we explore two distinct scenarios: the first corresponds to a free failure context (no failure or wrong suspicion (*f=0*)), while the second scenario accounts for a single failure context (*f=1*). We assume that the failure pertains to the coordinator of the first round.

B. Performance Evaluation

This section presents a comparative performance evaluation of CFABP against two well-established atomic broadcast protocols from the literature: CCBABP, based on the approach of Mostéfaoui *et al.* [14], and DCBABP, derived from Chandra *et al.* [9]. The comparison aims to assess the effect of removing the consensus dependency on the overall protocol efficiency.

The evaluation focuses on two key performance metrics: time complexity and message complexity. Each protocol is tested under two distinct scenarios: a failure-free environment ($f = 0$) and a single-process failure scenario ($f = 1$). For

fairness, both CCBABP and DCBABP are analyzed under two consensus variants—CCP and DCP—to ensure a consistent comparison baseline. The results are obtained through simulation and are summarized in the following subsections.

1) Time Complexity: the time complexity is denoted by the number of communication steps mandated by each protocol to reach the agreement (Table III). Under conditions

TABLE III
COMMUNICATION STEPS

Protocol	CFABP	CCBABP		DCBABP	
		CCP	DCP	CCP	DCP
Communication steps (f=0)	3	4	3	3	3
Communication steps (f=1)	3*2	4*2	3*2	7*2	7*2

of no failures, our CFABP aligns its outcomes with those of DCBABP, considering its two consensus protocols, and with CCBABP using the DCP protocol. These protocols require three communication steps to establish agreement on the message sequence. This arises from their reliance on all-to-all communication, obviating the need for a dedicated process (the coordinator) to decide on the order of the messages.

However, in the case of CCBABP using CCP protocol, the involvement of the coordinator as the round manager, endowed with the authority for the initial proposition and ultimate decision, increases the requisite communication steps. Conversely, in the alternate scenario (allowing for a single failure), our CFABP yields optimal outcomes, particularly in conjunction with CCBABP relying on DCP protocol. The outcomes related to DCBABP are rationalized by the additional demand of the consensus protocol, in addition to the existing all-to-all communication paradigm.

The results in Table III show that CFABP achieves a minimal and stable number of communication steps across both normal and faulty executions. This reflects the efficiency of eliminating repeated consensus invocations — a known bottleneck in [14]- and [9]-based solutions. In CFABP, coordination occurs only once per round, which makes latency predictable and decoupled from the number of failures. In contrast, the traditional protocols exhibit step inflation as soon as failures or coordinator changes occur. This highlights that CFABP scales better in dynamic environments while preserving atomicity.

Another significant point to consider is that in cases where the system experiences frequent process failures, our CFABP demonstrates superior performance when compared to both CCBABP and DCBABP, using both consensus protocols. This advantage comes from the design of our CFABP, which employs the rotating coordinator solely once at the outset to distribute the message set. In contrast, the other protocols employ the rotating coordinator twice initially for the same purpose and subsequently at the end to make the final determination regarding the message sequence. Consequently, the occurrence of frequent failures in the rotating coordinator substantially impacts the execution time of these protocols.

2) Message Complexity: in this section, we are interested in the number of messages exchanged to attain agreement regarding the sequence of messages, all within the context of the two predefined scenarios. It is worth noting that this complexity is quantified with respect to the total number of processes within the system (Table IV). The results presented

TABLE IV
NUMBER OF MESSAGES

Protocol	CFABP	CCBABP		DCBABP	
		CCP	DCP	CCP	DCP
Message complexity	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Number of messages (f=0)	n^2+ $2*n$	4*n	$2*n^2$ $+n$	$2*n^2$ $+n$	$2*n^2$ $+n$
Number of messages (f=1)	$(n^2$ $+2*n)$ $*2$	$(4*n)$ $*2$	$(n^2$ $+2*n)$ $*2$	$(2*n^2$ $+5n)$ $*2$	$(3*n^2$ $+3n)$ $*2$

in Table IV were proven by simulation as follows:

The illustrations presented in Fig. 7 and Fig. 8 investigate a scenario where CCBABP and DCBABP rely on CCP as their fundamental component in both predefined situations: the failure-free context (*f=0*) and the single failure scenario (*f=1*), respectively. These figures illustrate that CCBABP consistently outperforms the other two protocols due to the superior performance of CCP.

In the *f=0* scenario, both CFABP and DCBABP require all-to-all communication among processes to achieve agreement, leading to a quadratic number of exchanged messages. Despite DCBABP not employing the consensus block (only in this case), CCBABP performance is enhanced significantly due to the integration of one-to-all and all-to-one communications within CCP. Notably, CCBABP relies exclusively on consensus invocation, with no inter-process communication prior to consensus, in contrast to DCBABP.

The above results reveal that CFABP maintains a consistent $O(n^2)$ message complexity, which is expected for atomic broadcast protocols that use all-to-all communication. However, the constant factors are significantly lower than in DCBABP because CFABP avoids multiple consensus phases. The contrast with CCBABP (CCP variant) highlights a fundamental design difference: CFABP eliminates unnecessary message coordination rounds, leading to more predictable scalability when n increases. This is particularly relevant for systems like blockchain or replicated state machines, where the number of participants is large and message overhead is critical.

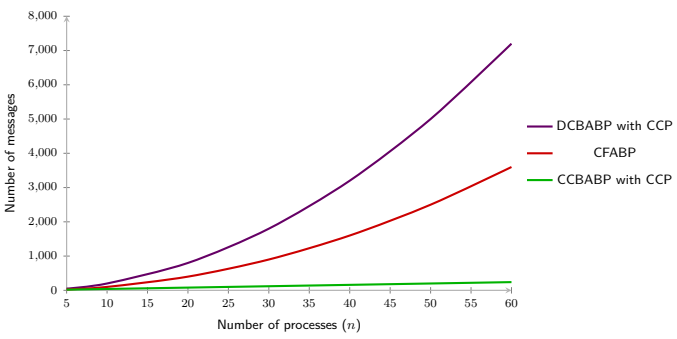

Fig. 7. Message complexity with CCP(f=0).

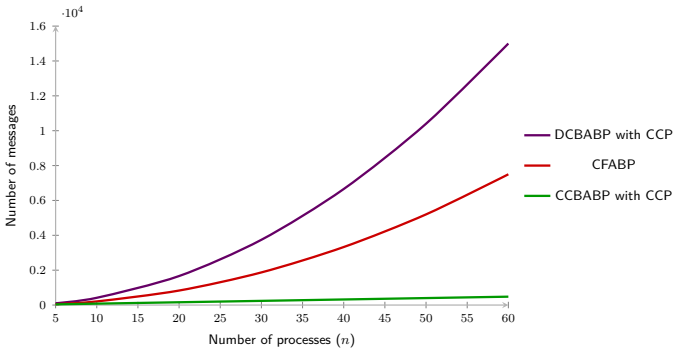

Fig. 8. Message complexity with CCP(f=1).

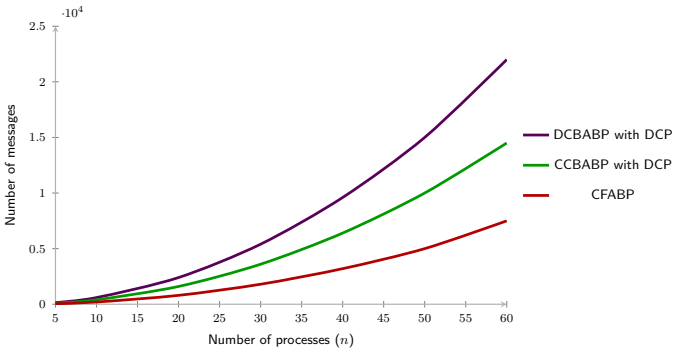

Fig. 10. Message complexity with DCP(f=1).

Figures Fig. 9 and Fig. 10 illustrate the scenario where CCBABP and DCBABP both employ DCP as their foundational component in both scenarios: the failure-free context (*f=0*) and the single failure scenario (*f=1*), respectively. Notably, CFABP consistently outperforms both CCBABP and DCBABP in terms of performance. Additionally, CCBABP performance surpasses that of DCBABP. These outcomes can be attributed primarily to the implementation of the DCP protocol within CCBABP and DCBABP.

Furthermore, DCP employs an all-to-all communication approach across two phases, resulting in a substantial decrease in the overall protocol performance. As a consequence, both CCBABP and DCBABP performances are impacted negatively. Particularly, DCBABP exhibits suboptimal behavior when faced with a single process crash. In such a scenario, the processes require the invocation of the DCP protocol in addition to the protocol standard all-to-all communication. This extra overhead contributes to the observed unsatisfactory results.

The DCP-based results emphasize the resilience advantage of CFABP. By design, CFABP's atomic delivery mechanism tolerates failure without re-invoking costly agreement subroutines. This makes it not only faster under $f > 0$, but also more predictable — a property that simplifies fault management in asynchronous distributed environments. The clear separation between message dissemination and agreement decision in CFABP is what enables this performance stability.

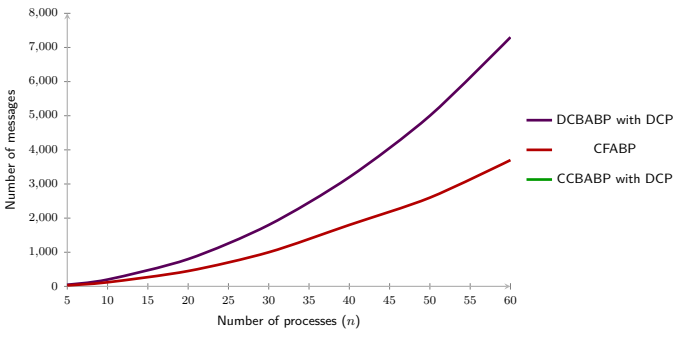

Fig. 9. Message Complexity with DCP (f=0).

C. Optimizations

The following two points present optimizations that can be applied to CFABP presented in Algorithm 1:

- using message identifiers instead of transmitting complete messages leads to a reduction in the message size, subsequently alleviating the system contention.
- the coordinator role can be bypassed, opting for direct all-to-all communication to exchange initial proposals and final decision on messages delivery. This decreases the number of communication steps required for the delivery decision, ultimately diminishing CFABP execution time. Moreover, during the second all-to-all communication phase, transmitting only a prefix of the messages set (a subset of messages containing the minimum shared proposed messages) further optimizes the execution time, resulting in improved protocol performance. This strategy amalgamates efficiency enhancements by minimizing the communication load and reducing the number of communication steps.

D. Results and Discussion

The simulation results show that our CFABP consistently demonstrates favorable outcomes in terms of time complexity across all scenarios, consequently leading to a notable reduction in energy consumption. Additionally, the deployment of the rotating coordinator as a message-set proposer for atomic delivery—while maintaining no influence over the final decision—endows the protocol with decentralization and symmetry, and optimizes the distribution of system workloads, thus contributing to improved load balancing. This design choice effectively prevents the coordinator from becoming a bottleneck, thereby enhancing overall throughput and preserving stable energy levels across processes.

The performance assessment clearly underscores that the selection of the consensus mechanism significantly affects the overall behavior of atomic broadcast protocols. From the obtained results, it is evident that CFABP outperforms both [14] and [9] atomic broadcast protocols in terms of efficiency, particularly under process-crash conditions. By avoiding explicit consensus invocation, CFABP achieves faster message ordering and fewer communication steps. Despite its simplicity, the protocol preserves identical reliability and

atomicity guarantees as the compared models.

Overall, CFABP achieves:

- lower latency in message delivery under both failure-free and failure scenarios.
- reduced communication overhead due to consensus-free coordination.
- comparable fault-tolerance with simpler recovery handling.

Hence, the proposed protocol provides a substantial improvement in performance and scalability for asynchronous mobile environments, validating its effectiveness as a lightweight alternative to classical consensus-based atomic broadcast algorithms.

Furthermore, the obtained results align with the recent research direction emphasizing energy-aware and coordination-minimized designs for distributed systems [30], [32], [34]. These works highlight the need for adaptive fault-tolerance and low-overhead broadcast mechanisms in dynamic environments such as mobile ad hoc networks and edge computing infrastructures. CFABP fits naturally within this trajectory by demonstrating that atomic broadcast can remain both reliable and efficient without depending on heavy consensus rounds or centralized control. This positions CFABP as a forward-compatible model for future distributed frameworks where scalability, resilience, and energy efficiency must coexist.

These observations highlight the practical benefits of the proposed approach. We now conclude by summarizing the contributions and outlining the protocol's applicability.

VII. Conclusion

This paper presents a new atomic broadcast protocol that addresses atomic messages delivery in mobile distributed systems. In contrast to existing consensus-based solutions, the proposed protocol takes an alternative route by eliminating the dependency on consensus mechanisms; *i.e.*, it is consensus-free. It operates through consecutive asynchronous rounds, during which processes collaborate to establish agreement on the message delivery sequence, without necessitating supplementary architectural components. The protocol is designed to tolerate a majority of correct processes and requires the $\diamondsuit S$ failure detector.

Simulation results underscore the efficiency of our protocol, particularly in relation to time complexity and, in certain scenarios, message complexity. Furthermore, the simulations highlight that consensus-free atomic broadcast protocols exhibit superior efficiency in environments characterized by frequent failures and pressing demand for swift message delivery.

Given the good performance and reliability of our atomic broadcast protocol, it proves well-suited for distributed environments like blockchain. Its ability to ensure total order and fault-tolerant message delivery directly supports the requirements of consensus. This positions it as a practical and efficient foundation for building consensus protocols. Therefore, it can serve as a solid core for blockchain systems.

References

[1] Z. Guo, X. Ren, and F. Ren, *Better realization of mobile cloud computing using mobile network computers*. Wireless Personal Communications, 111, 1805-1819, 2020. https://doi.org/10.1007/s11277-019-06958-y.

[2] C. Johnen, L. Arantes, and P. Sens, *FIFO and Atomic broadcast algorithms with bounded message size for dynamic systems*. In 2021 40th International Symposium on Reliable Distributed Systems (SRDS), 277-287, 2021. https://doi.org/10.1109/SRDS53918.2021.00035.

[3] S. Jafarali Jassbi, and E. Moridi, *fault-tolerance and energy efficient clustering algorithm in wireless sensor networks: FTEC*. Wireless Personal Communications, 107, 373-391, 2019. https://dl.acm.org/doi/10.1007/s11277-019-06281-6.

[4] K. Sabaghian, K. Khamforoosh, and A. Ghaderzadeh, *Data Replication and Placement Strategies in Distributed Systems: A State of the Art Survey*. Wireless Personal Communications, 129(4), 2419-2453, 2023. https://doi.org/10.1007/s11277-023-10240-7.

[5] M. Hurfin, R. Macedo, M. Raynal, and F. Tronel, *A general framework to solve agreement problems*. In Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems, 56-65, 1999. https://doi.org/10.1109/RELDIS.1999.805083.

[6] H. C. Hwang, J. Park, and J. G. Shon, *Design and implementation of a reliable message transmission system based on MQTT protocol in IoT*. Wireless Personal Communications, 91, 1765-1777, 2016. https://doi.org/10.1007/s11277-016-3398-2.

[7] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*. Journal of the ACM (JACM), 32(2), 374-382, 1985. https://doi.org/10.1007/978-981-95-5116-3_1.

[8] G. Taubenfeld, *Weak failures: Definitions, algorithms and impossibility results. In International Conference on Networked Systems*, 51-66, 2018. https://doi.org/10.1007/978-3-030-05529-5_4.

[9] T. D. Chandra, and S. Toueg, *Unreliable failure detectors for reliable distributed systems*. Journal of the ACM (JACM), 43(2), 225-267, 1996. https://doi.org/10.1007/3-540-46691-6_26.

[10] R. Guerraoui, and A. Schiper, *The generic consensus service*. IEEE Transactions on Software Engineering, 27(1), 29-41, 2001. https://doi.org/10.1109/32.895986.

[11] A. Durand, M. Raynal, and G. Taubenfeld, *Contention-related crash failures: Definitions, agreement algorithms, and impossibility results*. Theoretical Computer Science, 909, 76-86, 2022. https://doi.org/10.1016/j.tcs.2022.01.029.

[12] R. Ekwall, and A. Schiper, *Comparing atomic broadcast algorithms in high latency networks*, Technical Report LSR-Report-2006-003, EPFL, Switzerland 2006.

[13] P. Urban, N. Hayashibara, A. Schiper, and T. Katayama, *Performance comparison of a rotating coordinator and a leader-based consensus algorithm*. In Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 4-17, 2004. https://doi.org/10.1109/RELDIS.2004.1352999.

[14] A. Mostefaoui, and M. Raynal, *Low-cost consensus-based atomic broadcast*. In Proceedings. 2000 Pacific Rim International Symposium on Dependable Computing, 45-52, 2000. https://doi.org/10.1109/PRDC.2000.897283.

[15] R. Ekwall, and A. Schiper, *Solving atomic broadcast with indirect consensus*. In International Conference on Dependable Systems and Networks (DSN 06), 156-165, 2006. https://doi.org/10.1109/DSN.2006.65.

[16] L. Rodrigues, and M. Raynal, *Atomic broadcast in asynchronous crash-recovery distributed systems*. In Proceedings 20th IEEE international conference on distributed computing systems, 288-295,2000. https://doi.org/10.1109/TKDE.2003.1206269.

[17] S. Mena, and A. Schiper, *A new look at atomic broadcast in the asynchronous crash-recovery model*. In 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05), 202-211, 2005. https://doi.org/10.1109/RELDIS.2005.6.

[18] W. Chen, and B. S. Center, *Abortable consensus and its application to probabilistic atomic broadcast*. Technical Report MSR-TR-2006-135, 2007.

[19] F. Pedone, and A. Schiper, *Optimistic atomic broadcast*. In Distributed Computing: 12th International Symposium, DISC'98 Andros, Greece, 318-332, 1998. https://doi.org/10.1007/11523468_17.

[20] C. E. Bezerra, F. Pedone, B. Garbinato, and C. Geyer, *Optimistic atomic multicast*. In 2013 IEEE 33rd International Conference on Distributed Computing Systems, 380-389, 2013. https://doi.org/10.1109/ICDCS.2013.46.

[21] E. Jimenez, J. L. Lopez-Presa, and M. Patino-Martinez, *Uniform atomic broadcast and consensus in fully anonymous synchronous*

systems with crash failures. Computing, 105(6), 1165-1187, 2023. https://doi.org/10.1007/s00607-022-01135-9.

[22] P. Urban, I. Shnayderman, and A. Schiper, *Comparison of Failure Detectors and Group Membership: Performance Study of Two Atomic Broadcast Algorithms*. In 2003 International Conference on Dependable Systems and Networks, 645-654, 2003. https://doi.org/10.1109/DSN.2003.1209974.

[23] X. Wang, X. Sui, and S. Duan, *Otter: Scalable Sharding-Based Atomic Broadcast with Abortable Fork Detection*. Cryptology ePrint Archive, 2025. https://eprint.iacr.org/2025/740.

[24] S. Xie, D. Kang, H. Lyu, J. Niu, and M. Sadoghi, *Fides: Scalable Censorship-Resistant DAG Consensus via Trusted Components*. arXiv preprint arXiv:2501.01062, 2025. https://doi.org/10.48550/arXiv.2501.01062.

[25] L. Beltrando, M. Potop-Butucaru, and J. Alfaro, *TenderTee: Increasing the Resilience of Tendermint by using Trusted Environments*. International Conference of Distributed Computing and Networking, 2023. https://doi.org/10.1145/3571306.3571394.

[26] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, *An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends*. In 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 557-564, 2017. https://doi.org/10.1109/BIGDATACONGRESS.2017.85.

[27] M. Mededjel, G. Belalem, F. Z. Nesrine Benadda, and S. Kadakelloucha, *A blockchain application prototype for the internet of things*. Journal of Communications Software and Systems, 18(2), 124-136, 2022. https://doi.org/10.24138/jcomss-2021-0129.

[28] N. S. Sony, X. Ding, and M. Singhal, *A Committee Based Optimal Asynchronous Byzantine Agreement Protocol W.P.1*. In Proc. 26th Int. Conf. on Distributed Computing and Networking (ICDCN), 282-283, Jan. 2025. https://arxiv.org/abs/2410.23477.

[29] X. Sui, X. Wang, and S. Duan, *Signature-Free Atomic Broadcast with Optimal $O(n^2)$ Messages and $O(1)$ Expected Time*. In Proc. 2025 IEEE Symposium on Security and Privacy (SP), 1547-1565, IEEE, May 2025. https://eprint.iacr.org/2023/1549.

[30] S. Gupta, D. Kang, D. Malkhi, and M. Sadoghi, *Brief Announcement: Carry the Tail in Consensus Protocols*. In Proc. 39th Int. Symp. on Distributed Computing (DISC 2025), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025. https://doi.org/10.4230/LIPIcs.DISC.2025.59.

[31] I. Amores-Sesar, C. Cachin, J. Villacis, and L. Zanolini, *DAG-based Consensus with Asymmetric Trust*. In Proc. ACM Symposium on Principles of Distributed Computing (PODC), 151-161, June 2025. https://doi.org/10.1145/3732772.3733527.

[32] L. V. Ruchel, E. T. de Camargo, L. A. Rodrigues, R. C. Turchetti, L. Arantes, and E. P. Duarte Jr., *Scalable Atomic Broadcast: A Leaderless Hierarchical Algorithm*. Journal of Parallel and Distributed Computing, vol. 184, 104789, 2024. https://doi.org/10.1016/j.jpdc.2023.104789.

[33] M. Bravo, G. Chockler, A. Gotsman, A. Naser-Pastoriza, and C. Roldán, *Vertical Atomic Broadcast and Passive Replication*. In Proc. 38th Int. Symp. on Distributed Computing (DISC 2024), vol. 319, LIPIcs, 1-10, 2024. https://doi.org/10.4230/LIPIcs.DISC.2024.10.

[34] I. Amores-Sesar, V. Grøndal, A. Holmgård, and M. Ottendal, *DAG It Off: Latency Prefers No Common Coins*. In Proc. 39th Int. Symp. on Distributed Computing (DISC 2025), vol. 356, LIPIcs, 1-5, 2025. https://doi.org/10.4230/LIPIcs.DISC.2025.5.

[35] T. D. Chandra, V. Hadzilacos, and S. Toueg, *The weakest failure detector for solving consensus*. Journal of the ACM (JACM), 43(4), 685-722, 1996. https://doi.org/10.1145/234533.234549.

[36] V. Hadzilacos, and S. Toueg, *A modular approach to fault-tolerant broadcasts and related problems*. Technical Report, Cornell University, 1994. https://hdl.handle.net/1813/6207.

[37] A. Mostefaoui, and M. Raynal, *Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach*. In Distributed Computing: 13th International Symposium, DISC'99 Bratislava, 49-63, 1999. https://doi.org/10.1007/3-540-48169-9_4.

Nadjette Rebouh is currently a teacher researcher at the University of Bejaia (Algeria). She received the Master Degree in 2007 in computer science from the University of Bejaia (Algeria). Currently, she is a Lecturer at the University of Bejaia, Algeria and she is a member of the research team EPSIRT (Performance Evaluation of Computer Systems and Telecommunication Networks) since 2008 at the laboratory LAMOS of the University of Bejaia. She worked in the area of Distributed Computing, Blockchains, AI, IOT.

Louiza Bouallouche-Medjkoune is currently a Professor at the University of A.Mira (Bejaia, Algeria). She received the engineer degree in computer science from the University of Sétif (Algeria) and the Master Degree in applied mathematics from the University of Bejaia (Algeria). She received her Ph.D in 2006 in computer science from the University of Setif (Algeria) and the HDR from the University of Constantine (Algeria) in 2009. She works as a teacher at the department of Computer Science at the University of Bejaia (for Data Structures, Programming and Algorithmic, Performance evaluation, Queuing Theory and Markov chains, Simulation of Systems and Networks, Seminar on Performance Evaluation of Networks and Systems). She is head of the research team EPSIRT (Performance Evaluation of Computer Systems and Telecommunication Networks) since 2005 at the laboratory LAMOS of the University of Bejaia. Her research interests are in: performance Evaluation of Computer systems and telecommunication networks, Stability of Systems, Markov chains, Queuing Theory, Computer Networks (wired, wireless), Quality of Service of Networks and Systems, Routing and Protocols.