Quantitative Analysis of Consensus Algorithms

Fatemeh Borran, Martin Hutle, Nuno Santos, and André Schiper, *Member*, *IEEE*

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 algorithms using communication predicates, on top of a partial synchronous system that alternates between good and bad periods (synchronous and nonsynchronous periods). Communication predicates together with the detailed timing information of the underlying partially 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, such as the number of rounds of an algorithm is not necessarily a good metric for its performance.

Index Terms—Distributed systems, fault tolerance, distributed algorithms, round-based model, consensus, system modeling.

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 Π, 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 may initially be 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. The paper considers a partially synchronous system that alternates between periods of synchrony and periods of asynchrony (which includes the original definition), and 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 nonround-based algorithms, for instance, to consensus protocols driven by message reception like Paxos [12]. Section 9 discusses this issue in more detail.

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, also called good period, that allows the algorithm to solve x instances of consensus as $initialization + x \cdot per\text{-}consensus$. In a good period, the $initialization$ time is a one time duration that allows processes to synchronize to a specific round of the consensus algorithm; while $per\text{-}consensus$ 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 detailed performance analysis done in the paper. Our results also allow us 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 can become unacceptable for algorithms that resynchronize less often.

The paper is structured as follows: Section 2 discusses related work. In Section 3, we recall the concept of

- *F. Borran, N. Santos, and A. Schiper are with the École Polytechnique Fédérale de Lausanne, LSR, Station 14, Lausanne 1015, Switzerland. E-mail: {fatemeh.borran, nuno.santos, andre.schiper}@epfl.ch.*
- *M. Hutle is with Fraunhofer AISEC, Parkring 4, Garching 85748, Munich, Germany. E-mail: martin.hutle@aisec.fraunhofer.de.*

Manuscript received 4 Feb. 2010; revised 1 July 2011; accepted 21 Aug. 2011; published online 30 Sept. 2011.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2010-02-0025. Digital Object Identifier no. 10.1109/TDSC.2011.48.

HO machines and communication predicates, together with the corresponding algorithms, which form the basis for our analysis. Implementation of communication predicates is the topic of the subsequent sections. After defining our system model for our implementations in Section 4, we give an abstract algorithm that may serve as a generic implementation for many predicates, and which is used by all our implementations. After that, in Sections 6 to 8, we describe how our predicates can be implemented using different strategies. Different strategies lead to different lengths of the good period that is necessary to guarantee the predicate, and to different numbers of messages. We finally analyze our results in Section 9 and conclude the paper in Section 10.

2 Related Work

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 Global Stabilization Time (GST), i.e., the time at which the system becomes synchronous, for all correct processes to decide. However, the algorithms in [8] and ours are different.[1] Moreover, rounds in [8] are implemented on top of a clock synchronization algorithm, and while the algorithms are polynomial in the constants n, Δ, and Φ, 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 did not receive 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 nonsynchronous systems is [16]. The paper considers failure detectors, and uses as metric the minimum number of communication steps for deciding in a "nice" run, i.e., a run with no crashes and no false suspicions.

Later, Dutta et al. [6], Keidar and Shraer [11], and Alistarh et al. [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 17δ), where δ is the upper bound on message delivery time after stability is reached (δ 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).

1. The algorithms in [8] were mostly ignored by the research community, which instead has focused on Paxos-like algorithms [12].

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 (i.e., handles faults, being static or dynamic, permanent or transient, in a unified way). 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_p^r and a transition function T_p^r. Let s_p denote the current state of process p. For each round r and each p, the sending function $S_p^r(s_p)$ 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 $T_p^r(\vec{\mu}, s_p)$, where $\vec{\mu}$ 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 $HO(p,r)$ the set of processes (including itself) from which p receives a message at round r: $HO(p,r)$ is the *heard of* set of p in round r. If $q \notin HO(p,r)$, then the message sent by q to p in round r was subject to a transmission failure (this includes the case where q does not send a message in round r because it has crashed). Communication predicates are expressed over the sets $(HO(p,r))_{p\in\Pi, r>0}$. Communication predicates restrict transmission failures, for example, the predicate $\forall p, \forall r : |HO(p,r)| > n/2$ ensures that every process receives at least $n/2$ messages in every round.

A tuple $\mathcal{A} = \langle S_p^r, T_p^r \rangle$ is called an HO algorithm. Let $\mathcal{P}$ be a communication predicate. A tuple $\langle \mathcal{A}, \mathcal{P} \rangle$ is called an HO machine, and can be used to solve a distributed problem over the set of processes. 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 (CHO) machine 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 S_p^r and T_p^r 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 $Coord(p, \phi)$ 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) despite benign (non-Byzantine) faults, 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 $4\phi - 3$ to 4ϕ, where ϕ denotes the current phase. Roughly speaking, round $4\phi - 3$ corresponds to phase $1b$ of Paxos, round $4\phi - 2$ to phase $2a$, and round $4\phi - 1$ to phase $2b$. Phase $1a$ of Paxos is hidden in the implementation of round 4ϕ, for leader election. The termination property of consensus is guaranteed by the existence of a phase ϕ such that following predicate holds:

$$\begin{aligned}\mathcal{P}_{lv4}(\phi) :: \exists \Pi_0 \subseteq \Pi \; s.t. \; |\Pi_0| > n/2, \exists c \in \Pi, \forall p \in \Pi_0 : \\ Coord(p, \phi) = c \wedge \\ |HO(c, 4\phi - 3)| > n/2 \; \wedge \; c \in HO(p, 4\phi - 2) \wedge \\ \Pi_0 \subseteq \; HO(c, 4\phi - 1) \; \wedge \; c \in HO(p, 4\phi),\end{aligned}$$

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.

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

Initialization:
$x_p := v_p \in V$ /* v_p is the initial value of p */
$vote_p \in V \cup \{?\}$, initially ?
$commit_p$ a Boolean, initially `false`
$ready_p$ a Boolean, initially `false`
$ts_p \in \mathbb{N}$, initially 0

Round $r = 4\phi - 3$:
S_p^r:
  send $\langle x_p, ts_p \rangle$ to $Coord(p, \phi)$

T_p^r:
  if $p = Coord(p, \phi)$ **and** number of $\langle \nu, \theta \rangle$ received $> n/2$ **then**
    let $\overline{\theta}$ be the largest θ from $\langle -, \theta \rangle$ received
    $vote_p :=$ one $\overline{x}$ such that $\langle \overline{x}, \overline{\theta} \rangle$ is received
    $commit_p :=$ `true`

Round $r = 4\phi - 2$:
S_p^r:
  if $p = Coord(p, \phi)$ **and** $commit_p$ **then**
    send $\langle vote_p \rangle$ to all processes
T_p^r:
  if received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
    $x_p := v$
    $ts_p := \phi$

Round $r = 4\phi - 1$:
S_p^r:
  if $ts_p = \phi$ **then**
    send $\langle ack \rangle$ to $Coord(p, \phi)$

T_p^r:
  if $p = Coord(p, \phi)$ **and** number of $\langle ack \rangle$ received $> n/2$ **then**
    $ready_p :=$ `true`

Round $r = 4\phi$:
S_p^r:
  if $p = Coord(p, \phi)$ **and** $ready_p$ **then**
    send $\langle vote_p \rangle$ to all processes

T_p^r:
  if received $\langle v \rangle$ from $Coord(p, \phi)$ **then**
    DECIDE(v)
  $commit_p :=$ `false`
  $ready_p :=$ `false`

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:

$$\begin{aligned}\mathcal{P}_{lv3}(\phi) :: \exists \Pi_0 \subseteq \Pi \; s.t. \; |\Pi_0| > n/2, \exists c \in \Pi, \forall p \in \Pi_0 : \\ Coord(p, \phi) = c \; \wedge \; |\, HO(c, 3\phi - 2)| > n/2 \wedge \\ c \in HO(p, 3\phi - 1) \; \wedge \; \Pi_0 \subseteq \; HO(p, 3\phi).\end{aligned}$$

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

Initialization:
$x_p := v_p \in V$ /* v_p is the initial value of p */
$vote_p \in V \cup \{?\}$, initially ?
$commit_p$ a Boolean, initially `false`
$ts_p \in \mathbb{N}$, initially 0

Round $3\phi - 2$: identical to round $4\phi - 3$ of Algorithm 1.

Round $3\phi - 1$: identical to round $4\phi - 2$ of Algorithm 1.

Round $r = 3\phi$
S_p^r:
  if $ts_p = \phi$ **then**
    send $\langle ack, x_p \rangle$ to all processes

T_p^r:
  if $\exists v$ such that number of $\langle ack, v \rangle$ received $> n/2$ **then**
    DECIDE(v)
  $commit_p :=$ `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.

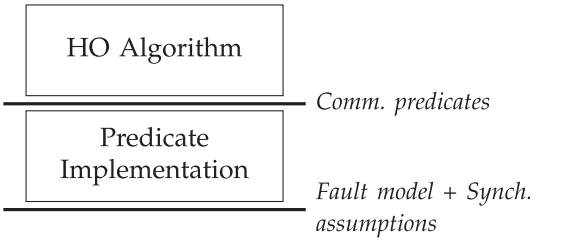

Fig. 1. The two layers when implementing predicates.

This algorithm, called OneThirdRule, 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; otherwise, decision can be reached in two rounds. For liveness, two distinct rounds (not necessarily consecutive) that satisfy the following predicate are needed:

$$\mathcal{P}_u(r) :: \exists \Pi_0 \subseteq \Pi \; s.t. \; |\Pi_0| > 2n/3, \forall p \in \Pi_0 : \\ HO(p,r) = \Pi_0.$$

Informally, this predicate ensures that a "large enough" set of processes all receive the same set of messages. Such a round is called *uniform* with cardinality $2n/3$; we use the term *uniform round* when the cardinality is clear from the context. The predicate $\mathcal{P}_{otr}$ will denote the existence of two distinct rounds (not necessarily consecutive) that satisfy $\mathcal{P}_u()$.

Algorithm 3. The *OneThirdRule* algorithm [5].

1: **Initialization:**
2: $x_p \leftarrow v_p$ /*v_p is the initial value of p*/
3: **Round** r:
4: S_p^r:
5: send $\langle x_p \rangle$ to all processes
6: T_p^r:
7: **if** $|HO(p,r)| > 2n/3$ **then**
8: $x_p :=$ the (smallest) most frequently received value
9: **if** more than $2n/3$ values received are equal to $\overline{x}$ **then**
10: DECIDE($\overline{x}$)

Remark. The algorithms and predicates given in this section ensure a decision of a majority of processes (two-thirds majority in the case of OTR). However, with a small modification, all processes that are eventually reachable will decide: since our agreement property is a uniform property (there are no "faulty" processes that are exempted from agreement), processes—once they have decided—can simply communicate their decision to all other processes. Once this communication is successful, also these processes will decide.

3.3 Implementation of Predicates

In this paper, we are interested in the question of how an HO machine $\langle \mathcal{A}, \mathcal{P} \rangle$, 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_p^r and transition function T_p^r, 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).

TABLE 1
Algorithms and Predicate Implementations Studied

Alg.	Rounds	Resilience	Predicate Implementations
OTR	2	$3f+1$	$2 \times \mathcal{P}_u$
LV-3	3	$2f+1$	$3 \times \mathcal{P}_u$, Phase Sync, Piggybacking
LV-4	4	$2f+1$	$4 \times \mathcal{P}_u$, Coord Sync

Given an implementation for some predicate $\mathcal{P}$, we are looking for the *length of a good period*, i.e., the duration the 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 $buffer_p$, 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 $send(\langle m \rangle, q)$, the tuple $\langle m, p \rangle$ is placed in $buffer_q$.
- In a *receive step*, some messages are received, that is, when process p executes $receive(S)$, a set $S \subseteq buffer_p$ is removed from $buffer_p$, 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 nondecreasing at any time. Real time and the local clock take values from $\mathbb{R}$.

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 $t + \Delta$.*

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 $t \in I$ is a Δ-timely message, provided that $t + \Delta \in I$.

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). *A process is said to be Φ-synchronous in interval I if there is a bound Φ such that in any subinterval of I 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 $0 < \alpha \leq \beta$, so that for any two times t_1, $t_2 \in I$ s.t. $0 < t_1 < t_2$:*

$$\frac{C_p(t_2) - C_p(t_1)}{t_2 - t_1} \in [\alpha, \beta]. \quad (1)$$

Note that our clock definition is very general, since it encompasses other definitions like the classical bounded-drift clocks ($\alpha = 1 - \rho$, $\beta = 1 + \rho$), whereas the values $\alpha = 1/\Phi$, $\beta = 1$ 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 $\Pi_0 \subseteq \Pi$ be a set of processes. An interval I is a* good period for Π_0, *if there are* a priori *known bounds $\Phi, \Delta \in \mathbb{N}$, and $\alpha, \beta \in \mathbb{R}$, with $\Phi > 0$, $0 < \alpha \leq \beta$, such that*

1. *in I, all processes in Π_0 are Φ-synchronous and have a local (α, β) bounded-drift clock in I,*
2. *no process that is not in Π_0 makes a step,*
3. *all links between processes in Π_0 are Δ-timely, and*
4. *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 $|\Pi_0| \geq k$. 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 envelope. 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 $\tau_C = \beta\tau_L$, measured by some process p, corresponds to a real-time interval of length in $[\tau_L, \tau_U]$, with $\tau_U = \frac{\beta}{\alpha}\tau_L$.*

Proof. Let $[t_1, t_2]$ be a real-time interval. Then, $C_p(t_2) - C_p(t_1) = \tau_C$ is the duration of the interval as measured by p, and $t_2 - t_1$ the real-time duration. From (1), we have $t_2 - t_1 \geq [C_p(t_2) - C_p(t_1)]/\beta = \tau_L$, and $t_2 - t_1 \leq [C_p(t_2) - C_p(t_1)]/\alpha = \frac{\beta}{\alpha}\tau_L$, 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, thereby allowing the implementation of all predicates considered in this paper. *Phase Synchronization* (Section 7) is an optimized implementation for $\mathcal{P}_{lv3}$, 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 $\mathcal{P}_{lv4}$.

The generic Algorithm 4 follows the subsequent 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 to which a message is sent 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_p receives a message from some round $r' > r_p$. 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.

Algorithm 4. Generic algorithm of the predicate layer

```
 1: Rcv_p ← ∅                                   /* set of messages received */
 2: r_p ← 1                                     /* round number */
 3: s_p ← init_p                                /* state of the process p */
 4: coord_p ← ⊥                                 /* coordinator of process p */
 5: t_p ← C_p()                                 /* timer */
 6: while true do
 7:     coord_p ← ElectCoord(p, r_p, C_p() − t_p, coord_p, Rcv_p)
 8:     if ¬SkipRound(p, r_p, C_p() − t_p, coord_p, Rcv_p) then
 9:         msgs ← S_p^{r_p}(s_p, coord_p)
10:         for all q ∈ Dest(p, r_p, C_p() − t_p, coord_p, Rcv_p) do
11:             if p = q then
12:                 Rcv_p ← Rcv_p ∪ {⟨msgs[p], p, r_p⟩}
                                               /* local delivery */
```

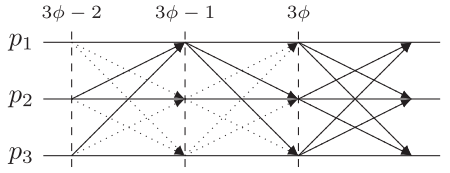

Fig. 2. Message pattern of full synchronization.

13: **else**
14: send($\langle msgs[q], r_p\rangle, q$)
15: $t_p \leftarrow C_p()$
16: **while** $\neg NextRound(p, r_p, C_p() - t_p, coord_p, Rcv_p)$ **do**
17: receive(S)
18: **for all** messages $\langle\langle x, r\rangle, q\rangle \in S$ **do**
19: $Rcv_p \leftarrow Rcv_p \cup \{\langle x, q, r\rangle\}$
20: $s_p \leftarrow T_p^{r_p}(\{\langle x, q\rangle \mid \langle x, q, r_p\rangle \in Rcv_p\}, s_p, coord_p)$
21: $r_p \leftarrow r_p + 1$

6 Full Synchronization

Our first implementation is given as Parameterization 1 for the generic algorithm. The implementation ensures uniform rounds in a good period, which "almost" ensures $\mathcal{P}_{lv4}$ (for LV-4) and $\mathcal{P}_{lv3}$ (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 HO sets of round r). Thus, uniformity of round $4\phi_0$ and of the four rounds of phase ϕ_0+1 allows us to ensure $\mathcal{P}_{lv4}$, and uniformity of round $3\phi_0$ and of the three rounds of phase ϕ_0+1 allows us to ensure $\mathcal{P}_{lv3}$. More generally, $2y$ consecutive uniform rounds ensure y instances of $\mathcal{P}_{otr}$, in the worst case $4y+4$ consecutive uniform rounds ensure y instances of $\mathcal{P}_{lv4}$, and $3y+3$ consecutive uniform rounds ensure y instances of $\mathcal{P}_{lv3}$.

With Parameterization 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 $\mathcal{P}_{otr}$, where every process must hear from every alive process, this induces some overhead for $\mathcal{P}_{lv3}$ and $\mathcal{P}_{lv4}$, since these predicates only require one-to-all or all-to-one patterns on some of their rounds. This is shown in Fig. 2 for predicate $\mathcal{P}_{lv3}$, where the full lines represent messages required by the predicate and dotted lines represent the additional messages sent by Parameterization 1.

In Sections 7 and 8, we provide implementations for $\mathcal{P}_{lv3}$ and $\mathcal{P}_{lv4}$ with lower message complexity. The reminder of the section describes *Full Synchronization* in more detail, computes the timeout τ_C, and the length of a good period.

Parameterization 1. A generic parameterization using full synchronization; where $ho(r) := \{q \mid \langle -, q, r\rangle \in Rcv\}$

$$NextRound(p, r, \tau, coord, Rcv) := \bigvee \begin{cases} \exists \langle -, -, r'\rangle \in Rcv : \ r' > r \\ \tau \geq \tau_C \end{cases}$$

$SkipRound(p, r, \tau, coord, Rcv) := \exists \langle -, -, r'\rangle \in Rcv : \ r' > r$
$Dest(p, r, \tau, coord, Rcv) := \Pi$
$ElectCoord(p, r, \tau, coord, Rcv) :=$

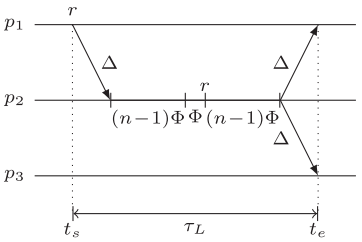

Fig. 3. Full synchronization: Lemma 2.

$$\begin{cases} \min(\Pi) & : \quad r = 1 \\ \min(ho(r-1)) & : \quad ho(r-1) \neq \emptyset \\ coord & : \quad \text{else} \end{cases}$$

6.1 Outline of Full Synchronization

As shown by function $NextRound$ in Parameterization 1, there are two ways for a process p to leave round r: 1) by receiving a message from a higher round $r' > r$, or 2) by expiration of a timeout. In both cases, there is at least one process whose timeout for round r expires; this makes the protocol driven by timeout. In case 1, 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 2, the timeout τ_C is chosen to ensure uniformity, i.e., $\mathcal{P}_u()$, in a good period (see Lemma 2 below). As shown by the function $ElectCoord$, the coordinator for some round r is the smallest process ($\min$) in the HO set of round $r-1$ (whenever this HO set is nonempty). Note the definition of the macro $ho(r)$ given in the caption of Parameterization 1; we will also use this notation in the following sections. This ensures a unique coordinator in good periods where rounds are uniform. For noncoordinated predicates, like $\mathcal{P}_u()$, no coordinator is needed and the function $ElectCoord$ can be ignored.

6.2 Timeout τ_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 $\mathcal{P}_u()$ for processes that are up in this good period. In the next section, we compute the length of a good period that is sufficient to ensure $\mathcal{P}_u()$.

Lemma 2 (Timeout τ_C). *Consider Parameterization 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 Π_0 by time t_g. Then, every new round $r > r_0$ started after time t_g is uniform ($\mathcal{P}_u(r)$) with cardinality k.*

Proof. We show that in round r, for every process $p \in \Pi_0$, we have: 1) p receives a message from all processes in Π_0, but 2) not from any process not in Π_0.

We start with point 2. Assume that p received a round r message from a process q that is not in Π_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 Π_0 has entered round r by t_g.

To prove point 1, 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

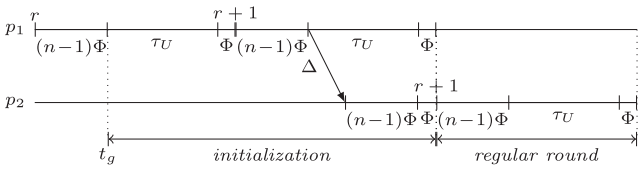

Fig. 4. Full synchronization: Theorem 1.

each process in Π_0 (p_2 in Fig. 3), the latest at $t_s + \Delta$, since messages are Δ-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 Φ time, p_1's message is received by all processes in Π_0 the latest at $t_s + \Delta + n\Phi$. Each process that receives this message jumps to round r, if not already 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 τ_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 Π_0 receives a message from every process in Π_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 $\mathcal{P}_{otr}$ 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 Parameterization 1 ensures x consecutive rounds that fulfill $\mathcal{P}_u()$.

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 Π_0. We distinguish two cases: 1) t_g is during these $n-1$ send steps (not shown in Fig. 4) of round $r = 3\phi$. 2) t_g after these send steps (see Fig. 4). It can be shown that case 2 is worse than case 1 in terms of length of the good period; thus, we consider case 2. Round $r+1$ is the first round that all processes in Π_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_1, no message of a round larger than r is received before p_1's timer expires, and $\tau_U = \frac{\beta}{\alpha}\tau_L$ is the time elapsed for a timeout $\tau_C = \tau_L\beta$; when the timeout expires, p_1 executes a receive step (ϕ), 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; Δ 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 (Φ) in which p_1's message is finally received; at this point, p_2 moves to round $r+1$.

We now show that case 1 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 2, 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 $t_g + \tau_U + \Delta + 2n\Phi$.

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 τ_U (in the worst case, no message of a larger round is received), one receive step (Φ).

Summing up the duration of "initialization" and of x "regular rounds" leads to $(x+1)[\tau_U + n\Phi] + \Delta + n\Phi$. Replacing τ_U with $\frac{\beta}{\alpha}\tau_L$, and τ_L 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 $\mathcal{P}_{otr}$, we need $2y$ uniform rounds, for y instances of $\mathcal{P}_{lv3}$, we need $3y+3$ uniform rounds, and for y instances of $\mathcal{P}_{lv4}$, 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 $\vartheta = \frac{\beta}{\alpha}(2\Delta + (2n-1)\Phi) + n\Phi$. The initialization time of LV-3 Full Sync, resp. LV-4 Full Sync, is $4\vartheta + \Delta + n\Phi$, resp. $5\vartheta + \Delta + n\Phi$. After initialization, the duration of one instance of LV-3 Full Sync, resp. LV-4 Full Sync, is 3ϑ, resp. 4ϑ.*

7 Phase Synchronization

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

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

7.1 Outline of Phase Synchronization

The "natural" message pattern depicted in Fig. 5 is generated by the function *Dest* in Parameterization 2.

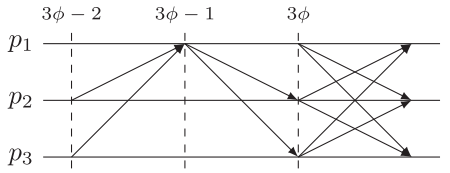

Fig. 5. Message pattern of phase synchronization.

Parameterization 2. LV-3 using phase synchronization; where $ho(r) := \{q \mid \langle -, q, r\rangle \in Rcv\}$

$NextRound(p, r, \tau, coord, Rcv) :=$

$$\bigvee \begin{cases} \exists \langle -, -, r'\rangle \in Rcv:\ r' > r \\ r \bmod 3 = 1 \wedge (\tau \geq \tau_{C1} \vee |ho(r)| > n/2) \\ r \bmod 3 = 2 \wedge \tau \geq \tau_{C2} \\ r \bmod 3 = 0 \wedge \tau \geq \tau_{C3} \end{cases}$$

$SkipRound(p, r, \tau, coord, Rcv) := \exists \langle -, -, r'\rangle \in Rcv:\ r' > r$

$Dest(p, r, \tau, coord, Rcv) :=$

$$\begin{cases} coord & : & r \bmod 3 = 1 \\ \Pi & : & r \bmod 3 = 0 \vee (r \bmod 3 = 2 \wedge p = coord) \\ \emptyset & : & \text{else} \end{cases}$$

$ElectCoord(p, r, \tau, coord, Rcv) :=$

$$\begin{cases} \min(\Pi) & : & r = 1 \\ \min(ho(r-1)) & : & r \bmod 3 = 1 \wedge ho(r-1) \neq \emptyset \\ coord & : & \text{else} \end{cases}$$

Round 3ϕ of phase ϕ 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 3ϕ 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 3ϕ 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 3ϕ, and sends its round 3ϕ 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 3ϕ, 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 3ϕ 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.

7.2 Timeouts τ_{C1}, τ_{C2}, τ_{C3}

We now compute values for τ_{C1}, τ_{C2}, and τ_{C3} that ensure $\mathcal{P}_{lh3}$ during a sufficiently long good period; the required length of a good period is proven in Section 7.3.

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

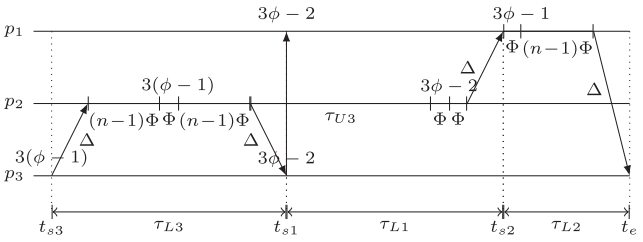

Fig. 6. Phase synchronization: Lemmas 3-5.

Proof. Let p_3 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 $t_{s3} + \Delta + (2n-1)\Phi + \tau_{L3}\frac{\beta}{\alpha}$. 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 $t_{s3} + \Delta + (2n+1)\Phi + \tau_{L3}\frac{\beta}{\alpha}$. This message is ready for reception at the coordinator Δ time later. Thus, if the coordinator executes the receive step and the transition function for round $3\phi - 2$ not before time $t_{s2} = t_{s3} + 2\Delta + (2n+1)\Phi + \tau_{L3}\frac{\beta}{\alpha}$, 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 τ_{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 τ_{C1} as given by the lemma ensures that $t_{s1} + \tau_{L1}$ is not before ts_2. Thus, the coordinator p_1 receives the round $3\phi - 2$ messages from all processes in Π_0. □

Lemma 4 (Timeout τ_{C2}). *Consider Parameterization 2 with the timeout $\tau_{C2} = [3\Delta + (3n+1)\Phi]\beta + \tau_{C3}(\frac{\beta}{\alpha} - 1) - \tau_{C1}$. Assume a phase ϕ 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 Π_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 $t_{s3} + \Delta + (2n+1)\Phi + \tau_{L3}\frac{\beta}{\alpha}$. Then, at most $\Delta + \Phi$ later, the coordinator has received this message from every process in Π_0, thus achieved the majority condition in line 2 of $NextRound$, and therefore sends its round $3\phi - 1$ message to all. The latter takes at most $(n-1)\Phi$ time, and this message will be ready for reception Δ time later. At this time, $t_e = t_{s3} + 3\Delta + (3n+1)\Phi + \tau_{L3}\frac{\beta}{\alpha}$, the timeout τ_{C2} may safely expire at any process. Because p_3 is the first process that starts round $3(\phi-1)$, no process starts round $3\phi - 1$ before $t_{s2} = t_{s3} + \tau_{L3} + \tau_{L1}$. Thus, choosing τ_{C2} as in the lemma ensures that τ_{C2} does not expire before time t_e at any process. □

Lemma 5 (Timeout τ_{C3}). *Consider Parameterization 2 with the timeout $\tau_{C3} = [2\Delta + (2n-1)\Phi]\beta$. Assume that a k-good period starts at time t_g and that round r_0 is the highest round started by any process in Π_0 by time t_g. Then, every new round $3\phi > r_0$ started after time t_g is uniform with cardinality k.*

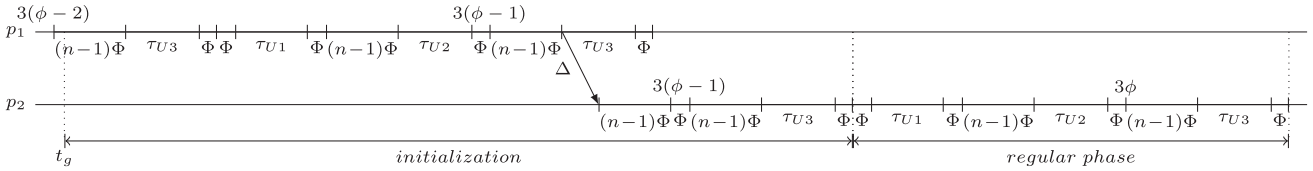

Fig. 7. LV-3 using phase synchronization: Theorem 2.

Proof. Similar to Lemma 2. □

Corollary 2. *Parameterization 2 with the timeout* $\tau_{C1} = 2\Phi\beta + \frac{\beta^2}{\alpha}[2\Delta + (2n-1)\Phi]$, $\tau_{C2} = [\Delta + n\Phi]\beta$, *and* $\tau_{C3} = [2\Delta + (2n-1)\Phi]\beta$ *ensures a* $\mathcal{P}_{lv3}(\phi)$, *if round* $3(\phi-1)$ *starts in a* $(\frac{n+1}{2})$*-good period.*

Proof. By Lemma 5, round $3(\phi-1)$ is uniform. Thus by the definition of *ElectCoord*, every process in Π_0 has the same coordinator. Applying the Lemmas 3 to 5 and replacing the equations for τ_{C1}, τ_{C2}, and τ_{C3} 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 $\mathcal{P}_{lv3}()$. For the predicate $\mathcal{P}_{lv3}()$, the initialization time is the time from the start of a good period until all processes start the first round of phase ϕ satisfying $\mathcal{P}_{lv3}(\phi)$. 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*

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

the generic algorithm with Parameterization 2 and timeouts according to Corollary 2 ensures y *consecutive phases that fulfill* $\mathcal{P}_{lv3}(\phi)$.

Proof. It can be shown that the "initialization" period (see Fig. 7) is the longest in the case t_g starts just after the first send step of some round $3(\phi-2)$, and round $3(\phi-2)$ is not uniform (only round $3(\phi-1)$ is uniform). The end of round $3(\phi-1)$ corresponds to the end of the "initialization" period. By time $t_g + (2n+1)\Phi + \tau_{U3} + \tau_{U1} + \tau_{U2}$ round $3(\phi-1)$ is started at some process p_1. This round $3(\phi-1)$ ends for all processes in Π_0 at the latest $(3n-1)\Phi + \tau_{U3} + \Delta$ time later (end of "initialization" period) using the same argument as in Theorem 1. By Corollary 2, we have $\mathcal{P}_{lv3}(\phi)$. Every "regular" phase then takes at most time $\tau_{U1} + \tau_{U2} + \tau_{U3} + (2n+2)\Phi$. 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 $3\phi - 1$ to terminate by the expiration of a timeout instead of terminating it by reception of a message from the coordinator. The other solution, based on piggybacking, 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.

This mechanism leads to an improved version of phase synchronization, called *Piggybacking*, in which the second round message of LV-3 is piggybacked to the third round message. By this optimization, the length of a good period for LV-3 can be reduced approximately by one Δ, while the message size is increased only by a small constant factor. The expression can be calculated by applying a similar analysis as before

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

The other benefit of piggybacking is to speed up best case scenarios, where $\Pi_0 = \Pi$ 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 $\mathcal{P}_{lv4}$, 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 $O(n^2)$ per regular phase during a good period, at the cost of a slightly larger length of the good period.

The predicate $\mathcal{P}_{lv4}$, contrary to $\mathcal{P}_{lv3}$, 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 $\mathcal{P}_{lv3}$. As for $\mathcal{P}_{lv3}$, we do this only once per phase, in the last round of a phase. This leads to the message pattern depicted in Fig. 8.

The messages represented by a full line are required by $\mathcal{P}_{lv4}$, while the messages represented by a dotted line in round 4ϕ are only for synchronization and election of a coordinator.

Section 8.1 describes the implementation in more detail. The values for timeouts are presented in Section 8.2 and the length of a good period is proved in Section 8.3.

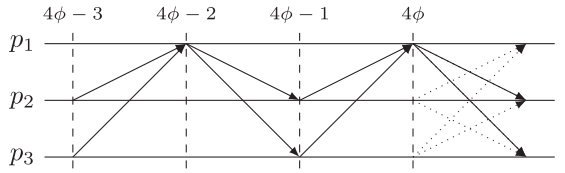

Fig. 8. Message pattern of synchronization by coordinator.

8.1 Outline of Synchronization by a Coordinator

Our algorithm requires only n messages in round 4ϕ during a good period. This optimization is based on the following two observations: 1) to choose a coordinator, it is enough for the HO sets to be nonempty, as long as the round is uniform, and 2) 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 4ϕ by timeout need to send a message to all (line 3 of *Dest* in Parameterization 3). Otherwise, if a process p receives a round 4ϕ message while in a lower round, p can advance to round 4ϕ silently, since there is at least another process that sent a message to all in round 4ϕ and, therefore, can be chosen as coordinator. This strategy results in a message complexity of cn for the election in round 4ϕ, 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 4ϕ message from the coordinator while in round $4\phi - 1$.

To reduce the time needed to start a phase in which $\mathcal{P}_{lv4}$ might hold, our algorithm skips some rounds of phase ϕ if it detects that $\mathcal{P}_{lv4}(\phi)$ cannot hold. This can happen in two cases: 1) in round $4\phi - 2$ by the coordinator if it does not receive a majority of messages during round $4\phi - 1$ (*SkipRound*, line 2), and 2) in round $4\phi - 1$ by any process if it does not receive a message from the coordinator in round $4\phi - 2$ (*SkipRound*, line 3).

Parameterization 3. LV-4 using synchronization by coordinator; where $ho(r) := \{q \mid \langle -, q, r\rangle \in Rcv\}$

$$NextRound(p, r, \tau, coord, Rcv) :=$$
$$\bigvee \begin{cases} \exists \langle -, -, r'\rangle \in Rcv :\ r' > r \\ r \bmod 4 = 0 \wedge \tau \geq \tau_{C4} \\ r \bmod 4 = 1 \wedge (\tau \geq \tau_{C1} \vee |ho(r)| > n/2) \\ r \bmod 4 = 2 \\ r \bmod 4 = 3 \wedge (\tau \geq \tau_{C3} \vee |ho(r)| > n/2) \end{cases}$$

$$SkipRound(p, r, \tau, coord, Rcv) :=$$
$$\bigvee \begin{cases} \exists \langle -, -, r'\rangle \in Rcv :\ r' > r \\ r \bmod 4 = 2 \wedge (p = coord \wedge |ho(r-1)| \leq n/2) \\ r \bmod 4 = 3 \wedge coord \notin ho(r-1) \end{cases}$$

$$Dest(p, r, \tau, coord, Rcv) :=$$
$$\begin{cases} coord & \text{for } r \bmod 4 \in \{1, 3\} \\ \Pi & \text{for } r \bmod 4 = 2 \wedge p = coord \\ \Pi & \text{for } r \bmod 4 = 0 \wedge ho(r) = \emptyset \\ \emptyset & \text{else} \end{cases}$$

$$ElectCoord(p, r, \tau, coord, Rcv) :=$$
$$\begin{cases} \min(\Pi) & \text{for } r = 1 \\ \min(ho(r-1)) & \text{for } r \bmod 4 = 1 \wedge ho(r-1) \neq \emptyset \\ coord & \text{else} \end{cases}$$

8.2 Timeouts τ_{C1}, τ_{C3}, τ_{C4}

Lemmas 6, 7, and 8 below establish results about timeouts τ_{C1}, τ_{C3}, and τ_{C4}. The proofs are similar to those in Section 7, and can be found in the supplemental appendix, available on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2011.48.

Lemma 6 (Timeout τ_{C1}). *Consider Parameterization 3 with $\tau_{C1} = [3\Delta + 3n\Phi]\beta + \tau_{C4}(\frac{\beta}{\alpha} - 1)$. Assume every process starts round $4(\phi - 1)$ in an $(\frac{n+1}{2})$-good period and phase ϕ has a unique coordinator c. Then, 1) c hears from a majority of processes in round $4\phi - 3$, and 2) all processes in Π_0 hear from c in round $4\phi - 2$.*

Lemma 7 (Timeout τ_{C3}). *Consider Parameterization 3 with the timeout $\tau_{C3} = [3\Delta + 2n\Phi]\beta$. Assume every process starts round $4(\phi - 1)$ in an $(\frac{n+1}{2})$-good period, and ϕ has a unique coordinator c. Then, 1) c hears from a majority of processes in round $4\phi - 1$, and 2) all processes in Π_0 hear from c in round 4ϕ.*

Lemma 8 (Timeout τ_{C4}). *Consider Parameterization 8 with the timeout $\tau_{C4} = [2\Delta + (2n-3)\Phi]\beta$.[2] 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 Π_0 by time t_g. Then, every round $4\phi > r_0$ started after time t_g is uniform with nonzero cardinality.*

Corollary 3. *Parameterization 3 with the timeout $\tau_{C1} = [\Delta + (n+3)\Phi]\beta + [2\Delta + (2n-3)\Phi]\frac{\beta^2}{\alpha}$, $\tau_{C3} = [3\Delta + 2n\Phi]\beta$, and $\tau_{C4} = [2\Delta + (2n-3)\Phi]\beta$ ensures $\mathcal{P}_{lv4}(\phi)$, if round $4(\phi - 1)$ starts in an $(\frac{n+1}{2})$-good period.*

Proof. By Lemma 8, round $4(\phi - 1)$ is uniform with nonzero cardinality. Thus by the definition of *ElectCoord*, every process in Π_0 has the same coordinator. Applying Lemmas 6 to 8 and solving the equations for τ_{C1}, τ_{C2}, and τ_{C4} yields the result. □

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 $\mathcal{P}_{lv4}(\phi)$. Just like with the predicate $\mathcal{P}_{lv3}(\phi)$, 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*

$$y\left[(2\Delta + (2n-3)\Phi)\frac{\beta}{\alpha} + 4\Delta + (2n+5)\Phi\right] + (2\Delta + (2n-3)\Phi)\frac{\beta^2}{\alpha^2} + (5\Delta + (5n-3)\Phi)\frac{\beta}{\alpha} + \Delta + (3n+1)\Phi,$$

the generic algorithm with Parameterization 3 ensures y consecutive phases ϕ that fulfill $\mathcal{P}_{lv4}(\phi)$.

2. Note that the timeout is different from the one in Lemma 2, since in some cases, processes do not send messages in this round.

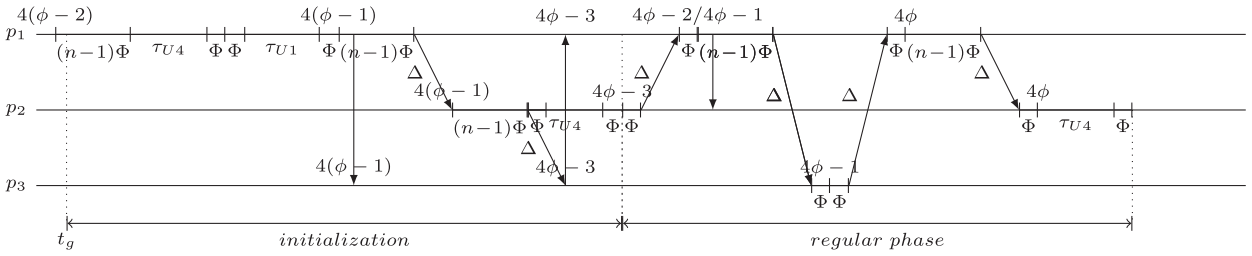

Fig. 9. Synchronization by a coordinator: Theorem 3.

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

Initialization period. It can be shown that the "initialization" period is the longest in the following case: 1) t_g starts just after the first send step of the highest round $4(\phi-2)$ reached by some process p, and 2) 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 5Δ. 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 Parameterization 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 Δ later. Round $4(\phi-1)$ terminates by the expiration of a timeout (2Δ), see lines 2 of $NextRound$ in Parameterization 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 Π_0 have finished round $4(\phi-1)$, which ends the initialization period and starts a regular phase. Thus, the initialization period lasts for 8Δ and a regular phase starts at $t_g+8\Delta$.

Regular phase. We show that the duration of a regular phase is 6Δ. Rename the regular phase to ϕ. Let the last process enter the regular phase ϕ at time t_r. Then by $t_r+\Delta$, the coordinator has a majority of round $4\phi-3$ messages, and 2Δ later, the coordinator receives the round $4\phi-1$ messages from all processes in Π_0. By $t_r+4\Delta$, the coordinator message of round 4ϕ is ready for reception at all processes in Π_0. Round 4ϕ terminates by the expiration of a timeout (2Δ). 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 ϕ is measured as the time since the last process enters round $4\phi-3$ until the last process ends the round 4ϕ.

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 1) a new round $4(\phi-2)$, or 2) 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 1 happens first, the latest by $t_{s4a} = t_g + n\Phi + \tau_{L3}\frac{\beta}{\alpha}$, 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 2 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 $t_{s4b} = t_g + (n+1)\Phi + (\tau_{L4}+\tau_{L1})\frac{\beta}{\alpha}$, 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 $\mathcal{M}_0$ be the set of processes from which p_1 received round $4(\phi-1)-3$ messages. Processes in $\mathcal{M}_0$ 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 $t_g + n\Phi + \tau_{L4}\frac{\beta}{\alpha}$ they are in round $4(\phi-1)-3$, and $\Delta+\Phi$ time later the message was received by the coordinator. Therefore, the latest by $t_g + \Delta + (n+2)\Phi + \tau_{L4}\frac{\beta}{\alpha}$ 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 $t_{s4c} = t_g + 2\Delta + (2n+2)\Phi + \tau_{L4}\frac{\beta}{\alpha}$, 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_{s4a}, t_{s4b}, and t_{s4c}, we can conclude that by time $t_g + (n+2)\Phi + (\tau_{L4} + \tau_{L1})\frac{\beta}{\alpha}$, 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 $4(\phi-1)$. $\tau_{U4} + \Phi$ later all processes will have entered round $4\phi - 3$, which marks the end of the initialization. This happens by $t_e = t_g + \Delta + (3n+1)\Phi + (2\tau_{L4} + \tau_{L1})\frac{\beta}{\alpha}$.

We can use Corollary 3 to show that $\mathcal{P}_{lv4}(\phi)$ 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 $\mathcal{P}_{lv4}(\phi)$, 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 $2\Delta + (n+2)\Phi$. Therefore, at time $t_e + 4\Delta + (2n+4)\Phi$, all processes have advanced to round 4ϕ, and $\tau_{L4}\frac{\beta}{\alpha} + \Phi$ later have completed round 4ϕ. Therefore, the first phase ends at $t_e + 4\Delta + (2n+5)\Phi + \tau_{L4}\frac{\beta}{\alpha}$. By applying the same reasoning, we can show that the following phases will take the same time. Therefore, y good phases will complete by $t_e + y[4\Delta + (2n+5)\Phi + \tau_{L4}\frac{\beta}{\alpha}]$. 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 $\mathcal{P}_{lv4}$. □

9 Comparison

In this section, we compare quantitatively the algorithms analyzed above. However, before doing so, we would like to clarify the scope of our results. Our analysis and conclusions are valid for the round-based algorithms above and can be easily adapted to other round-based algorithms. However, the results do not apply directly to failure detector-based algorithms [3] or Paxos-like protocols [12]. The key difference is that round-based protocols—like the ones presented in this paper—are usually driven by timeouts, i.e., these protocols require at least one process to expire its round timeout in every round, in order to proceed to the next round. In contrast, algorithms for the asynchronous or partially synchronous model usually proceed as fast as messages are sent and received, i.e., they are message driven [3], [12].

9.1 Impact of Clock Precision

First, we analyze the impact of the clock on τ_{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 Φ. The results are shown in Fig. 10. The x-axis corresponds to β/α (see Section 4). Larger values of β/α correspond to larger variations in clock skew (worse clock *precision*); identical clock skew (including perfect clocks for which $\alpha = \beta = 1$) corresponds to $\beta/\alpha = 1$.

The y-axis corresponds to τ_{good}/Δ, which is the duration of a good period expressed using Δ as the time unit.

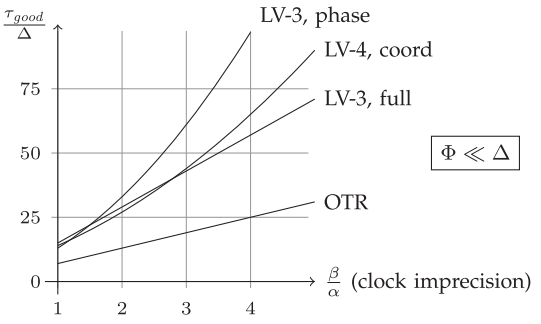

Fig. 10. Duration of good period as a function of clock drift.

Fig. 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 Φ 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 $\mathcal{P}_{otr}$), we have one single option to consider, namely, two uniform rounds ($\mathcal{P}_u$). For LV-3 (predicate $\mathcal{P}_{lv3}$), we have three options: three uniform rounds (line 2), Phase Sync (line 3), and Piggybacking (line 4). Finally for LV-4 (predicate $\mathcal{P}_{lv4}$), we have two options: four uniform rounds (line 5) and Coord Sync (line 6), which is designed specifically for $\mathcal{P}_{lv4}$. 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 $x = 0$ for the initialization time, time for two rounds for consensus). Line 2 (LV-3, $3 \times \mathcal{P}_u$) follows from Corollary 1. Line 3 (LV-3, Phase Synch) 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, $4 \times \mathcal{P}_u$) follows from Corollary 1 and, finally, line 6 (LV-4 Coord Synch) 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, in terms of the length of a good period, varies significantly depending on the round implementation. For example, the

TABLE 2
Summary of Results for $\beta/\alpha = 1$

	Alg.	# rounds	Resilience	Pred. Impl.	length of good period		# messages	
					initialization	per consensus	initialization	per consensus
1	OTR	2	$3f+1$	$2 \times \mathcal{P}_u$	$3\Delta + (4n-1)\Phi$	$4\Delta + (6n-2)\Phi$	n^2	$2n^2$
2	LV-3	3	$2f+1$	$3 \times \mathcal{P}_u$	$9\Delta + (13n-4)\Phi$	$6\Delta + (9n-3)\Phi$	$4n^2$	$3n^2$
3	LV-3	3	$2f+1$	Phase Sync	$8\Delta + (12n-1)\Phi$	$5\Delta + (7n+2)\Phi$	$2n^2+2n$	n^2+2n
4	LV-3	3	$2f+1$	Piggybacking	$8\Delta + (12n-1)\Phi$	$4\Delta + (4n+1)\Phi$	$2n^2+2n$	n^2+2n
5	LV-4	4	$2f+1$	$4 \times \mathcal{P}_u$	$11\Delta + (12n-5)\Phi$	$8\Delta + (9n-4)\Phi$	$5n^2$	$4n^2$
6	LV-4	4	$2f+1$	Coord Sync	$8\Delta + (10n-5)\Phi$	$6\Delta + (4n+2)\Phi$	$2n^2+3n$	$4n$

generic implementations of $\mathcal{P}_{lv3}$ and $\mathcal{P}_{lv4}$, 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 $\mathcal{P}_{lv3}$ over uniform rounds is 9Δ and the decision time is 6Δ (line 2), while an improved implementation of this predicate (Piggybacking) achieves 8Δ and 4Δ, 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 (both in terms of length of a good period and message complexity) 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 $\mathcal{P}_u$), where every round is of the same duration, the performance can be easily estimated as $(round\ duration) \times (\#rounds\ of\ algorithm)$. 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 ("$\Delta/rounds$") for each of our algorithms, considering the best round implementation for LV-3 and LV-4 (both in terms of length of a good period and message complexity). The table shows the results both for short good periods and long good periods. As we have mentioned in Section 1, 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 3.5Δ (OTR and LV-4) to 4Δ (LV-3), and a large variation for long good periods, from 1.3Δ (LV-3) to 2Δ (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 $2 \times \mathcal{P}_u$ is the algorithm with the shortest initialization time, namely 3Δ, compared to 8Δ for LV-3 and LV-4. OTR has also the shortest time until the first decision (7Δ) and the shortest time per consensus after initialization (4Δ). 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 (4Δ instead of 6Δ). This is to be expected, as LV-3 and LV-4 differ on the last rounds: one all-to-all round for LV-3 versus two rounds for LV-4 (all-to-coordinator and an 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.

TABLE 3
Average Round Duration ($\beta/\alpha = 1$, $\Phi \ll \Delta$)

		Short good periods		Long good periods	
Alg.	# rounds	1st decision	$\Delta/rounds$	per-consensus	$\Delta/rounds$
OTR	2	7Δ	3.5	4Δ	2
LV-3 (Piggybacking)	3	12Δ	4	4Δ	1.3
LV-4 (Coord Sync)	4	14Δ	3.5	6Δ	1.5

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 considering 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 implementations of rounds, and have computed for each algorithm 1) the time from the beginning of a good period until the first decision, and 2) 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 that the number of rounds of an algorithm is not always a good metric for the performance of an algorithm. Finally, we can observe trade-offs 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 nonround-based consensus algorithms, in order to understand the performance trade-offs between round-based and nonround-based consensus algorithms, and to understand whether the overall conclusions would be similar or very different.

Acknowledgments

Research funded by the Swiss National Science Foundation under grant number 200021-111701 and Hasler Foundation under grant number 2070. Nuno Santos was partially funded by grant SFRH/BD/17276/2004 of the Portuguese Foundation for Science and Technology (FCT).

References

[1] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers, "How to Solve Consensus in the Smallest Window of Synchrony," *Proc. 22nd Int'l Conf. Distributed Computing (DISC '08)*, pp. 32-46, 2008.

[2] F. Borran, M. Hutle, N. Santos, and A. Schiper, "Quantitative Analysis of Consensus Algorithms," Technical Report EPFL-REPORT-150216, EPFL, 2010.

[3] T.D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM*, vol. 43, no. 2, pp. 225-267, Mar. 1996.

[4] B. Charron-Bost and A. Schiper, "Improving Fast Paxos: Being Optimistic with no Overhead," *Proc. Pacific Rim Int'l Symp. Dependable Computing*, 2006.

[5] B. Charron-Bost and A. Schiper, "The Heard-of Model: Computing in Distributed Systems with Benign Faults," *Distributed Computing*, vol. 22, pp. 49-71, 2009.

[6] P. Dutta, R. Guerraoui, and I. Keidar, "The Overhead of Consensus Failure Recovery," *Distributed Computing*, vol. 19, nos. 5/6, pp. 373-386, Apr. 2007.

[7] P. Dutta, R. Guerraoui, and L. Lamport, "How Fast Can Eventual Synchrony Lead to Consensus," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '05)*, pp. 22-27, 2005.

[8] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM*, vol. 35, no. 2, pp. 288-323, Apr. 1988.

[9] M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, no. 2, pp. 374-382, Apr. 1985.

[10] E. Gafni, "Round-by-Round Fault Detectors (Extended Abstract): Unifying Synchrony and Asynchrony," *Proc. 16th Ann. ACM Symp. Principles of Distributed Computing (PODC '98)*, pp. 143-152, 1998.

[11] I. Keidar and A. Shraer, "Timeliness, Failure-Detectors, and Consensus Performance," *Proc. 25th Ann. ACM Symp. Principles of Distributed Computing (PODC '06)*, pp. 169-178, 2006.

[12] L. Lamport, "The Part-Time Parliament," *ACM Trans. Computer Systems*, vol. 16, no. 2, pp. 133-169, May 1998.

[13] L. Lamport, "Fast Paxos," Technical Report MSR-TR-2005-12, Microsoft Research, 2005.

[14] R.D. Prisco, B. Lampson, and N. Lynch, "Revisiting the Paxos Algorithm," *Proc. 11th Int'l Workshop Distributed Algorithms (WDAG '97)*, pp. 111-125, 1997.

[15] N. Santoro and P. Widmayer, "Time is Not a Healer," *Proc. Sixth Ann. Symp. Theoretical Aspects of Computer Science (STACS '89)*, pp. 304-313, Feb. 1989.

[16] A. Schiper, "Early Consensus in an Asynchronous System with a Weak Failure Detector," *Distributed Computing*, vol. 10, no. 3, pp. 149-157, Apr. 1997.

Fatemeh Borran received the master's and PhD degrees in computer science from EPFL in April 2006 and February 2011, respectively. She is currently a postdoctoral researcher at the Distributed Systems Laboratory at EPFL. Her research interests include distributed algorithms, fault tolerance, consensus problem, and data consistency in MANETs.

Martin Hutle received the master's and PhD degrees in computer science from the Vienna University of Technology in 2002 and 2005, respectively. He held postdoctoral positions at the Institute of Computer Engineering at the Vienna University of Technology from 2005 to 2006 and at the Distributed Systems Laboratory at EPFL from 2006 to 2010. Currently, he is working at the Fraunhofer Research Institution for Applied and Integrated Security. His research interests are in various topics in fault-tolerant distributed computing, real-time systems, and the security and safety of embedded systems.

Nuno Santos received the BSc degree in mathematics and the MSc degree in computer science from the University of Coimbra, Portugal, in 2000 and 2003, respectively. He is working toward the PhD degree at EPFL, working on the analysis, implementation, and evaluation of Paxos-like consensus algorithms. From 2003 to 2006, he worked for one year as a software engineer at Wit-Software, Portugal, and for two years at the European Organization for Nuclear Research (CERN), Switzerland. His research interests include fault-tolerant distributed systems, consensus algorithms, and performance evaluation.

André Schiper graduated in physics from the ETHZ in Zurich in 1973 and received the PhD degree in computer science from the EPFL in 1980. He has been a professor of computer science at EPFL since 1985, leading the Distributed Systems Laboratory. His research interests are in the areas of dependable distributed systems, middleware support for dependable systems, replication techniques (including for database systems), group communication, distributed transactions, and MANETs. He is member of the editorial boards of Springer-Verlag's *Distributed Computing* (2003-), the *IEEE Transactions on Dependable and Secure Computing* (2004-2008), and Inderscience's *International Journal of Security and Networks* (2005-). He is a member of the IEEE.