Quantitative Analysis of Consensus Algorithms

Fatemeh Borran, Martin Hutle, Nuno Santos, André Schiper

Abstract—Consensus is one of the key problems in fault-tolerant distributed computing. Although the solvability of consensus is now a well-understood problem, comparing different algorithms in terms of efficiency is still an open problem. In this paper, we address this question for round-based consensus algorithm using communication predicates, on top of a partially synchronous system that alternates between good and bad periods (synchronous and non-synchronous periods). Communication predicates together with the detailed timing information of the underlying partial-synchronous system provide a convenient and powerful framework for comparing different consensus algorithms and their implementations. This approach allows us to quantify the required length of a good period to solve a given number of consensus instances. With our results, we can observe several interesting issues, e.g., that the number of rounds of an algorithm is not necessarily a good metric for its performance.


1 INTRODUCTION

Consensus is one of the key problems in fault-tolerant distributed computing. The problem is related to replication and appears when implementing atomic broadcast, group membership, or similar services. Consensus is defined over a set of processes $\Pi$, where each process $p_i \in \Pi$ has an initial value $v_i$: all processes must agree on a common value that is the initial value of one of the processes.

Consensus cannot be solved deterministically in an asynchronous system with faults, as established by the FLP impossibility result [9]. Later it was shown that consensus can be solved in a partially synchronous system with a majority of correct processes [8]. Roughly speaking, a partially synchronous system is initially asynchronous, but eventually becomes synchronous; links may be initially lossy, but eventually become reliable. The failure detector model was introduced a few years later [3]. The model is defined as an asynchronous system “augmented” with a device called failure detector, defined by some completeness and accuracy properties (see [3] for details). Over the years the failure detector model has become very popular.

Now that solving consensus is well understood, it remains to understand the efficiency of consensus algorithms. In other words, a quantitative comparison of consensus algorithms is relevant. Existing work is referenced in Section 2; the paper goes beyond this work and proposes a more detailed timing analysis of some consensus algorithms. In the context of a partially synchronous system that alternates between periods of synchrony and periods of asynchrony, the paper compares, for various consensus algorithms, the window of synchrony that allows processes to decide. Such a timing analysis requires a model with time, which explains our choice of the partially synchronous model. Moreover, in order to decouple the timing analysis from irrelevant details of consensus algorithms, we do our analysis for a round-based model built on top of a partially synchronous system [8]. Such a modular approach allows us not only to reuse the same timing analysis for different consensus algorithms, but also to compare various round implementations for the same round-based consensus algorithm. Note that our results do not necessarily apply to non round-based algorithms, e.g., to consensus protocols driven by message reception.

Specifically, we express and analyze consensus algorithms in the round-based model described in [5], which combines the transmission fault model of Santoro and Widmayer [15] with communication predicates introduced by Gafni [10]. For each consensus algorithm and different round implementations, we express the minimal period of synchrony that allows the algorithm to solve $x$ instances of consensus as $\text{initialization} + x \cdot \text{per-consensus}$. The initialization time is a one time duration that allows processes to synchronize to a specific round of the consensus algorithm; per-consensus is the recurring duration for solving one instance of consensus. This allows us to highlight two extreme cases: “short” and “long” periods of synchrony. If the period of synchrony is long, the initialization cost is amortized over all instances of consensus, and can thus be ignored. This is not the case if the period of synchrony is short.

One important observation from our results is that the number of rounds of an algorithm is not necessarily a good metric for its performance. This justifies the fine performance analysis done in the paper. Our results allow us also to quantify the influence of the clock precision. We show that a large clock skew, as it is the case when using e.g. step counting, has only limited influence for algorithms that try to resynchronize in every round, but
The problem of analytical quantitative evaluation of consensus algorithms was initially addressed in [8], in the context of a partially synchronous system. For the algorithms proposed in the paper, the authors compute upper bounds for the time needed after GST (Global Stabilization Time) for all correct processes to decide. However, the algorithms in [8] and ours are different.\footnote{The algorithms [8] have nowadays been replaced with algorithms à la Paxos.} Moreover, rounds in [8] are implemented on top of a clock synchronization algorithm, and while the algorithms are polynomial in the constants \(n, \Delta\) and \(\Phi\), the authors made no effort to optimize these constants. This makes the comparison with our work hard. After this early work, analytical performance evaluation of consensus algorithms has not received much attention for a while. This is probably due to the advent of failure detectors, which led to consider an asynchronous system as the underlying model, and to ignore timing analysis. One of the first papers to reinitiate analytical performance study of consensus algorithms in non-synchronous systems is [16]. The paper considers failure detectors, and uses as metric the minimum number of communication steps for deciding in a “nice” run (run with no crashes, no false suspicions).

Later, [6], [11], and [1] study the performance of consensus algorithms expressed in a round-based computational model. The performance metric is the number of rounds needed for processes to decide once the system has become synchronous. However, as pointed out in [11], the efficiency expressed in terms of number of rounds, does not predict the time it takes to decide after the system stabilizes. This observation is not followed in [11] by any analysis, even though the authors note that this is an interesting subject for further studies. Such a timing analysis is done in [7] for a modified version of Paxos. The authors show that, with their modified Paxos algorithm, consensus can be solved in \(O(\delta)\) after the system stabilizes (actually \(17\delta\)), where \(\delta\) is the upper bound on message delivery time after stability is reached (\(\delta\) includes the time needed to process a message after reception). Timing analysis is also done in [14] for Paxos, but only for an execution started during a good period, and leader election outside of the Paxos algorithm (it uses a failure detector implementation in which every process sends periodically messages to all).

### 3 Background

#### 3.1 Round-based model and consensus

We consider a round-based computational model in order to express our consensus algorithms. The round-based model was introduced in [8], as a convenient computational model on top of a partially synchronous system model. The round-based model was later extended by Gafni [10] with the notion of predicates. In [5] it was shown how a round-based model extended with predicates can unify all benign faults. We use here the notations from [5].

In a round-based model, an algorithm consists, for each round \(r\) and process \(p \in \Pi\), of a sending function \(S^r_p\) and a transition function \(T^r_p\). Let \(s_p\) denote the current state of process \(p\). For each round \(r\) and each \(p\), the sending function \(S^r_p(s_p)\) determines a vector of messages to be sent, one message for each process (\(null\) if there is no message for this process). At the end of a round \(r\), \(p\) makes a state transition according to \(T^r_p(\bar{\mu}, s_p)\), where \(\bar{\mu}\) is the partial vector of messages received in round \(r\). Rounds are communication-closed: a message sent in round \(r\) to \(q\) and not received by \(q\) in round \(r\) is lost.

We denote by \(\text{HO}(p,r)\) the set of processes (including itself) from which \(p\) receives a message at round \(r\): \(\text{HO}(p,r)\) is the heard of set of \(p\) in round \(r\). If \(q \notin \text{HO}(p,r)\), then the message sent by \(q\) to \(p\) in round \(r\) was subject to a transmission failure. Communication predicates are expressed over the sets \(\text{HO}(p,r)\). Communication predicates restrict transmission failures, for example, the predicate \(\forall p, \forall r : |\text{HO}(p,r)| > n/2\) ensures that every process receives at least \(n/2\) messages in every round.

Let \(A = (S^r_p,T^r_p)\) be an HO algorithm and \(P\) a communication predicate. The tuple \((A,P)\) specifies an HO machine. An HO machine can be used to solve a problem. We consider the consensus problem in the paper. With consensus, each process \(p\) has an initial value \(v_p\) and decides irrevocably. The problem is specified by the following conditions:

- **Integrity**: Any decision value is the initial value of some process.
- **Agreement**: No two processes decide differently.
- **Termination**: All processes eventually decide.

A coordinated HO machine (CHO) is an extension of an HO machine that includes the notion of coordinator. This allows the specification of coordinator-based algorithms, by giving predicates not only over the HO sets but also over the current coordinator. In a CHO machine,
Coord(p, r) denotes the process that p considers to be the coordinator at round r, henceforth called the coordinator of p in round r. The functions SP and TP take the current coordinator as an additional parameter, reflecting the fact that the messages to be sent and the state transitions depend also on the coordinator.

Consensus algorithms, including coordinator-based algorithms, consist of a sequence of one or more rounds that are repeatedly executed. This sequence of one or more rounds is called a phase. Typically, the coordinator is changed only at the beginning of a phase. This is the case of all the coordinated algorithms we consider here; therefore we will use the notation Coord(p, φ) to refer to the coordinator of process p during all the rounds of phase φ.

3.2 Consensus algorithms analyzed

In this paper, we analyze three consensus algorithms that are safe by design, (i.e., they never violate the integrity or agreement properties of consensus), but require some predicate to ensure liveness.

3.2.1 Variant of Paxos: LastVoting in four rounds (LV-4)

The first consensus algorithm we consider is a variant of Paxos [12], called LastVoting [5], see Algorithm 1. It is a variant of Paxos in the sense that the algorithm is expressed here in a round model, which is not the way Paxos has been expressed [12]. LastVoting is coordinator-based, and each phase of LastVoting consists of four rounds 4φ−3 to 4φ, where φ denotes the current phase. Roughly speaking, round 4φ−3 corresponds to phase 1b of Paxos, round 4φ−2 to phase 2a, and round 4φ−1 to phase 2b. Phase 1a of Paxos is hidden in the implementation of round 4φ0, for leader election. The termination property of consensus is guaranteed by the existence of a phase φ such that following predicate holds:

\[ \mathcal{P}_{last}(\phi) :: \exists \Pi_0 \subseteq \Pi \text{ s.t. } |\Pi_0| > n/2, \exists c \in \Pi, \forall p \in \Pi_0 : \]
\[ \text{Coord}(p, \phi) = c \land \]
\[ |\text{HO}(c, 4\phi - 3)| > n/2 \land c \in \text{HO}(p, 4\phi - 2) \land \]
\[ \Pi_0 \subseteq \text{HO}(c, 4\phi - 1) \land c \in \text{HO}(p, 4\phi). \]

which ensures, loosely speaking, agreement on the coordinator c during one phase φ, and communication between c and a majority of processes during phase φ.

3.2.2 Variant of Paxos: LastVoting in three rounds (LV-3)

LastVoting in three rounds is a well-known variant of Paxos in which the last two rounds of a phase are aggregated in a single round [12], as shown by Algorithm 2: in round 3φ all processes send their ack message directly to all other processes, instead of via the coordinator. LV-3 terminates in a phase φ satisfying the following predicate:

\[ \mathcal{P}_{last}(\phi) :: \exists \Pi_0 \subseteq \Pi \text{ s.t. } |\Pi_0| > n/2, \exists c \in \Pi, \forall p \in \Pi_0 : \]
\[ \text{Coord}(p, \phi) = c \land |\text{HO}(c, 3\phi - 2)| > n/2 \land \]
\[ c \in \text{HO}(p, 3\phi - 1) \land \Pi_0 \subseteq \text{HO}(p, 3\phi). \]

\[ \text{if } p = \text{Coord}(p, \phi) \text{ and number of } (\langle v, \theta \rangle) \text{ received } > n/2 \text{ then } \]
\[ \text{let } \theta \text{ be the largest } \theta \text{ from } (\langle -, \theta \rangle) \text{ received } \]
\[ \text{vote}_p := \text{one } \theta \text{ such that } (\langle \theta, \theta \rangle) \text{ is received } \]
\[ \text{commit}_p := \text{true} \]

\[ \text{Round } r = 4\phi - 1 : \]
\[ \text{if } ts_p = \phi \text{ then } \]
\[ \text{send } (\text{ack}) \text{ to Coord}(p, \phi) \]

\[ \text{Round } r = 3\phi : \]
\[ \text{if } p = \text{Coord}(p, \phi) \text{ and number of } (\langle \text{ack}, v \rangle) \text{ received } > n/2 \text{ then } \]
\[ \text{ready}_p := \text{true} \]

Algorithm 1 LV-4: LastVoting in four rounds [5].

1: Initialization: \[ x_p := v_p \in V \]
2: \[ v_p \in V \cup \{?) \}, \text{initially } ? \]
3: \[ \text{commit}_p \text{ a Boolean, initially } \text{false} \]
4: \[ \text{ready}_p \text{ a Boolean, initially } \text{false} \]
5: \[ ts_p \in \mathbb{N}, \text{initially } 0 \]
6: \[ \text{if } p = \text{Coord}(p, \phi) \text{ and number of } (\langle v, \theta \rangle) \text{ received } > n/2 \text{ then } \]
7: \[ \text{let } \theta \text{ be the largest } \theta \text{ from } (\langle -, \theta \rangle) \text{ received } \]
8: \[ \text{vote}_p := \text{one } \theta \text{ such that } (\langle \theta, \theta \rangle) \text{ is received } \]
9: \[ \text{commit}_p := \text{true} \]

\[ \text{Round } r = 4\phi - 2 : \]
10: \[ \text{send } (\text{ack}) \text{ to all processes} \]
11: \[ \text{for } p = \text{Coord}(p, \phi) \text{ and number of } (\langle \text{ack}, v \rangle) \text{ received } > n/2 \text{ then } \]
12: \[ \text{ready}_p := \text{true} \]

\[ \text{Round } r = 4\phi - 1 : \]
13: \[ \text{if } ts_p = \phi \text{ then } \]
14: \[ \text{send } (\text{ack}) \text{ to all processes} \]
15: \[ \text{for } p = \text{Coord}(p, \phi) \text{ and number of } (\langle \text{ack}, v \rangle) \text{ received } > n/2 \text{ then } \]
16: \[ \text{ready}_p := \text{true} \]

\[ \text{Round } r = 4\phi : \]
17: \[ \text{if } p = \text{Coord}(p, \phi) \text{ and number of } (\langle \text{ack}, v \rangle) \text{ received } > n/2 \text{ then } \]
18: \[ \text{ready}_p := \text{true} \]

Algorithm 2 LV-3: LastVoting in three rounds [4].

1: Initialization: \[ x_p := v_p \in V \]
2: \[ v_p \in V \cup \{?) \}, \text{initially } ? \]
3: \[ \text{commit}_p \text{ a Boolean, initially } \text{false} \]
4: \[ ts_p \in \mathbb{N}, \text{initially } 0 \]

\[ \text{Round } 3\phi - 2 : \text{ identical to round } 4\phi - 3 \text{ of Algorithm 1.} \]

\[ \text{Round } 3\phi - 1 : \text{ identical to round } 4\phi - 2 \text{ of Algorithm 1.} \]

\[ \text{Round } 3\phi : \]
7: \[ \text{if } ts_p = \phi \text{ then } \]
8: \[ \text{send } (\text{ack}) \text{ to all processes} \]
9: \[ \text{for } p = \text{Coord}(p, \phi) \text{ and number of } (\langle \text{ack}, v \rangle) \text{ received } > n/2 \text{ then } \]
10: \[ \text{ready}_p := \text{true} \]
11: \[ \text{for } p = \text{Coord}(p, \phi) \text{ and number of } (\langle \text{ack}, v \rangle) \text{ received } > n/2 \text{ then } \]
12: \[ \text{DECIDE}(v) \]
13: \[ \text{commit}_p := \text{false} \]

3.2.3 OneThirdRule (OTR)

Contrary to LV-4 and LV-3, which are both coordinator-based algorithms, Algorithm 3 does not use a coordinator. This algorithm, called OneThirdRule (or simply OTR), appears in [5]. It has similarities with a fast round of the Fast Paxos algorithm [13]. Every round of OTR has the same sending and transition function. Decision can be reached in one round if all initial values are identical,
In this paper we are interested in the question of how an HO machine \( \mathcal{A} \), where \( \mathcal{P} \) is some predicate, can be implemented in a “classical” message-passing model. Fig. 1 illustrates how these parts work together in a system. The top layer, the HO Algorithm \( \mathcal{A} \), is defined solely in terms of the sending function \( S_{\mathcal{P}} \) and transition function \( T_{\mathcal{P}} \), and assumes some communication predicate \( \mathcal{P} \). The communication predicate \( \mathcal{P} \) is implemented by the Predicate Implementation layer, which builds on top of the system model. These two layers are independent, apart from the interface defined by the communication predicate. This enforces a clear separation between the high-level computational model of the HO Algorithm and the low-level system model and allows each layer to be developed independently. In the rest of this paper, we give implementations for the communication predicates specified above (summarized in Table 1).

Given an implementation for some predicate \( \mathcal{P} \), we are looking for the length of a good-period, i.e., the duration our system has to be synchronous at some arbitrary point in time in order to ensure the predicate.

Note that for coordinator-based algorithms, the predicate implementation layer is also responsible for electing the coordinator. This is in contrast to failure detector based solutions, in which the failure detectors (or the leader election oracle) are provided by some external service. Such a service typically uses heartbeat messages. No such external service is used here. The difficulty is, during a good period, to elect a common coordinator, resynchronize the processes and exchange the necessary messages to ensure the predicate, within a time as short as possible, and using as few messages as possible.

### 4 System Model

We describe now the system model for the implementation of the predicate layer. We consider a similar model as in [8], with some modification to reflect good periods of bounded length. Further, we use clocks instead of a bound on the maximum speed of processes, a more general approach, as we explain later.

Let \( \Pi = \{p_1, \ldots, p_n\} \) be the set of processes, with \( n > 2 \). Processes are connected by a communication network, modeled for each \( p \in \Pi \) by a variable \( \text{buffer}_p \), which contains all messages that have been sent to \( p \) but were not yet received by \( p \). Processes proceed by making steps, where a step is either a receive step or a send step:

- In a send step, a single message can be sent to another process in the system, that is, when process \( p \) executes \( \text{send}(\langle m, q \rangle) \), the tuple \( \langle m, p \rangle \) is placed in \( \text{buffer}_q \).
- In a receive step, some messages are received, that is, when process \( p \) executes \( \text{receive}(S) \), a set \( S \subseteq \text{buffer}_p \) is removed from \( \text{buffer}_p \), and delivered to \( p \). Note that \( S \) may be empty.

In each step, some computation can be done, and the local clock \( C_p(t) \) of process \( p \) at real time \( t \) can be read. We assume that local clocks are monotonically non-decreasing at any time. Real time and the local clock take values from \( \mathbb{R} \).

### 3.3 Implementation of predicates

In this paper we are interested in the question of how an HO machine \( (\mathcal{A}, \mathcal{P}) \), where \( \mathcal{P} \) is some predicate, can be implemented in a “classical” message-passing model. Fig. 1 illustrates how these parts work together in a system. The top layer, the HO Algorithm \( \mathcal{A} \), is defined solely in terms of the sending function \( S_{\mathcal{P}} \) and transition function \( T_{\mathcal{P}} \), and assumes some communication predicate \( \mathcal{P} \). The communication predicate \( \mathcal{P} \) is implemented by the Predicate Implementation layer, which builds on top of the system model. These two layers are independent, apart from the interface defined by the communication predicate. This enforces a clear separation between the high-level computational model of the HO Algorithm and the low-level system model and allows each layer to be developed independently. In the rest of this paper, we give implementations for the communication predicates specified above (summarized in Table 1).

Given an implementation for some predicate \( \mathcal{P} \), we are looking for the length of a good-period, i.e., the duration our system has to be synchronous at some arbitrary point in time in order to ensure the predicate.

Note that for coordinator-based algorithms, the predicate implementation layer is also responsible for electing the coordinator. This is in contrast to failure detector based solutions, in which the failure detectors (or the leader election oracle) are provided by some external service. Such a service typically uses heartbeat messages. No such external service is used here. The difficulty is, during a good period, to elect a common coordinator, resynchronize the processes and exchange the necessary messages to ensure the predicate, within a time as short as possible, and using as few messages as possible.

### 4 System Model

We describe now the system model for the implementation of the predicate layer. We consider a similar model as in [8], with some modification to reflect good periods of bounded length. Further, we use clocks instead of a bound on the maximum speed of processes, a more general approach, as we explain later.

Let \( \Pi = \{p_1, \ldots, p_n\} \) be the set of processes, with \( n > 2 \). Processes are connected by a communication network, modeled for each \( p \in \Pi \) by a variable \( \text{buffer}_p \), which contains all messages that have been sent to \( p \) but were not yet received by \( p \). Processes proceed by making steps, where a step is either a receive step or a send step:

- In a send step, a single message can be sent to another process in the system, that is, when process \( p \) executes \( \text{send}(\langle m, q \rangle) \), the tuple \( \langle m, p \rangle \) is placed in \( \text{buffer}_q \).
- In a receive step, some messages are received, that is, when process \( p \) executes \( \text{receive}(S) \), a set \( S \subseteq \text{buffer}_p \) is removed from \( \text{buffer}_p \), and delivered to \( p \). Note that \( S \) may be empty.

In each step, some computation can be done, and the local clock \( C_p(t) \) of process \( p \) at real time \( t \) can be read. We assume that local clocks are monotonically non-decreasing at any time. Real time and the local clock take values from \( \mathbb{R} \).

### 3.3 Implementation of predicates

In this paper we are interested in the question of how an HO machine \( (\mathcal{A}, \mathcal{P}) \), where \( \mathcal{P} \) is some predicate, can be implemented in a “classical” message-passing model. Fig. 1 illustrates how these parts work together in a system. The top layer, the HO Algorithm \( \mathcal{A} \), is defined solely in terms of the sending function \( S_{\mathcal{P}} \) and transition function \( T_{\mathcal{P}} \), and assumes some communication predicate \( \mathcal{P} \). The communication predicate \( \mathcal{P} \) is implemented by the Predicate Implementation layer, which builds on top of the system model. These two layers are independent, apart from the interface defined by the communication predicate. This enforces a clear separation between the high-level computational model of the HO Algorithm and the low-level system model and allows each layer to be developed independently. In the rest of this paper, we give implementations for the communication predicates specified above (summarized in Table 1).

Given an implementation for some predicate \( \mathcal{P} \), we are looking for the length of a good-period, i.e., the duration our system has to be synchronous at some arbitrary point in time in order to ensure the predicate.

Note that for coordinator-based algorithms, the predicate implementation layer is also responsible for electing the coordinator. This is in contrast to failure detector based solutions, in which the failure detectors (or the leader election oracle) are provided by some external service. Such a service typically uses heartbeat messages. No such external service is used here. The difficulty is, during a good period, to elect a common coordinator, resynchronize the processes and exchange the necessary messages to ensure the predicate, within a time as short as possible, and using as few messages as possible.
Definition 1 (Δ-timely message): A message m sent at time t by some process to a process p is called Δ-timely, if it is received at the latest by the first receive step of p at or after time t + Δ.

A link between processes p and q is said to be Δ-timely in an interval I if every message sent by p to q at a time t ∈ I is a Δ-timely message, provided that t + Δ ∈ I.

Process synchrony is ensured by making steps at a minimum rate. Note that in contrast to [8], there is no restriction on the maximum speed of processes, since we will use clocks instead of step counting in order to measure time.

Definition 2 (Φ-synchronous process in interval I): The bound Φ is said to hold for a process p in some time interval I, if in any sub-interval of length Φ, p takes at least one step.

Definition 3 (local (α, β) bounded-drift clock in interval I): A local clock C_p(t) has a bounded-drift in a time interval I, if there are a priori known constants α and β with 0 < α ≤ β, so that for any two times t_1, t_2 ∈ I s.t. 0 ≤ t_1 < t_2:

\[ \frac{C_p(t_2) - C_p(t_1)}{t_2 - t_1} \in [α, β]. \] (1)

Note that our clock definition is very general, since it includes other assumptions like the classical bounded-drift clocks (α = 1 - ρ, β = 1 + ρ), whereas the values of α = 1/Φ, β = 1 are obtained asymptotically if step-counting is used for measuring time (this would require an upper bound on the frequency of steps, of course).

Definition 4 (good period): Let Π_0 ⊆ Π be a set of processes. An interval I is a good period for Π_0, if there are a priori known bounds Φ, Δ ∈ N, and α, β ∈ R, with Φ > 0, 0 < α ≤ β, such that (i) in I all processes in Π_0 are Φ-synchronous and have a local (α, β)-bounded-drift clock in I, (ii) no process that is not in Π_0 makes a step, (iii) no links between processes in Π_0 are Δ-timely, and (iv) no messages from processes not in Π_0 are received by a process in Π_0.

A k-good period is a good period for some arbitrary Π_0 with |Π_0| ≥ k. In the sequel, when k is clear from the context we will use only the term good period.

Note that we do not specify why processes outside Π_0 do not make steps; they might have crashed, be just temporarily unavailable, or be mute for any other reason. Therefore the notion of correct or faulty process is not suitable in our context; however, with respect to some Π_0-good period, we say a process is up in this good period iff it is in Π_0, else it is down.

Due to clock drift, a timeout measured by a process does not necessarily match the elapsed real time. Nevertheless, during good periods the clock drift is bounded, which allows us to bound the time measured by a process within a real time envelop. The following Lemma shows the relation between real time and process time, which will be used in the rest of the paper to set process timeouts:

Lemma 1: In a good period, a time interval of length τ_C = βτ_L, measured by some process p, corresponds to a real time interval of length in [τ_L, τ_U], with τ_U = \frac{β}{2}τ_L.

Proof: Let [t_1, t_2] be a real time interval. Then, C_p(t_2) - C_p(t_1) is the duration of the interval as measured by p, and t_2 - t_1 the real time duration. From equation (1), we have t_2 - t_1 ≥ |C_p(t_2) - C_p(t_1)|/β = τ_L, and t_2 - t_1 ≤ |C_p(t_2) - C_p(t_1)|/α = \frac{β}{2}τ_L, which proves the result.

We will keep the notation of τ_L, τ_C, and τ_U consistent within the paper to denote these different kind of durations.

5 The Generic Protocol

We give in this section a generic algorithm for the predicate layer, an algorithm that is parametrized by four abstract functions. The instantiation of these functions will allow us to devise three different algorithms for the predicate layer that differ mainly by the message pattern and the way the coordinator is elected. The first method is called Full Synchronization (Section 6); it ensures uniform rounds, therefore allowing the implementation of all predicates considered in this paper. Phase Synchronization (Section 7) is an optimized implementation for P_{LV3}, where round synchronization takes place only once per phase. Finally, Synchronization by a Coordinator (Section 8), where synchronization uses only messages from coordinator process(es), is specialized for P_{LV4}.

The generic Algorithm 4 follows the following pattern. One iteration of the while loop (line 6) corresponds to one round: the sending function is called at line 9 and the transition function is called at line 20. Messages are sent at line 14: the abstract function Dest (line 10) specifies the set of processes a message is sent to in the current round. Message reception occurs at line 17. The receive statement is executed repeatedly until NextRound returns true (line 16). This typically happens when a timer has expired or when a message from some higher round is received. Note also that some rounds may be totally skipped (no message sent, no message received): this happens whenever the function SkipRound (line 8) returns true, which typically occurs if process p in round r receives a message from some round r’ > r. In this case, p skips all rounds from r_p to r’ - 1. Finally, function ElectCoord specifies how a coordinator for each round is determined.

6 Full Synchronization

Our first implementation is given as Parametrization 1 for the generic algorithm. The implementation ensures uniform rounds in a good period, which “almost” ensures P_{LV4} (for LV-4) and P_{LV3} (for LV-3): only the election of the coordinator is missing. However, a uniform round r allows the election of a unique coordinator for round r + 1 (the coordinator can be determined through a deterministic function on the H0 sets of round r). Thus uniformity of round φ_0 and of the four rounds of phase φ_0 + 1 allows us to ensure P_{LV4}, and uniformity of round

•
there are two ways for a process $p$ and the first condition of
by expiration of a timeout. In case (i), the process goes
$\tau$ by receiving a message from a higher round
$P$ and $\tau$ of process $p$.
$\tau$ to ensure
instances of
in the HO set of round $r - 1$ (whenever this HO set is
non empty). Note the definition of the macro $ho(r)$ given
in the caption of Parametrization 1; we will also use this
notation in the following sections. This ensures a unique
coordinator in good periods where rounds are uniform.
For non-coordinated predicates, like $P_u()$, no coordinator
is needed and the function $ElectCoord$ can be ignored.

6.2 Timeout $\tau_C$
We first assume that a good period, which starts at some
time $t_g$, holds forever, and show that the timeout $\tau_C =
[2\Delta + (2n - 1)\Phi] \beta$ ensures $P_u()$ for processes that are up
in this good period. In the next subsection we compute
the length of a good period that is sufficient to ensure
$P_u()$.

Lemma 2 (Timeout $\tau_C$): Consider Parametrization 1
with the timeout $\tau_C = [2\Delta + (2n - 1)\Phi] \beta$. Assume that
a $k$-good period starts at time $t_g$ and holds forever, and
that round $r_0$ is the highest round started by any process
in $\Pi_0$ by time $t_g$. Then every new round $r > r_0$ started
after time $t_g$ is uniform ($P_u(r)$) with cardinality $k$.

Proof: We show that in round $r$, for every process $p \in
\Pi_0$, we have: (i) $p$ receives a message from all processes
in $\Pi_0$, but (ii) not from any process not in $\Pi_0$.

We start with (ii). Assume that $p$ received a round
message from a process $q$ that is not in $\Pi_0$. By
the definition of a good period, $p$ could not have received
this message after $t_g$. If $p$ had received this message before
$t_g$, then $p$ would have advanced to round $r$ immediately,
which contradicts our assumption that no process in $\Pi_0$
has entered round $r$ by $t_g$.

To prove (i), assume some process $p_1$ is the first to finish
sending its round $r$ messages at time $t_s > t_g$ (see Fig. 3).
These messages are ready for reception at each process in
$\Pi_0$ ($p_2$ in Fig. 3), the latest at $t_s + \Delta$, since messages are
$\Delta$-timely. These messages are received in the next receive
step, which occurs the latest after $n - 1$ send steps (in the
case the process was just starting executing send steps).
Since a step takes up to $\Phi$ time, $p_1$’s message is received
by all processes in $\Pi_0$ the latest at $t_s + \Delta + n\Phi$. Each process
that receives this message jumps to round $r$, if not already

3φ₀ and of the three rounds of phase φ₀ + 1 allows us
to ensure $P_{l:\diamond}$. More generally, 2y consecutive uniform
rounds ensure $y$ instances of $P_{l:\star}$, in the worst case $4y + 4$
consecutive uniform rounds ensure $y$ instances of $P_{l:u}$,
and $3y + 3$ consecutive uniform rounds ensure $y$ instances of
$P_{l:v}$.

With Parametrization 1, every process sends a message
to every other process in all rounds (see function $Dest$).
While sending to all in all rounds seems natural for
$P_{l:u}$, where every process has to hear from every alive
process, this induces some overhead for $P_{l:v}$ and $P_{l:u}$,
since these predicates only require one-to-all or all-to-
one patterns on some of their rounds. This is shown in
the following picture for predicate $P_{l:v}$, where the full
lines represent messages required by the predicate and
dotted lines represent the additional messages sent by
Parametrization 1:

Fig. 2. Message pattern of Full Synchronization

In the next sections, we provide implementations for
$P_{l:v}$ and $P_{l:u}$ with lower message complexity.

6.1 Outline of Full Synchronization
As shown by function $NextRound$ in Parametrization 1,
there are two ways for a process $p$ to leave round $r$: (i)
by receiving a message from a higher round $r' > r$, or (ii)
by expiration of a timeout. In case (i), the process goes
directly to round $r'$. Note that the function $SkipRound$
and the first condition of $NextRound$ play together to
achieve this. In case (ii) the timeout $\tau_C$ is chosen to ensure
uniformity, i.e., $P_u()$, in a good period (see Lemma 2
below). As shown by the function $ElectCoord$, the coordi-
nator for some round $r$ is the smallest process ($min$)
in the HO set of round $r - 1$ (whenever this HO set is
non empty). Note the definition of the macro $ho(r)$ given
in the caption of Parametrization 1; we will also use this
notation in the following sections. This ensures a unique
coordinator in good periods where rounds are uniform.
For non-coordinated predicates, like $P_u()$, no coordinator
is needed and the function $ElectCoord$ can be ignored.

Algorithm 4 Generic algorithm of the predicate layer

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$Rcv_p \leftarrow \emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>$r_p \leftarrow 1$</td>
</tr>
<tr>
<td>3</td>
<td>$s_p \leftarrow \text{init}_p$</td>
</tr>
<tr>
<td>4</td>
<td>$\text{coord}_p \leftarrow \perp$</td>
</tr>
<tr>
<td>5</td>
<td>$t_p \leftarrow \text{Cp}_p$</td>
</tr>
<tr>
<td>6</td>
<td>while true do</td>
</tr>
<tr>
<td>7</td>
<td>coord$_p \leftarrow \text{ElectCoord}(p, r_p, C_p() - t_p, \text{coord}_p, Rcv_p)$</td>
</tr>
<tr>
<td>8</td>
<td>if $\neg \text{SkipRound}(p, r_p, C_p() - t_p, \text{coord}_p, Rcv_p)$ then</td>
</tr>
<tr>
<td>9</td>
<td>msg$_s \leftarrow S^\star(s_p, \text{coord}_p)$</td>
</tr>
<tr>
<td>10</td>
<td>for all $q \in \text{Dest}(p, r_p, C_p() - t_p, \text{coord}_p, Rcv_p)$ do</td>
</tr>
<tr>
<td>11</td>
<td>if $p = q$ then</td>
</tr>
<tr>
<td>12</td>
<td>$Rcv_p \leftarrow Rcv_p \cup {\text{msgs}[p, r_p] }$</td>
</tr>
<tr>
<td>13</td>
<td>else</td>
</tr>
<tr>
<td>14</td>
<td>send(\text{msgs}[q], r_p, q)</td>
</tr>
<tr>
<td>15</td>
<td>$t_p \leftarrow \text{Cp}_p$</td>
</tr>
<tr>
<td>16</td>
<td>while $\neg NextRound(p, r_p, C_p() - t_p, \text{coord}_p, Rcv_p)$ do</td>
</tr>
<tr>
<td>17</td>
<td>receive(S)</td>
</tr>
<tr>
<td>18</td>
<td>for all messages $(x, r, q) \in S$ do</td>
</tr>
<tr>
<td>19</td>
<td>$Rcv_q \leftarrow Rcv_q \cup {(x, q, r)}$</td>
</tr>
<tr>
<td>20</td>
<td>$s_p \leftarrow T^\star{[(x, q) \mid (x, q, r) \in Rcv_p], s_p, \text{coord}_p}$</td>
</tr>
<tr>
<td>21</td>
<td>$r_p \leftarrow r_p + 1$</td>
</tr>
</tbody>
</table>
there, and thus, by time $t_s + \Delta + (2n-1)\Phi$ has performed $n-1$ send steps and has sent its round $r$ message to all. This message is ready for reception by the latest at time $t_e = t_s + 2\Delta + (2n-1)\Phi$.

The timeout $\tau_C = [2\Delta + (2n-1)\Phi] \beta$, together with Lemma 1, ensure that no timeout of length $\tau_C$ started at time $t_s$ expires before $t_e$. So when the timeout expires, all messages for round $r$ are either received or ready to be received. Before calling the transition function for round $r$, a receive step is performed; thus in round $r$ every process in $\Pi_0$ receives a message from every process in $\Pi_0$. 

6.3 Length of a good period

The next theorem computes the required duration of a good period in order to ensure $x$ consecutive uniform rounds. The special case $x = 0$ gives the initialization time, which for $P_{otr}$ is the time from the start of the good period until the beginning of the first uniform round, which is $\frac{\beta}{\alpha} (2\Delta + (2n-1)\Phi) + 2n\Phi + \Delta$. Also from the formula, we can compute the time required for each uniform round after stabilization, which is $\frac{\beta}{\alpha} (2\Delta + (2n-1)\Phi) + n\Phi$.

**Theorem 1:** In any good period of length

$$(x + 1) \left[ \frac{\beta}{\alpha} (2\Delta + (2n-1)\Phi) + n\Phi \right] + \Delta + n\Phi$$

the generic algorithm with Parametrization 1 ensures $x$ consecutive rounds that fulfill $P_u()$.

**Proof:** Assume a good period starts at time $t_g$ and at this time process $p_1$ has the highest round number $r$ among the processes in $\Pi_0$. We distinguish two cases: (i) $t_g$ is during these $n-1$ send steps (not shown in Fig. 4) of round $r = 3\Phi$. (ii) $t_g$ after these send steps (see Fig. 4). It can be shown that case (ii) is worse than case (i) in terms of length of the good period, thus we consider case (ii). Round $r+1$ is the first round that all processes in $\Pi_0$ start after $t_g$. According to Lemma 2, round $r+1$, $r+2$, etc. are uniform if the good period is long enough. We compute the maximum time it takes for any process $p_2$ to complete round $r + x$. As shown by Fig. 4, $p_2$ starts round $r+1$ at latest at time $t_g + \tau_U + 2n\Phi + \Delta$ (end of “initialization” in Fig. 4). This expression is obtained as follows: by the definition of $P_u$, no message of a round larger than $r$ is received before $p_1$’s timer expires, and $\tau_U = \frac{\beta}{\alpha} t_L$ is the time elapsed for a timeout $\tau_C = t_L \beta$; when the timeout expires, $p_1$ executes a receive step ($\phi$), moves to round $r+1$, executes $n-1$ send steps $((n-1)\Phi)$; in the worst case the message to $p_2$ is sent in the last of these send steps; $\Delta$ later the message is ready for reception on $p_2$; at this time $p_2$ may be executing $n$ send steps $((n-1)\Phi)$ before the reception step ($\Phi$) in which $p_1$’s message is finally received; at this point $p_2$ moves to round $r+1$.

We now show that case (i) leads to a shorter good period. Here, by time $t_g + (n-2)\Phi$, at least one message of round $r$ was sent by $p_1$. By time $t_g + (n-2)\Phi + \Delta + n\Phi$, this message is received by some process, and at the latest $(n-1)\Phi$ time later, this process has sent its round $r$ messages to all. Thus, after time $t_g + 2\Delta + (4n-4)\Phi$, every process has performed its send steps for round $r$. Consequently, doing now the same analysis as in case (ii), it cannot be the case anymore that a process is performing send steps when the message for round $r+1$ is ready for reception. This leads to the fact that also in this case $p_2$ starts round $r+1$ not after time $t_g + \tau_U + \Delta + n\Phi$.

Process $p_2$ needs at most $n\Phi + \tau_U$ to complete round $r+1$ (see “regular round” in Fig. 4): $n-1$ send steps $((n-1)\Phi)$, timeout $\tau_U$ (in the worst case no message of a larger round is received), one receive step ($\Phi$).

Summing up the duration of “initialization” and of $x$ “regular rounds” leads to $(x+1)[\tau_U + n\Phi] + \Delta + n\Phi$. Replacing $\tau_U$ with $\frac{\beta}{\alpha} t_L$ and $\tau_U$ with $2\Delta + (2n-1)\Phi$ (see Lemma 2) establishes the result.

As mentioned at the beginning of Section 6, in the worst case for $y$ instances of predicate $P_{otr}$ we need $2y$ uniform rounds, for $y$ instances of $P_{lv3}$ we need $3y + 3$ uniform rounds, and for $y$ instances of $P_{lv4}$ we need $4y + 4$ uniform rounds. It follows that the initialization time of LV-3 Full Sync corresponds to the initialization time to get uniform rounds, plus the duration of three uniform rounds. After initialization, each instance of LV-3 Full Sync requires three uniform rounds. Applying a similar reasoning to LV-4, we have:

**Corollary 1:** Let $\theta_3 = \frac{\beta}{\alpha} (2\Delta + (2n-1)\Phi) + n\Phi$. The the initialization time of LV-3 Full Sync, resp. LV-4 Full Sync, is $4\theta_3 + \Delta + n\Phi$, resp. $5\theta_3 + \Delta + n\Phi$. After initialization, the duration of one instance of LV-3 Full Sync, resp. LV-4 Full Sync, is $3\theta_3$, resp. $4\theta_3$.

7 Phase Synchronization

Full synchronization sends extra messages with respect to the “natural message pattern” induced by the predicates $P_{lv3}$ and $P_{lv4}$. In this section, we give an implementation for $P_{lv3}$, that uses only the “natural” messages, which are the following:

![Fig. 3. Full synchronization: Lemma 2.](image)

![Fig. 4. Full synchronization: Theorem 1.](image)
Parametrization 2 LV-3 using phase synchronization, where $ho(r) := \{q \mid <q,r> \in Rcv\}$

\[
\text{NextRound}(p, r, \tau, \text{coord}, Rcv) := \begin{cases} 
\exists (-,<r'>) \in Rcv : \ r' > r \\
\ r \ mod \ 3 = 1 \land (\tau \geq \tau_{C1} \lor |ho(r)| > n/2) \land r \ mod \ 3 = 2 \land \tau \geq \tau_{C2} \\
\ r \ mod \ 3 = 0 \land \tau \geq \tau_{C3}
\end{cases}
\]

\[
\text{SkipRound}(p, r, \tau, \text{coord}, Rcv) := \exists (-,<r'>) \in Rcv : \ r' > r 
\]

\[
\text{Dest}(p, r, \tau, \text{coord}, Rcv) := \begin{cases} 
\text{coord} : r \ mod \ 3 = 1 \\
\Pi : r \ mod \ 3 = 0 \lor (r \ mod \ 3 = 2 \land p = \text{coord}) \\
\emptyset : \text{else}
\end{cases}
\]

\[
\text{ElectCoord}(p, r, \tau, \text{coord}, Rcv) := \begin{cases} 
\min(\Pi) : r = 1 \\
\min(ho(r-1)) : r \ mod \ 3 = 1 \land ho(r-1) \neq \emptyset \\
\text{coord} : \text{else}
\end{cases}
\]

Here, processes are synchronized only at round $3\phi$ of every phase $\phi$. As in the case of full synchronization, round $3\phi$ allows the election of the coordinator for phase $\phi + 1$.

\subsection{Outline of Phase Synchronization}

The “natural” message pattern just depicted is generated by the function $Dest$ in Parametrization 2.

Round $3\phi$ of phase $\phi$ is identical to a round in the full synchronization case: all processes wait for the timeout before electing a new coordinator and moving to the first round of the next phase. According to function $NextRound$, only round $3\phi - 2$ (line 2 in $NextRound$) may be terminated by the reception of messages. Rounds $3\phi - 1$ and $3\phi$ terminate by expiration of the timeout as before.

Round $3\phi - 1$ requires some clarification. In this round processes only need to receive a message from the coordinator, thus it seems natural for a process to advance to round $3\phi$ as soon as it receives such message. But this solution is not correct as shown by the following scenario. Consider processes $p_1$, $p_2$, and $p_c$, with $p_c$ being the coordinator. Process $p_1$ receives $p_c$’s message for round $3\phi - 1$, advances to round $3\phi$, and sends its round $3\phi$ message to all. This message is delivered quickly to $p_2$, which receives it before the round $3\phi - 1$ message from $p_c$. If $p_2$ advances immediately to round $3\phi$, it will miss the round $3\phi - 1$ message from $p_c$ that arrives later. Algorithm 2 avoids this problem by delaying the start of round $3\phi$ until all processes had time to receive the round $3\phi - 1$ message from the coordinator. Another solution, based on piggybacking, is described at the end of Section 7.

\subsection{Timeouts $\tau_{C1}, \tau_{C2}, \tau_{C3}$}

**Lemma 3 (Timeout $\tau_{C1}$):** Consider Parametrization 2 with $\tau_{C1} = \left[2\Delta + (2n+1)\Phi\right] \beta + \tau_{C3} \left(\frac{\beta}{n} - 1\right)$. Assume every process starts round $3(\phi - 1)$ in a $(\frac{n+1}{2})$-good period and phase $\phi$ has a unique coordinator. Then the coordinator hears from a majority of processes in round $3\phi - 2$.

**Proof:** Let $p_1$ be the first process that starts the timeout for round $r = 3(\phi - 1)$ at time $t_{s3}$ (see Fig. 6). By time $t_{s3} + \Delta + n\Phi$ all other processes, e.g., $p_2$, are in round $3(\phi - 1)$. After sending their round $3(\phi - 1)$ messages, which takes at most $(n-1)\Phi$ time, their timeout $\tau_{C3} = \tau_{L3}\beta$ will expire by time $t_{s3} + \Delta + (2n-1)\Phi + \tau_{L3}\beta$. A receive and a send step later, each process has sent its round $3\phi - 2$ message to the coordinator ($p_1$ in Fig. 6), which is by time $t_{s3} + \Delta + (2n+1)\Phi + \tau_{L3}\beta$. This message is ready for reception at the coordinator $\Delta$ time later. Thus if the coordinator executes the receive step and the transition function for round $3\phi - 2$ not before time $t_{s2} = t_{s3} + 2\Delta + (2n+1)\Phi + \tau_{L3}\beta$, the set of messages passed to the transition function includes the messages of round $3\phi - 2$ from a majority.

Because $p_3$ is the first process to start the timeout for round $3(\phi - 1)$, no timeout $\tau_{C3}$ for this round expires at any process (including the coordinator) before $t_{s3} + \tau_{L3}$. Therefore, no process (including the coordinator) starts round $3\phi - 2$ before $t_{s1} = t_{s3} + \tau_{L3}$, and executes the transition function for round $3\phi - 2$ before $t_{s1} + \tau_{L1}$. The timeout $\tau_{C1}$ as given by the lemma ensures that $t_{s1} + \tau_{L1}$ is not before $t_{s2}$. Thus the coordinator $p_1$ receives the round $3\phi - 2$ messages from all processes in $\Pi_0$.

**Lemma 4 (Timeout $\tau_{C2}$):** Consider Parametrization 2 with the timeout $\tau_{C2} = \left[3\Delta + (3n+1)\Phi\right] \beta + \tau_{C3}\left(\frac{\beta}{n} - 1\right) - \tau_{C1}$. Assume a phase $\phi$ with a unique coordinator, where round $3(\phi - 1)$ starts in a $(\frac{n+1}{2})$-good period. Then in round $3\phi - 1$, every process in $\Pi_0$ hears from the coordinator.

**Proof:** Let $p_3$ be the first process that starts the timeout for round $r = 3(\phi - 1)$ at time $t_{s3}$. By a similar reasoning as for Lemma 3, each process has sent its round $3\phi - 2$ message to the coordinator by time $t_{s3} + \Delta + (2n+1)\Phi + \tau_{L3}\beta$. Then, at most $\Delta + \Phi$ later, the coordinator has received this message from every process in $\Pi_0$, thus
achieved the majority condition in line 2 of NextRound, and therefore sends its round 3φ − 1 message to all. The latter takes at most (n − 1)Φ time, and this message will be ready for reception Δ time later. At this time, 𝑡ₚ = 𝑡ₚ₃ + 3Δ + (3n + 1)Φ + τ₃L₃2, the timeout 𝜏₃C₂ may safely expire at any process. Because 𝑝₃ is the first process that starts round 3(φ − 1), no process starts round 3φ − 1 before 𝑡ₚ = 𝑡ₚ₃ + τ₃L₃ + τ₃L₁. Thus choosing τ₃C₂ as in the lemma ensures that τ₃C₂ does not expire before time 𝑡ₚ at any process.

Lemma 5 (Timeout τ₃C₃): Consider Parametrization 2 with the timeout τ₃C₁ = [2Δ + (2n − 1)Φ]β, assume that a k-good period starts at time 𝑡ₙ and that round 𝑟₀ is the highest round started by any process in Π₀ by time 𝑡ₙ. Then every new round 3φ > 𝑟₀ started after time 𝑡ₙ is uniform with cardinality ℎ.

Proof: Similar to Lemma 2.

Corollary 2: Parametrization 2 with the timeout τ₃C₁ = 2Φβ + 3(Δ + (2n − 1)Φ) and τ₃C₂ = [Δ + nΦ]β, and τ₃C₃ = [2Δ + (2n − 1)Φ]β ensures a ℘ₐₙₙ(φ), if round 3(φ − 1) starts in a (2Δ + 1)-good period.

Proof: By Lemma 5, round 3(φ − 1) is uniform. Thus by the definition of ElectCoord, every process in Π₀ has the same coordinator. Applying the Lemmas 3 to 5 and replacing the equations for τ₃C₁, τ₃C₂, and τ₃C₃ yields the result.

7.3 Length of a good period

We are now ready to compute the required duration of a good period in order to ensure y consecutive phases of ℘ₐₙₙ(ϕ). For the predicate ℘ₐₙₙ(ϕ), the initialization time is the time from the start of a good period until all processes start the first round of phase ϕ satisfying ℘ₐₙₙ(ϕ). This can be computed from the formula below, by setting y = 0. After initialization, the time required for each consensus instance is given by the multiplication factor of y.

Theorem 2: In any good period of length

\[
y \left(2Δ + (2n − 1)Φ\right) β^2 \alpha^2 + \left(3Δ + (3n + 1)Φ\right) β \alpha + (2n + 2)Φ\right] + \left(2Δ + (2n − 1)Φ\right) β^2 \alpha^2 + (5Δ + 5nΦ) β \alpha + Δ + 5nΦ\]

the generic algorithm with Parametrization 2 and timeouts according to Corollary 2 ensures y consecutive phases that fulfill ℘ₐₙₙ(ϕ).

Proof: It can be shown that the “initialization” period (see Fig. 7) is the longest in case 𝑡ₙ starts just after the first send step of some round 3(φ − 2), and round 3(φ − 2) is not uniform (only round 3(φ − 1) is uniform). The end of round 3(φ − 1) corresponds to the end of the “initialization” period. By time 𝑡ₙ + (2n + 1)Φ + τ₃U₃ + τ₃U₁ + τ₃U₂ round 3(φ − 1) is started at some process 𝑝₁. This round 3(φ − 1) ends for all processes in Π₀ at the latest (3n − 1)Φ + τ₃U₃ + Δ time later (end of “initialization” period) using the same argument as in Theorem 1. By Corollary 2, we have ℘ₐₙₙ(ϕ). Every “regular” phase then takes at most time τ₃U₁ + τ₃U₂ + τ₃U₃ + (2n + 2)Φ. The result follows by replacing the timeouts with the expressions from Corollary 2.

7.4 Piggybacking

We have explained in Section 7.1 why we choose round 3φ − 1 to terminate by the expiration of a timeout. The other solution requires some changes to our generic implementation. Thus we present only the overall idea and the results.

In this approach, a process p piggybacks all the messages it received for a round r on its message for round r + 1. If some process q receives the round r + 1 message from p before entering round r + 1, q can include these round r messages to its received set before ending round r. In some cases, this shortens the length of a good period.

In general, this mechanism can be used if all processes wait for the same quorum in some round, e.g., in the second round of LV-3, where all processes wait for a single message from the same process, i.e., the coordinator. By this optimization the length of a good period for LV-3 using phase synchronization can be reduced approximately by one Δ. The expression can be calculated by applying a similar analysis as before:

\[
y \left(2Δ + (2n − 1)Φ\right) β^2 \alpha^2 + Δ + 2Δ + (2n + 2)Φ\right] + \left(2Δ + (2n − 1)Φ\right) β^2 \alpha^2 + (5Δ + 5nΦ) β \alpha + Δ + 5nΦ\]

The other benefit of piggybacking is to speed up best case scenarios, where Π₀ = Π₁ in a good period. In this case, the implementation of the predicate does not rely on any timeout, and the length of a good period depends only on the actual transmission delay of messages, and no more on Δ. However, applying piggybacking in every round induces an important overhead and can considerably increase the effective message transmission delay.

8 Synchronization by a coordinator

If we use full synchronization or phase synchronization to implement ℘ₐₙₙ₄, LV-4 will never perform better than LV-3 in terms of message complexity or length of the good period, because LV-4 requires one more round per phase. In this section we give another implementation that achieves a message complexity of O(n) instead of O(n²) per regular phase during a good period, at the cost of a slightly larger length of the good period.

The predicate ℘ₐₙₙ₄, contrary to ℘ₐₙₙ₃, does not require any round where all processes hear from each other. Without such a round, we need to send additional messages in some round in order to synchronize processes and choose a coordinator, like we do for ℘ₐₙₙ₃. As for ℘ₐₙₙ₃ we do this only once per phase, in the last round of a phase. This leads to the following message pattern:

The messages represented by a full line are required by ℘ₐₙₙ₄, while the messages represented by a dotted line in round 4φ are only for synchronization and election of a coordinator.
8.1 Outline of Synchronization by a Coordinator

Our algorithm requires only $n$ messages in round $4\phi$ during a good period. This optimization is based on the following two observations: (i) to choose a coordinator, it is enough for the HO sets to be non-empty, as long as the round is uniform, and (ii) to synchronize to the same round in a good period it is enough if all processes receive a message from the process with the highest round.

Based on these observations, it is easy to see that in addition to the coordinator, only the processes that entered round $4\phi$ by timeout need to send a message to all (line 3 of \textit{Dest} in Parametrization 3). Otherwise, if a process $p$ receives a round $4\phi$ message while in a lower round, $p$ can advance to round $4\phi$ silently, since there is at least another process that sent a message to all in round $4\phi$ and, therefore, can be chosen as coordinator. This strategy results in a message complexity of $cn$ for the election in round $4\phi$, where $c$ is the number of processes that compete to become coordinator. After the first election in a regular phase during a good period we have $c = 1$, since all processes will receive a round $4\phi$ message from the coordinator while in round $4\phi - 1$.

To reduce the time needed to start a phase in which $P_{iv4}$ might hold, our algorithm skips some rounds of phase $\phi$ if it detects that $P_{iv4}(\phi)$ cannot hold. This can happen in two cases: (i) in round $4\phi - 2$ by the coordinator if it does not receive a majority of messages during round $4\phi - 1$ (\textit{SkipRound}, line 2), and (ii) in round $4\phi - 1$ by any process if it does not receive a message from the coordinator in round $4\phi - 2$ (\textit{SkipRound}, line 3).

8.2 Timeouts $\tau_{C1}$, $\tau_{C3}$, $\tau_{C4}$

Lemmas 6, 7 and 8 below establish results about timeouts $\tau_{C1}$, $\tau_{C3}$ and $\tau_{C4}$. The proofs are similar to those in Section 7, and can be found in [2].

Lemma 6 (Timeout $\tau_{C1}$): Consider Parametrization 3 with $\tau_{C1} = [3\Delta + 3n\Phi] + \tau_{C4} \left(\frac{n}{\alpha} - 1\right)$. Assume every process starts round $4(\phi - 1)$ in a $(\frac{n+1}{2})$-good period and phase $\phi$ has a unique coordinator $c$. Then (i) $c$ hears from a majority of processes in round $4\phi - 3$, and (ii) all processes in $\Pi_0$ hear from $c$ in round $4\phi - 2$.

Lemma 7 (Timeout $\tau_{C3}$): Consider Parametrization 3 with the timeout $\tau_{C3} = [3\Delta + 2n\Phi] + \tau_{C4}$. Assume every process starts round $4(\phi - 1)$ in a $(\frac{n}{2})$-good period, and $\phi$ has a unique coordinator $c$. Then (i) $c$ hears from a majority of processes in round $4\phi - 1$, and (ii) all processes in $\Pi_0$ hear from $c$ in round $4\phi$.

Lemma 8 (Timeout $\tau_{C4}$): Consider Parametrization 3 with the timeout $\tau_{C4} = [2\Delta + (2n - 3)\Phi]$. Assume that a $k$-good period, $k \geq 1$, starts at time $t_q$ and that round $r_0$ is the highest round started by any process in $\Pi_0$ by time $t_g$. Then every round $4\phi > r_0$ started after time $t_q$ is uniform with non-zero cardinality.

Corollary 3: Parametrization 3 with the timeout $\tau_{C1} = [\Delta + (n + 3)\Phi] + [2\Delta + (2n - 3)\Phi] + \tau_{C3} = [3\Delta + 2n\Phi] + \tau_{C4} = [2\Delta + (2n - 3)\Phi] + \tau_{C4}$ ensures $P_{iv4}(\phi)$, if round $4(\phi - 1)$ starts in a $(\frac{n+1}{2})$-good period.

Proof: By Lemma 8, round $4(\phi - 1)$ is uniform with non-zero cardinality. Thus by the definition of \textit{ElectCoord}, every process in $\Pi_0$ has the same coordinator. Applying Lemma 6 to 8 and solving the equations for $\tau_{C1}$, $\tau_{C2}$, and $\tau_{C4}$ yields the result.  

3. Note that the timeout is different from the one in Lemma 2, since in some cases, processes do not send messages in this round.
8.3 Length of a good period

The next theorem computes the required duration of a good period in order to ensure $y$ consecutive phases of $P_{1;\alpha}(\phi)$. Just like with the predicate $P_{1;\alpha}(\phi)$, the initialization time is given by setting $y = 0$, and the time per phase by the multiplication factor of $y$.

**Theorem 3:** In any good period of length

$$y \left[ (2\Delta + (2n - 3)\Phi) + 4\Delta + (2n + 5)\Phi \right] +$$

$$+ \left( 2\Delta + (2n - 3)\Phi \right) \frac{\beta^2}{\alpha^2} + \left( 5\Delta + (5n - 3)\Phi \right) \frac{\beta}{\alpha} + \Delta + (3n + 1)\Phi$$

the generic algorithm with Parametrization 3 ensures $y$ consecutive phases $\phi$ that fulfill $P_{1;\alpha}(\phi)$.

We first present an informal correctness argument for the length $6y\Delta + 8\Delta$ (we ignore the $\Phi$ terms). Then, we present the complete proof.

**Initialization period.** It can be shown that the “initialization” period is the longest in the following case: (i) $t_g$ starts just after the first send step of the highest round $4(\phi - 2)$ reached by some process $p$, and (ii) round $4(\phi - 2)$ is not uniform (only round $4(\phi - 1)$ is uniform). In this case $p$ will go through the full timeouts of round $4(\phi - 2)$ and $4(\phi - 1) - 3$, which takes $5\Delta$. By this time, say $t_0$, if no other process has started round $4(\phi - 1)$, process $p$ will do so, skipping rounds $4(\phi - 1) - 2$ and $4(\phi - 1) - 1$ (see lines 2 and 3 of $SkipRound$ in Parametrization 3). At latest at $t_0 + \Delta$ the coordinator will start round $4(\phi - 1)$; its round $4(\phi - 1)$ message is ready for reception $\Delta$ later. Round $4(\phi - 1)$ terminates by the expiration of a timeout ($2\Delta$), see lines 2 of $NextRound$ in Parametrization 3, which means that the coordinator message will be received only at $t_0 + 3\Delta$. So latest at $t_0 + 3\Delta$ all processes in $\Pi_0$ have finished round $4(\phi - 1)$, which ends the initialization period and starts a regular phase. Thus, the initialization period lasts for $8\Delta$ and a regular phase starts at $t_g + 8\Delta$.

**Regular phase.** We show that the duration of a regular phase is $6\Delta$. Rename the regular phase to $\phi$. Let the last process enter the regular phase $\phi$ at time $t_r$. Then by $t_r + \Delta$ the coordinator has a majority of round $4\phi - 3$ messages, and $2\Delta$ after the coordinator we receive the round $4\phi - 1$ messages from all processes in $\Pi_0$. By $t_r + 4\Delta$ the coordinator receives the round $4\phi$ and round $4\phi$ is ready for reception at all processes in $\Pi_0$. Round $4\phi$ terminates by the expiration of a timeout ($2\Delta$). So by $t_r + 6\Delta$ all processes have decided.

**Proof:** We first compute the time by which all processes have entered the first round of a good phase, which we will call the initialization period. The duration a phase $\phi$ is measured as the time since the last process enters round $4\phi - 3$ until the last process sends the round $4\phi$.

Assume a good period starts at time $t_g$ (see Fig. 9). We start by computing the latest time a process $p$ will enter a new round $4\phi - 1$ after $t_g$.

After $t_g$, there will be a process $p_1$ that will either start (i) a new round $4\phi - 2$, or (ii) a new round $4\phi - 1 - 2$ before any other process which depends on $p_1$’s round at $t_g$. If $p_1$ is in round two or three of a phase, then the case (i) happens first, the latest by $t_{s4a} = t_g + n\Phi + \tau_{\alpha L} \frac{\beta}{\alpha}$, which is the time required to complete rounds two and three of a phase. If $p_1$ is in round four or one of a phase, then case (ii) happens first. The time by which it happens depends on whether $p_1$ advances to round $4(\phi - 1) - 2$ by timeout or by receiving messages.

- Process $p_1$ advances by timeout (line 3 of $NextRound$). Then $p_1$ will skip rounds $4(\phi - 1) - 2$ and $4(\phi - 1) - 1$. If $p_1$ is a coordinator, then by line 2 of $SkipRound$ it will skip the round without executing send steps. Otherwise, the $Dest$ function ensures that $p_1$ will not send to anyone and line 4 of $NextRound$ makes it advances immediately to round $4(\phi - 1) - 1$. In either case, $p_1$ will skip round $4(\phi - 1) - 1$ by line 3 of $SkipRound$, since no round $4(\phi - 1) - 2$ was sent at this time. Thus, $p_1$ starts round $4(\phi - 1)$ immediately, proposing itself as coordinator because of line 3 of $Dest$. This happens the latest by $t_{s4b} = t_g + (n + 1)\Phi + (\tau_{\alpha L} + \tau L_1) \frac{\beta}{\alpha}$, which is the maximum time required to complete rounds $4(\phi - 2)$ and $4(\phi - 1) - 3$.

- Process $p_1$ advances by receiving messages. This may happen either by receiving a message from round $4(\phi - 1) - 2$ (line 1 of $NextRound$) or by receiving a majority of messages in round $4(\phi - 1) - 3$ (line 3 of $NextRound$). The former could not have happened in this situation, because $p_1$ is the first process to enter round $4(\phi - 1) - 2$, so no round $4(\phi - 1) - 2$ message was sent by this time. In the latter case, $p_1$ may be the coordinator. If it is not, then it will skip round $4(\phi - 1) - 2$ and $4(\phi - 1) - 3$ for the same reasons as if $p_1$ had advanced by timeout. Otherwise, if $p_1$ is the coordinator, let $M_0$ be the set of processes from which $p_1$ received round $4(\phi - 1) - 3$ messages. Processes in $M_0$ must have started round $4(\phi - 2)$ before $t_g$, since by assumption no process started a new round $4(\phi - 1)$ between $t_g$ and $p_1$. Thus, the latest by $t_g + n\Phi + \tau_{\alpha L} \frac{\beta}{\alpha}$ they are in round $4(\phi - 1) - 3$, and $\Delta + \Phi$ time later the message was received by the coordinator. Therefore, the latest by $t_g + \Delta + (n + 2)\Phi + \tau_{\alpha L} \frac{\beta}{\alpha}$ the coordinator has received a majority of messages and $n\Phi + \Delta$ later, the round $4(\phi - 1) - 2$ messages of $p_1$ are ready for reception at all processes. If all alive processes are still in round $4(\phi - 1) - 2$ or lower when the messages arrive, and if they have as coordinator $p_1$ (recall that the phase is not necessarily well coordinated), then they will receive the message, advance to round $4(\phi - 1) - 1$, and reply to $p_1$. Since we are in a good period, the remaining rounds of the phase will complete successfully and the phase will satisfy the predicate. Since this case does not correspond to the longest it takes to satisfy the predicate, we will not consider it. If there is a process $q$ that doesn’t have $p_1$ as coordinator, or that ended round $4(\phi - 1) - 3$ before receiving $p_1$’s message, then by lines 2 and 3 of $SkipRound$, $q$ will advance to round $4(\phi - 1)$ and send a message to all, by line 2 of $ElectCoord$. This
happens the latest at \( t_{4c} = t_g + 2\Delta + (n+2)\Phi + \tau L_{4L} \frac{\beta}{\alpha} \), which is the time by which \( p_1 \)’s messages are received by all processes.

Considering all the cases above, and taking the maximum of \( t_{4a}, t_{4b} \), and \( t_{4c} \) we can conclude that by time \( t_g + (n+2)\Phi + \tau L_{4L} \frac{\beta}{\alpha} \) there is a process \( p \) that started a new round \( 4(\phi-1) \). \((n+1)\Phi + \Delta \) time later, every process has a round \( 4(\phi-1) \) message ready to be received and \( n \) steps later, all processes will have received it and started round \( 4(\phi-1) \). \( \tau L_{4L} \Phi \) later all processes will have entered round \( 4\phi - 3 \), which marks the end of the initialization. This happens by \( t_e = t_g + \Delta + (3n+1)\Phi + (2\tau L_{4L} + \tau L_1) \frac{\beta}{\alpha} \).

We can use Corollary 3 to show that \( P_{lv4}(\phi) \) will be true since round \( 4(\phi-1) \) starts in a good period. Using the timeouts specified in Corollary 3, the algorithm is able to complete \( y \) phases before the end of the good period specified in this Theorem, then by Corollary 3 all those phases will satisfy \( P_{lv4}(\phi) \), which proves this Theorem.

The duration of the good phase can be computed as follows starting from \( t_e \). Rounds \( 4\phi - 3 \) to \( 4\phi - 1 \) are message driven, consisting of two participants-coordinator-participants message exchange, each taking \( \Delta + (n+2)\Phi \). Therefore, at time \( t_e + 4\Delta + (2n + 4)\Phi \) all processes have advanced to round \( 4\phi \), and \( \tau L_{4L} \frac{\beta}{\alpha} \) later have completed round \( 4\phi \). Therefore, the first phase ends at \( t_e + 4\Delta + (2n + 4)\Phi + \tau L_{4L} \frac{\beta}{\alpha} \). By applying the same reasoning, we can show that the following phases will take the same time. Therefore, \( y \) good phases will complete by \( t_e + y[4\Delta + (2n + 4)\Phi + \tau L_{4L} \frac{\beta}{\alpha}] \). By expanding and simplifying this expression, we show that the duration of the good period specified in this theorem is enough to complete \( y \) phases of \( P_{lv4} \).

9 Comparison

In this section we compare quantitatively the different algorithms analyzed previously.

9.1 Impact of clock precision

First, we analyze the impact of the clock on \( \tau_{good} \), the duration of a good period that is sufficient to solve consensus. In order to simplify the comparison, we make the reasonable assumption \( \Phi \ll \Delta \); this allows us to ignore the terms in \( \Phi \). The results are shown in Figure 10. The x-axis corresponds to \( \beta/\alpha \) (see Sect. 4). Larger values of \( \beta/\alpha \) correspond to larger variations in clock skew (worse clock precision); identical clock skew (including perfect clocks for which \( \alpha = \beta = 1 \)) correspond to \( \beta/\alpha = 1 \).

The y-axis corresponds to \( \tau_{good}/\Delta \), which is the duration of a good period expressed using \( \Delta \) as the time unit.

Figure 10 shows that OTR is less sensitive to clock imprecision than the other algorithms. It shows another interesting result, namely that with perfect clocks, the different implementations of the LastVoting algorithm lead to almost the same result. This is no more the case with large clock imprecision (which occurs for instance when clocks are built from step counting and \( \Phi \)) large. We note also that the performance of algorithms that synchronize more often (like LV-3 with full synchronization) degrades less quickly with less precise clocks.

9.2 Analysis of the results for precise clocks

We do now a finer analysis of the different algorithms for the case \( \beta/\alpha = 1 \). We compare algorithms not only in terms of the duration of a good period, but also in terms of the duration of initialization after a good period starts.

9.2.1 Overview

Table 2 is an overview of the results obtained in the paper for all three algorithms (OTR, LV-3 and LV-4). For OTR (predicate \( P_{otr} \)), we have one single option to consider, namely two uniform rounds \( (P_u) \). For LV-3 (predicate \( P_{lv3} \)), we have three options: three uniform rounds (line 2), Phase Sync (line 3) and Piggybacking (line 4). Finally for LV-4 (predicate \( P_{lv4} \)), we have two options: four uniform rounds (line 5) and Coord Sync.
(line 6), which is designed specifically for $P_{1r4}$. For each option, Table 2 shows the initialization time, the time for each consensus after initialization, and the number of messages required for initialization and for each consensus. The time until the first decision after the beginning of a good period can be determined by summing the columns initialization and per consensus.

Line 1 (OTR) follows from the beginning of Section 6 (e.g., Theorem 1 with $z = 0$ for the initialization time, time for two rounds for consensus). Line 2 (LV-3, $3 \times P_u$) follows from Corollary 1. Line 3 (LV-3, Phase Sync) follows from the beginning of Section 7. Line 4 (LV-3, Piggybacking) follows from the expression in Section 7.4 ($y = 0$ for the initialization time, the multiplying factor of $y$ for the duration of one instance of consensus). Line 5 (LV-4, $2 \times P_u$) follows from Corollary 1 and, finally, line 6 (LV-4 Coord Sync) follows from the beginning of Section 8.3.

### 9.2.2 Impact of the round implementation

We see from Table 2 that the performance of our algorithms varies significantly depending on the round implementation. For example, the generic implementations of $P_{1r3}$ and $P_{1r4}$, based on uniform rounds (lines 2 and 5), perform clearly worse than the implementations designed specifically for the corresponding predicates (lines 3, 4 and 6). More precisely, for $\Phi \ll \Delta$, the initialization time of $P_{1r3}$ over uniform rounds is $9\Delta$ and the decision time is $6\Delta$ (line 2), while the best implementation of this predicate (Phase Sync) achieves $8\Delta$ and $4\Delta$ respectively (line 4), while sending a smaller number of messages.

This shows that the round layer implementation plays a crucial role in the performance of round-based algorithms. Specifically, the simplest option (generic uniform rounds) does not lead to the most efficient solution. This is because algorithms like LV-3 and LV-4 do not require uniformity in all rounds. For these two algorithms, the first two rounds of a phase only require sending messages between a coordinator and all processes, which as shown can be implemented more efficiently than generic uniform rounds. Moreover, looking at the rounds of a phase together instead of separately, provides additional cross-round optimization opportunities.

A consequence of using round implementations tailored to the communication predicate is that the number of communication rounds of an algorithm is no longer a good metric of its performance. When using a generic round implementation (like $P_u$), where every round is of the same duration, the performance can be easily estimated as $(\text{round duration}) \times (\# \text{rounds of algorithm})$. But this is no longer the case with optimized round implementations like Phase Sync, Piggybacking or Coord Sync, where the duration of a round differs between predicate implementations and between rounds of the same algorithm. This is confirmed by Table 3, which shows the average round duration ("$\Delta/\text{rounds}$") for each of our algorithms, considering the best round implementation for LV-3 and LV-4. The table shows the results both for short good periods and long good periods. As we have mentioned in the introduction, a period of synchrony is "short" if it allows only a few decisions, in which case the initialization time is important. A period of synchrony is "long" if the number of decisions taken is large enough so that the initialization time is amortized over many consensus instances, and can be ignored. The results show some variation on the average duration for short good periods, ranging from $3.5\Delta$ (OTR and LV-4) to $4\Delta$ (LV-3), and a large variation for long good periods, from $1.3\Delta$ (LV-3) to $2\Delta$ (OTR).

### 9.2.3 Quantitative comparison of OTR, LV-3 and LV-4

Let us go back to Table 2 in order to compare the best implementations of our three consensus algorithms: OTR (only one implementation), LV-3 (Piggybacking) and of LV-4 (Coord Sync). We consider again the (reasonable) assumption $\Phi \ll \Delta$. OTR with $2 \times P_u$ is the algorithm with the shortest initialization time, namely $3\Delta$, compared to $8\Delta$ for LV-3 and LV-4. OTR has also the shortest time until the first decision (7$\Delta$) and the shortest time per consensus after initialization (4$\Delta$). However, this comes at the cost of requiring a greater number of replicas ($3f + 1$ for OTR, compared to $2f + 1$ for LV-3 and LV-4), and of a slightly higher message complexity ($2n^2$ for OTR, $n^2 + 2n$ for LV-3 and $4n$ for LV-4). Among the algorithms that have a resilience of $2f + 1$, LV-3 has the same initialization time as LV-4 (Coord Sync), but a lower per consensus time ($4\Delta$ instead of $6\Delta$). This is to be expected, as LV-3 and LV-4 differ on the last rounds: one all-to-all round for LV-3 vs. two rounds for LV-4 (all-to-coordinator and an

<table>
<thead>
<tr>
<th>Alg.</th>
<th># rounds</th>
<th>Resilience</th>
<th>Pred Impl</th>
<th>length of good period</th>
<th># messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>initialization per consensus</td>
<td>initialization per consensus</td>
</tr>
<tr>
<td>OTR</td>
<td>2</td>
<td>$3f + 1$</td>
<td>$2 \times P_u$</td>
<td>$3\Delta + (4n - 1)\Phi$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>LV-3</td>
<td>3</td>
<td>$2f + 1$</td>
<td>$3 \times P_u$</td>
<td>$9\Delta + (13n - 4)\Phi$</td>
<td>$4n^2$</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;Phase Sync&quot;</td>
<td>$8\Delta + (12n - 1)\Phi$</td>
<td>$2n^2 + 2n$</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;Piggybacking&quot;</td>
<td>$8\Delta + (12n - 1)\Phi$</td>
<td>$2n^2 + 2n$</td>
</tr>
<tr>
<td>LV-4</td>
<td>4</td>
<td>$2f + 1$</td>
<td>$4 \times P_u$</td>
<td>$11\Delta + (12n - 5)\Phi$</td>
<td>$5n^2$</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;Coord Sync&quot;</td>
<td>$8\Delta + (10n - 5)\Phi$</td>
<td>$2n^2 + 3n$</td>
</tr>
</tbody>
</table>

**TABLE 2**
Summary of results for $\beta/\alpha = 1$. 


TABLE 3
Average round duration ($\beta/\alpha = 1$, $\Phi \ll \Delta$).

<table>
<thead>
<tr>
<th>Alg.</th>
<th># rounds</th>
<th>Short good periods</th>
<th>Long good periods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1st decision $\Delta$/rounds</td>
<td>per-consensus $\Delta$/rounds</td>
</tr>
<tr>
<td>OTR</td>
<td>2</td>
<td>$7\Delta$</td>
<td>$4\Delta$</td>
</tr>
<tr>
<td>LV-3 (Piggybacking)</td>
<td>3</td>
<td>$12\Delta$</td>
<td>$4\Delta$</td>
</tr>
<tr>
<td>LV-4 (Coord Sync)</td>
<td>4</td>
<td>$14\Delta$</td>
<td>$6\Delta$</td>
</tr>
</tbody>
</table>

coordinator-to-all). On the other hand, LV-4 (Coord Sync) is the algorithm with the lowest message complexity, with only $4n$ messages per consensus, while all other algorithms have at least one round with $n^2$ messages.

The analysis clearly shows that none of the algorithms is the best choice in every situation. In unstable networks, where the good periods are of short duration, OTR is the best algorithm, as it requires the shortest good period for the first decision. On stable networks, with long good periods, OTR and LV-3 are similar in terms of time per consensus. Since LV-3 is more resilient and has a slightly lower message complexity, it is a better choice. If the number of messages is important and the network is stable, then it is worth to consider LV-4, as it requires only $4n$ messages.

10 CONCLUSION

The paper has derived analytical performance results for several round-based consensus algorithms (OTR, LV-3, LV-4) in a system that alternates between good and bad periods. We have considered different implementation of rounds, and have computed for each algorithms (i) the time from the beginning of a good period until the first decision, and (ii) the time for each additional decision. The results show that the performance of round-based algorithms largely depends on the implementation of rounds. The results also show the number of rounds of an algorithm is not always a good metric for the performance of an algorithm. Finally we can observe tradeoffs in resilience, minimum duration of a good period, decision time in the case of long good periods, and message complexity.

As the next step, we plan to extend the analysis to non round-based consensus algorithms, in order to understand the performance trade-offs between round-based and non round-based consensus algorithms, and to understand whether the overall conclusions would be similar or very different.
Fig. 11. Synchronization by a coordinator: Lemmas 6–8.

APPENDIX

PROOFS FOR SECTION 8.2

Lemma 6 (Timeout $\tau_{C1}$): Consider Parametrization 3 with $\tau_{C1} = \lfloor 3\Delta + 3n\Phi \rfloor \beta + \tau_{C4} \left( \frac{\beta}{2} - 1 \right)$. Assume every process starts round $4(\phi - 1)$ in a $\left( \frac{\beta + 1}{2} \right)$-good period and phase $\phi$ has a unique coordinator $c$. Then (i) $c$ hears from a majority of processes in round $4\phi - 3$, and (ii) all processes in $\Pi_0$ hear from $c$ in round $4\phi - 2$.

Proof: We start with (i), and compute first the latest time at which all round $4\phi - 3$ messages are ready to be received. Assume $p_1$ is the first process to execute a send step for round $4(\phi - 1)$ at $t_{s1}$. In the following, we use $p_2$ to represent the process that takes the longest to send a round $4\phi - 3$ message. By $t_{s4} + (n - 2)\Phi$, $p_1$ finishes sending its round $4(\phi - 1)$ messages. $\Delta + n\Phi$ time later, $p_2$ receives the message from $p_1$ and starts the timeout $\tau_{C4}$. This timeout expires at most $\tau_{U4} + \Phi$ later, which means that $p_2$ starts round $4\phi - 3$ by $t_{s4} + \Delta + (2n - 1)\Phi + \tau_{U4}$. Then, at most $\Phi$ time later, $p_2$ has sent its message to the coordinator ($p_1$ in Fig. 11), and $\Delta$ time later this message is ready for reception. Thus, by time $t_0 = t_{s4} + 2\Delta + 2n\Phi + \tau_{U4}$, the messages from all processes in $\Pi_0$ are ready to be received by the coordinator. The coordinator ($p_1$ in Fig. 11) enters round $4\phi - 3$ not before $t_{s4} + \tau_{L4}$, thus its timeout for round $4\phi - 3$ will not expire before $t_{s4} + \tau_{L4} + \tau_{L1}$, which is after $t_0$. In addition, since we are in a $\left( \frac{\beta + 1}{2} \right)$-good period, there is at least a majority of processes in $\Pi_0$ that has sent a round $4\phi - 3$ message; so the coordinator receives a majority of messages. This proves (i).

To show (ii), recall that by $t_0$ all round $4\phi - 3$ messages are ready to be received by the coordinator $p_1$. One receive step later the coordinator advances to round $4\phi - 2$, and $n - 1$ send steps later it sends all its round $4\phi - 2$ messages. The message is ready for reception at all processes ($p_3$ in Fig. 11) by $t_{s1} = t_0 + n\Phi + \Delta = t_{s4} + 3\Delta + 3n\Phi + \tau_{U4}$. We now show that no timeout for round $4\phi - 3$ expires before $t_{s1}$. Since timeouts are started after sending, by definition, $t_{s4}$ is also the earliest time a process $p_3$ in $\Pi_0$ starts the timeout for round $4(\phi - 1)$. By Lemma 1, $p_3$ then waits at least until $t_{s1} = t_{s4} + \tau_{L4}$ before sending its round $4\phi - 3$ message to the coordinator and start the timeout $\tau_{C1}$. The timeout for round $4\phi - 3$ expires at $p_3$ the earliest at $t_{s4} + \tau_{L4} + \tau_{L1}$, which is equal to $t_{s1}$. So all processes in $\Pi_0$ hear from the coordinator in round $4\phi - 2$, which completes the proof of (ii).

Lemma 7 (Timeout $\tau_{C2}$): Consider Parametrization 3 with the timeout $\tau_{C2} = \lfloor 3\Delta + 2n\Phi \rfloor \beta$. Assume every process starts round $4(\phi - 1)$ in a $\left( \frac{\beta + 1}{2} \right)$-good period, and $\phi$ has a unique coordinator $c$. Then (i) $c$ hears from a majority of processes in round $4\phi - 1$, and (ii) all processes in $\Pi_0$ hear from $c$ in round $4\phi$.

Proof: Since round $4(\phi - 1)$ started in a $\left( \frac{\beta + 1}{2} \right)$-good period, by Lemma 6, the coordinator receives all the round $4\phi - 3$ messages, and sends its round $4\phi - 2$ messages.

We start with (i), and compute first the earliest time by which all round $4\phi - 1$ messages are ready to be received by the coordinator. Let’s assume that the coordinator sends its first round $4\phi - 2$ message at time $t_{s3}$ to a process $p_2$. The coordinator finishes sending its round $4\phi - 2$ messages at latest by $t_{s3} + (n - 2)\Phi; \Delta + 2\Phi$ later all processes have received this message, advanced to round $4\phi - 1$ and sent a round $4\phi - 1$ message back to the coordinator. By time $t_0 = t_{s3} + n\Phi + 2\Delta$ these message are ready for reception by the coordinator. The coordinator starts its timeout for round $4\phi - 1$ not before $t_{s3}$, i.e., the timeout expires not before $t_{s3} + \tau_{L3}$, which is after $t_0$. Thus, the coordinator receives a majority of round $4\phi - 1$ messages, which proves (i).

We prove now (ii), and start our considerations at time $t_0$ (see above) when the coordinator is ready to receive a majority of round $4\phi - 1$ messages. One receive step later the coordinator advances to round $4\phi$, and $n - 1$ steps later it finishes sending its round $4\phi$ messages. This message is ready for reception at any process by time $t_{s3} = t_{s3} + 3\Delta + 2n\Phi$. We show now that no timeout for round $4\phi$ expires before $t_{s3}$. The earliest $p_2$ can receive the round $4\phi - 2$ message from the coordinator $p_1$ is at time $t_{s3}$. Upon receiving the coordinator’s message, $p_2$ enters round $4\phi - 2$ (line 1 of $NextRound$), advances to round $4\phi - 1$ (line 4 of $NextRound$), and sends a message to the coordinator. So the earliest time at which $p_2$ may start round $4\phi - 1$ is $t_{s3}$, and the earliest time that any process in $\Pi_0$ expires the timeout for round $4\phi - 1$ is $t_{s3} + \tau_{L3} = t_{s3} + 3\Delta + 2n\Phi$. This time is not before $t_{s3}$, which proves (ii).

Lemma 8 (Timeout $\tau_{C4}$): Consider Parametrization 3 with the timeout $\tau_{C4} = \lfloor 2\Delta + (2n - 3)\Phi \rfloor \beta$. Assume that a $k$-good period, $k \geq 1$, starts at time $t_g$ and that round $r_0$ is the highest round started by any process in $\Pi_0$ by time $t_g$. Then every round $4\phi > r_0$ started after time $t_g$ is uniform with non-zero cardinality.

Proof: We show that every process in $\Pi_0$ receives a message from all processes in a non-empty subset of $\Pi_0$ and not from any other process.

By the same argument as in the proof of Lemma 2, item 4. Note that our reasoning is different here to the one in the proof of Lemma 2, where we start our considerations with the start of the timeout, while here we start with the first message that is sent.

5. Note that the timeout is different from the one in Lemma 2, since in some cases, processes do not send messages in this round.
(ii), we can conclude that in round $4\phi > r_0$ no process in \(\Pi_0\) receives a message from processes in \(\Pi \setminus \Pi_0\).

We consider now messages received from processes in \(\Pi_0\). Consider Fig. 11 while renaming \(4(\phi - 1)\) to \(4\phi\). Let \(t_{s_4}\) be the earliest time some process \(p_i \in \Pi_0\) sends a round \(4\phi > r_0\) message. Since timeouts are started after sending, by definition, \(t_{s_4}\) is also the earliest that a process can start the timeout for this round. The timeout \(t_{C_4} = (2\Delta + (2n - 3)\Phi)\beta\), together with Lemma 1, ensures that no timeout of length \(t_{C_4}\) started at time \(t_{s_4}\) expires before \(t_{s_1} = t_{s_4} + 2\Delta + (2n - 3)\Phi\).

We will now show that all round \(4\phi\) messages are received before \(t_{s_1}\). Process \(p_1\) finishes his send steps by time \(t_{s_4} + (n - 2)\Phi\). By time \(t_{s_4} + (n - 2)\Phi + \Delta\) the messages sent by \(p_1\) are ready to be received by all other processes. This is the latest time some process, say \(p_2\), may start the send steps for round \(4\phi\), since if \(p_2\) receives the round \(4\phi\) message while on a lower round, it will skip the send steps of this round (line 3 of \(Dest\)). Assume \(p_2\) started its send steps at \(t_{s_4} + \Delta + (n - 2)\Phi\). Its messages will be ready for reception at all processes \((n - 1)\Phi + \Delta\) later, i.e., by \(t_{s_4} + 2\Delta + (2n - 3)\Phi\) which, as seen previously, is before any timeout expires. So when the timeout expires, all messages sent in round \(4\phi\) are either received or ready to be received; thus in round \(4\phi\) every process in \(\Pi_0\) receives all messages sent in round \(4\phi\), hence the round is uniform.

Trivially, at least one message is sent in round \(4\phi\), which shows the non-empty cardinality.

\[\square\]

**References**


