Synchronization modulo P in Dynamic Networks

Louis Penet de Monterno[a,*], Bernadette Charron-Bost[b], Stephan Merz[c]

[a] *École polytechnique, IP Paris, 91128, Palaiseau, France*
[b] *DI ENS, École Normale Supérieure, 45 rue d'Ulm, 75005, Paris, France*
[c] *University of Lorraine, CNRS, Inria, LORIA, 54000, Nancy, France*

Abstract

We define the mod P-synchronization problem as a weakening of the firing squad problem, where all nodes fire not at the same round, but at rounds that are all equal modulo P. We introduce an algorithm that achieves mod P-synchronization despite asynchronous starts in every dynamic network whose dynamic radius is bounded by some integer Δ, that is, there always exists a *temporal* path of length at most Δ from some fixed node γ, called a *central node* of the network, to all the other nodes. As opposed to the perfect synchronization achieved in the firing squad problem, mod P-synchronization thus does not require the network to be strongly connected. In our algorithm, nodes know Δ, but they ignore which nodes are central in the network. We also prove that if the bound Δ on the radius exists but is unknown, then mod P-synchronization is impossible.

All nodes in our algorithm fire in less that $6Pn$ rounds, where n is the number of nodes, after all nodes become active, but use unbounded counters. We then present a refinement of this algorithm so that memory usage becomes bounded while maintaining the same time complexity. The correctness of our first algorithm has been formally established in the proof assistant Isabelle.

Keywords: Distributed Computing, Dynamic Graph, Synchronization, Firing Squad

1. Introduction

Distributed algorithms are often designed in a synchronous computing model, in which computation is divided into *communication-closed rounds*: any message sent at some round can be received only at that round. In this model, it is usually assumed that in each run of an algorithm, all nodes start simultaneously, i.e., at the same round, or even at round one. For instance, most synchronous consensus algorithms (e.g., [19, 12, 21]), as well as many distributed algorithms for dynamic networks (e.g., [15, 16]) require synchronous starts.

*Corresponding author.

This assumption makes the sequential composition of two distributed algorithms $A; B$ – in which each node starts executing B when it has completed the execution of A – quite problematic. Indeed, nodes start the algorithm B asynchronously when the algorithm A terminates asynchronously, and the properties of B are no longer guaranteed in this context of asynchronous starts.

This leads to the problem of simulating synchronous starts, classically referred to as the *firing squad problem*: each node is initially *passive* and then becomes *active* at an unpredictable round. The goal is to guarantee that the nodes, once they are all active, eventually synchronize by *firing* – i.e., entering a designated state for the first time – at the same round.

Unfortunately, the impossibility result in [7] demonstrates that the firing squad problem is not solvable without a strong connectivity property of the network, namely, there exists some positive integer d such that the communication graph within every period of d consecutive rounds is strongly connected and a bound Δ on the delay d is known, in the sense that algorithms depend on Δ. In many situations, this connectivity property is not guaranteed: as an example, in the dynamic graphs corresponding to the Heard-Of models for benign failures [9], a node that suffers permanent and complete send omissions is constantly a sink in the communication graph.

However, looking more closely at many distributed algorithms designed in the round-based model, we see that these algorithms actually do not require perfectly synchronous starts, and still work under the weaker condition that all the nodes start executing the algorithms in rounds with numbers that are equal modulo P, for some positive integer P. In this paper, the equality modulo P is denoted $\equiv_P$. The corresponding synchronization problem, that we call mod P-*synchronization*, is formally specified as follows:

Termination. If all nodes become active, then every node eventually fires.

mod P-simultaneity. If two nodes fire at rounds t and t', then $t' \equiv_P t$.

Indeed, let A be an algorithm structured in regular *phases* consisting of a fixed number P of consecutive rounds: the behaviour of each node (i.e., the update rule of its state and the message it sends) at round t is determined by the value of t modulo P. Moreover, assume that A has been proved correct with respect to some specification when all nodes start A synchronously (at round one), but with any dynamic graph in a family $\mathcal{G}$ that is stable under the addition of arbitrary finite prefixes. For instance, the *ThreePhaseCommit* algorithm for non-blocking atomic commitment [3], as well as the consensus algorithms in [13] or the *LastVoting* algorithm [9] – corresponding to the consensus core of *Paxos* [17] – fulfill all the above requirements for phases of length $P = 3$ and $P = 4$, respectively, and the family $\mathcal{G}$ of dynamic graphs in which there exists an infinite number of "good" communication patterns (e.g., a sequence of $2P$ consecutive communication graphs in which a majority of nodes is heard by all nodes in each graph). The use of a mod P-synchronization algorithm prior to the algorithm A yields a new algorithm that executes exactly like A does, after a finite preliminary period during which every node becomes active and fires.

The above property on the set of dynamic graphs $\mathcal{G}$ then guarantees this variant of A to be correct with asynchronous starts and dynamic graphs in $\mathcal{G}$.

Another typical example for which the perfect synchronization requirement in the firing squad problem can be weakened into mod P-synchronization is the development of the basic *rotating coordinator* strategy for a given algorithm C in the context of asynchronous starts. Roughly speaking, this strategy consists in the following: each node u has unique identifiers in $\{1, \ldots, n\}$, and maintains a local counter c_u whose current value is the number of rounds elapsed since the node u started executing C. At each round, the coordinator of u is the node with the identifier that is equal to the current value of c_u modulo n. Since there may be only one coordinator per round, such a selection rule requires synchronized counters. Clearly, with the use of a mod n-synchronization algorithm in a preliminary phase, the above scheme implements the rotating coordinator strategy from the first round where all nodes have fired.

The mod P-*synchronization* problem is clearly related to the synchronization problem of periodic clocks that has been extensively studied (e.g., see [1, 14, 5]). In the latter problem, only *eventual* synchronization is required, and nodes are not aware of the round at which synchronization is achieved (no "firing event").

The definition of this "mod-P firing squad" problem is one of the contributions in the paper. A natural question is then whether mod P-synchronization may be achieved without strong connectivity. In this paper, we address this issue and show that this problem is solvable under the assumption that a bound Δ on the radius of the network is given, that is, every node receives a message from some central node (possibly indirectly) in every period of Δ consecutive rounds. By contrast, the firing squad problem is only solvable in strongly connected dynamic graphs [8]. In other words, every node must be central. In fact, we exhibit an algorithm, denoted by $\mathit{SynchMod}_P$, that achieves synchronization modulo P in any dynamic graph, assuming $\Delta \leq P$. When $\Delta > P$, one can find an integer M such that $\Delta \leq PM$ and apply the $\mathit{SynchMod}_{PM}$ algorithm in order to solve the mod PM-synchronization problem, and hence the mod P-synchronization problem. In this sense, the case $\Delta > P$ can be reduced to the case $\Delta \leq P$. This reduction is presented in Section 3.4. Interestingly, our algorithm requires no node identifiers. In particular, nodes are not assumed to know which nodes are central in the graph. Other than the radius of the communication graph being at most Δ, no other assumption is made on the dynamic graph.

The correctness proof of our algorithm relies on a series of preliminary lemmas that consider all the possible cases for the respective values of the variables in the algorithm. In order to increase our confidence in the correctness and remove any doubts on such combinatorial proofs, we have developed a formal proof of the correctness of our algorithm[1] in the interactive theorem prover Is-

[1]The complete Isabelle development is available at `https://github.com/louisdm31/asynchronous_starts_HO_model/tree/master/proof/sync-mod`. We provide in the appendix the Isabelle definition of the $\mathit{SynchMod}_P$ algorithm we used in our formal proof.

abelle/HOL [18]. The "paper and pencil" proof presented in this article closely follows our formal proof.

This paper is a revised version of the conference paper [20]. This version provides more detailed explanations of the algorithm, and in particular detailed proofs of every lemma, whereas our previous publication only contained the proofs of the main lemmas. Moreover, we now provide a constructive liveness proof, that directly entails the time complexity. In contrast, the previous version contains a non-constructive liveness proof, and the time complexity comes from a separate proof. The Isabelle proof has been updated to follow the rewritten proof.

The paper is structured as follows. Section 2 formally defines the computational model. Section 3.1 introduce the $\mathit{SynchMod}_P$ algorithm. Sections 3.2 and 3.3 provide a detailed proof of correctness of the $\mathit{SynchMod}_P$ algorithm in the particular case $P > 2$ and a bound Δ on the radius where $\Delta \leq P$. Section 3.4 generalizes the results of the preceding section and in particular shows impossibility when Δ is not known. Section 3.5 introduces a variation of the $\mathit{SynchMod}_P$ algorithm that reduces memory usage. Finally, Section 4 concludes the paper.

2. Preliminaries

2.1. The Computational Model

We consider a networked system with a *fixed* set V of n nodes. We assume a round-based computational model in the spirit of the Heard-Of model [9], in which point-to-point communications are organized into *synchronized rounds*: each node sends messages to all nodes and receives messages sent by *some* of the nodes. Rounds are communication closed in the sense that no node receives messages in round t ($t = 1, 2, \ldots$) that are sent in a round different from t. The collection of communications (which nodes receive messages from which nodes) at each round t is modelled by a directed graph (digraph, for short) with a set of nodes equal to V. The digraph at round t is denoted by $\mathbb{G}(t) = (V, E_t)$, and is called the *communication graph at round* t. The set of u's incoming neighbors in the digraph $\mathbb{G}(t)$ is denoted by $\mathit{In}_u(t)$.

We assume a self-loop at each node in all these digraphs since every node can communicate with itself instantaneously. The sequence of such digraphs $\mathbb{G} = (\mathbb{G}(t))_{t\geq 1}$ is called a *dynamic graph* [6].

In round t, each node u successively (a) broadcasts[1] messages determined by its state at the beginning of round t, (b) receives *some* of the messages sent to it, and finally (c) performs an internal transition to a successor state. A *local algorithm* for a node is given by a *sending function* that determines the

[1] The system is anonymous and nodes have no knowledge about the dynamic graph. Therefore, the only valid way to send messages is "send to all". The communication graph describes which message will actually be received.

messages to be sent in step (a) and a *transition function* for state updates in step (c). An *algorithm* for the set of nodes V is a collection of local algorithms, one per node.

We also introduce the notion of *start schedules*, represented as collections $\mathbb{S} = (s_u)_{u \in V}$, where each s_u is a positive integer or is equal to ∞.

The execution of an algorithm A with the dynamic graph $\mathbb{G}$ and the start schedule $\mathbb{S}$ then proceeds as follows: Each node u is initially *passive*. If $s_u = \infty$, then the node u remains passive forever. Otherwise, s_u is a positive integer, and u becomes *active* at the beginning of round s_u, setting up its local variables. In round t ($t = 1, 2 \dots$), a passive node sends only heartbeats, corresponding to *null* messages, and cannot change its state. An active node applies its sending function in A to its current state to generate the messages to be sent, then it receives the messages sent by its incoming neighbors in the directed graph $\mathbb{G}(t)$, and finally applies its transition function $\mathcal{T}_u$ in A to its current state and the list of messages it has just received (including the null messages from passive nodes), to compute its next state. Since each local algorithm is deterministic, an execution of the algorithm A is entirely determined by the initial state of the network, the dynamic graph $\mathbb{G}$, and the start schedule $\mathbb{S}$.

The states "passive" and "active" do not refer to any physical notion, and are relative to the algorithm under consideration: as an example, if two algorithms A and B are sequentially executed according to the order "A followed by B", then at some round, a node may be active w.r.t. A while it is passive w.r.t. B. In such a situation, the node is integrally part of the system and can send messages, but these messages are empty with respect to the semantics of the algorithm B. Those messages are then interpreted as heartbeats by B.

As this paper is based on the synchronized computing model, this begs the question of the realism of this assumption in real-world systems. In fact, synchronized rounds can be emulated in any asynchronous system. A minimal implementation would work as follows. Each node would hold a local clock which is incremented after each state update. Each message would be tagged by this clock, such that the communication-closure property could be enforced. In each asynchronous execution, the set of received messages by each node between each state update would yield a dynamic graph that characterizes the corresponding synchronous execution. This construction does not require any assumption. The remaining question is how to guarantee that the resulting dynamic graph satisfies a certain property, and which assumption on the underlying asynchronous system is necessary. This question has been the focus of several papers [10, 2, 9]. As an example, consider the case of a non-faulty system for which there is an upper bound on the transit time of each message sent by some node γ. In such a system, it it possible to construct synchronous rounds in which the radius of dynamic graph is equal to 1, and hence, the solvability result of this paper applies. This scenario could be refined by allowing some failures.

2.2. Network Model and Start Model

Let us first recall the notion of *product* of two digraphs $G_1 = (V, E_1)$ and $G_2 = (V, G_2)$, denoted by $G_1 \circ G_2$ and defined as follows [8]: $G_1 \circ G_2$ has V as its set of nodes, and (u, v) is an edge if there exists $w \in V$ such that $(u, w) \in G_1$ and $(w, v) \in G_2$. For any dynamic graph $\mathbb{G}$ and any integer $t' > t \geq 1$, we let

$$\mathbb{G}(t : t') \stackrel{\text{def}}{=} \mathbb{G}(t) \circ \mathbb{G}(t+1) \circ \cdots \circ \mathbb{G}(t').$$

By extension, we let $\mathbb{G}(t : t) = \mathbb{G}(t)$. The set of incoming neighbors of u in $\mathbb{G}(t : t')$ is noted as $In_u(t : t')$.

Each edge (u, v) in the digraph $\mathbb{G}(t : t')$ corresponds to a *path* $u \triangleright v$ *in the interval* $[t, t']$, i.e., a finite sequence of nodes $u = w_{t-1}, w_t, \ldots, w_{t'} = v$ such that each pair (w_{i-1}, w_i) is an edge of $\mathbb{G}(i)$. This path is said to be *active* if each node $w_{t-1}, w_t, \ldots, w_{t'}$ is active in rounds $t-1, t, \ldots t'$, respectively.

The *eccentricity of a node* u in a dynamic graph $\mathbb{G}$ is defined as

$$e_{\mathbb{G}}(u) \stackrel{\text{def}}{=} \inf\{d \in \mathbb{N}^+ \mid \forall t \in \mathbb{N}^+, \forall v \in V : (u, v) \text{ is an edge in } \mathbb{G}(t : t+d-1)\}.$$

The node u is *central in* $\mathbb{G}$ if its eccentricity is finite. The *radius* of $\mathbb{G}$ is then defined as:

$$\mathrm{rad}(\mathbb{G}) \stackrel{\text{def}}{=} \inf_{i \in V} e_{\mathbb{G}}(i).$$

A *network model* is any non-empty set of dynamic graphs. We will focus on those network models $\mathcal{G}^*_\Delta$ of dynamic graphs $\mathbb{G}$ satisfying $0 < \mathrm{rad}(\mathbb{G}) \leq \Delta$, namely,

$$\exists \gamma \in V, \forall t \in \mathbb{N}, \forall u \in V, \gamma \in In_u(t+1 : t+\Delta).$$

In particular, the network model $\mathcal{G}^*_\Delta$ contains some dynamic graphs which are partitioned during less than Δ consecutive rounds. We can easily check that, because of self-loops, any dynamic graph $\mathbb{G}$ that permanently contains a spanning tree rooted in some node γ satisfies

$$\mathrm{rad}(\mathbb{G}) \leq e_{\mathbb{G}}(\gamma) \leq |V| - 1.$$

Any dynamic graph which is permanently strongly connected also satisfies $\mathrm{rad}(\mathbb{G}) \leq |V| - 1$, *a fortiori*.

We also define a *start model* as a non-empty set of start schedules. A start schedule $\mathbb{S} = (s_u)_{u \in V}$ is *complete* if every s_u is finite, i.e., no node is passive forever. Synchronous starts correspond to complete start schedules where all s_u are finite and equal. The point of this paper is to simulate mod P*-synchronous starts* defined by $s_u \equiv_P s_v$ for every pair of nodes u and v, with any complete start schedule.

3. The Algorithm

3.1. Definition and Informal Description of the Algorithm

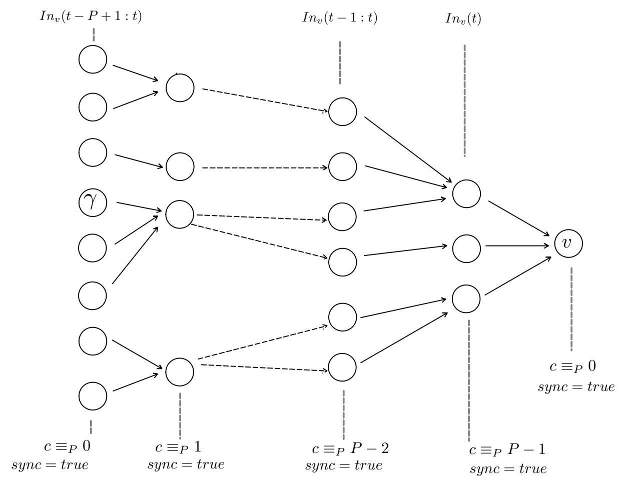
We fix some $P > 2$. The *SynchMod*$_P$ algorithm appears as Algorithm 1. Each node holds a *level* variable. When it becomes active, it moves from passive

Algorithm 1: The $\mathit{SynchMod}_P$ algorithm.

Initialization:
  $c_u \in \mathbb{N}$, initially 0
  $\mathit{synch}_u \leftarrow \mathit{false}$
  $\mathit{ready}_u \leftarrow \mathit{false}$
  $\mathit{force}_u \in \{0, 1, 2\}$, initially 0
  $\mathit{level}_u \in \{0, 1, 2\}$, initially 0

At each round:
  send $\langle c_u, \mathit{synch}_u, \mathit{force}_u, \mathit{ready}_u \rangle$ to all
  receive incoming messages: let In^a be the set of nodes from which a non-null message is received.
  if *all received messages are non-null* **then**
    $\mathit{synch}_u \leftarrow \bigwedge_{v \in In^a} \mathit{synch}_v \wedge c_v \equiv_P c_u$
  end
  else
    $\mathit{synch}_u \leftarrow \mathit{false}$
  end
  $\mathit{ready}_u \leftarrow \bigwedge_{v \in In^a} \mathit{ready}_v$
  $\mathit{force}_u \leftarrow \max_{v \in In^a} \mathit{force}_v$
  $c_u \leftarrow 1 + \min_{\substack{v \in In^a \\ \mathit{force}_v = \mathit{force}_u}} c_v$
  if $c_u \equiv_P 0$ **then**
    if $\mathit{level}_u = 0 \wedge \mathit{synch}_u$ **then**
      $\mathit{level}_u \leftarrow 1$
      if $\mathit{force}_u < 2$ **then**
        $\mathit{force}_u \leftarrow 1$
        $c_u \leftarrow 0$
      end
    end
    else if $\mathit{level}_u = 1 \wedge \mathit{ready}_u \wedge \mathit{synch}_u$ **then**
      $\mathit{level}_u \leftarrow 2$ `/* the node` u `fires */`
      $\mathit{force}_u \leftarrow 2$
      $c_u \leftarrow 0$
    end
    $\mathit{synch}_u \leftarrow \mathit{true}$
    $\mathit{ready}_u \leftarrow \mathit{level}_u > 0$
  end

Figure 1: Impact of the state of incoming neighbors of v between round $t-P$ and t on the decision of v in round t: case where every c_u is congruent to 0 in round $t-P$.

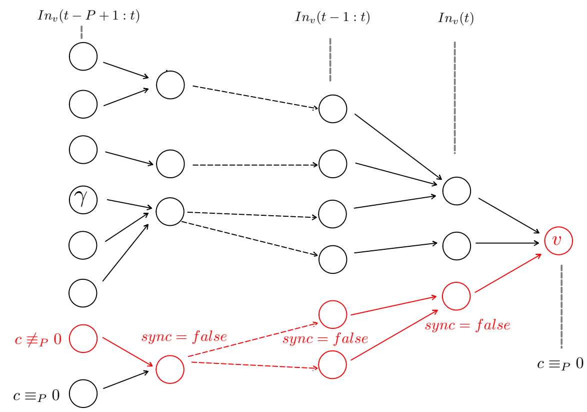


state to level 0. It later moves to level 1, then to level 2. Each time a node moves from some level to the next, this constitutes a *level-up event*. From now on, the level reached during a given level-up event will be called the *strength* of this event. Reaching level 2 means firing. The conditional statements at lines 20 and 27 of Algorithm 1 are executed when the node reaches level 1 and 2 respectively. The intuition of the algorithm can be summarized by two simple ideas.

Firstly, *each node keeps track of the most recent strongest level-up event.* Only the strongest level-up events are considered: if some node "knows" about a level-up event from level 1 to level 2, it will not record any level-up event from level 0 to level 1, nor any level-up event from passive state to level 0. Among the strongest level-up events, the nodes keep track of the age of the most recent one. This defines an ordering on the set of level-up events. For that purpose, they hold two variables c_u and $force_u$. At any round, node u knows that c_u rounds ago, some node reached a level equal to $force_u$ from the previous level (as will be proved in Lemma 6). If $z_u(t)$ denotes the level-up event that node u "remembers" in round t, then Lemma 7 shows that u only remembers the strongest most recent level-up event.

Secondly, let γ denote any central node, whose eccentricity is less or equal to P. *A node may level up in round t only if its counter c_u is congruent to zero, and the counter of γ was also congruent to zero, P rounds ago.*
Since the nodes do not know a fixed central node, they conservatively level up only if all of their incoming neighbors $v \in In_u(t-P+1:t)$ were congruent to zero P rounds ago. By definition, γ is one of these incoming neighbors. For that purpose, they use a Boolean variable *synch*. When the counter of some node v becomes congruent to zero in some round $t-P$, it sets its $synch_v$ variable

Figure 2: Impact of the state of incoming neighbors of v between round $t-P$ and t on the decision of v in round t: case where some c_u are not congruent to 0 in round $t-P$.



to *true* in line 32. During the next $P-1$ rounds, it will check whether the counters of its incoming neighbors are all congruent to its own counter (line 11). In case they are not, the node will set its $synch_u$ variable to *false*. This *false* value will disseminate to its outgoing neighbors (also line 11). Any node whose *synch* variable is false cannot move to the next level during the next P rounds. In contrast, if in round t, the $synch_u$ variable is still true, node u knows that no non-congruence was detected between round $t-P$ and round t. This means that every central node was congruent with zero in round $t-P$ (as will be proved in Lemma 3.b). In that case, a level-up event will take place (see Fig. 1). In contrast, if some node $v \in In_u(t-P+1:t)$ is not congruent to zero in round $t-P$, then the line 11 guarantees that $synch_u$ will ultimately be *false* at the beginning of round t (see Fig. 2). In addition to *synch*, the *ready* variable makes sure that a node u can move to level 2 only if, P rounds ago, γ was already in level 1 (as will be proved in Lemma 4). Intuitively, the round t_γ in which γ reaches level 1 is used as a landmark for the mod P-synchronization: Lemma 8 shows that nodes fire in rounds which are congruent to t_γ modulo P.

Observe that the presence of self-loops in each communication graph implies that, in the pseudo-code of Algorithm 1, the minima and maxima are well-defined.

3.2. Notation and Preliminary Lemmas

In the rest of this section, we fix an execution ρ of the $SynchMod_P$ algorithm for a complete activation schedule $\mathbb{S}$ and a dynamic graph $\mathbb{G} \in \mathcal{G}^*_\Delta$ with $\Delta \leq P$. Let $s^{\max} \stackrel{\text{def}}{=} \max_{u \in V} s_u$ (note that $s^{\max} < \infty$) and let γ denote some central node of $\mathbb{G}$ satisfying $\mathrm{e}_{\mathbb{G}}(\gamma) \leq \Delta$.

If the node u is active in round t, for every variable x of the algorithm, we denote the value of x_u just before u executes line 19 at round t and at the very end of round t by $x_u^{pre}(t)$ and $x_u(t)$ respectively. By extension, $x_u(t)$ refers to the initial state if $t = s_u - 1$. We now prove that this execution satisfies both properties of the mod P-synchronization problem.

We denote $In_u^a(t)$ the subset of nodes in $In_u(t)$ which are active in round $t-1$ in this execution. Some simple claims follow immediately from the definition of the transition function, regardless of the connectivity properties of $\mathbb{G}$. We consider some node $u \in V$ and some round t in which u is active (i.e., $t \geq s_u$).

Lemma 1.

(a) *$level_u(t+1) \in \{level_u(t), level_u(t)+1\}$*

(b) *If $c_u(t) \neq 0$, then $force_u(t) = force_u^{pre}(t)$ and $c_u(t) = c_u^{pre}(t)$.*

(c) *$c_u(t) \equiv_P c_u^{pre}(t)$.*

(d) *If $synch_u^{pre}(t) = true$ holds, then each node $v \in In_u(t)$ is active at round $t-1$ with: $c_v^{pre}(t-1) + 1 \equiv_P c_u^{pre}(t)$.*

(e) *If $c_u^{pre}(t) \not\equiv_P 1$ and $synch_u^{pre}(t)$ holds, then each node $v \in In_u(t)$ is active in round $t-1$ with $synch_v^{pre}(t-1)$.*

(f) *If $c_u^{pre}(t) \not\equiv_P 1$ and $synch_u^{pre}(t) = ready_u^{pre}(t) = true$, then for every node $v \in In_u^a(t)$, it holds that $ready_v^{pre}(t-1) = true$.*

(g) *For every $v \in In_u^a(t)$, we have*
$$force_v^{pre}(t-1) \leq force_v(t-1) \leq force_u^{pre}(t) \leq force_u(t).$$

(h) *For every $v \in In_u^a(t)$, if $force_v^{pre}(t-1) = force_u^{pre}(t)$ then*
$$c_u^{pre}(t) \leq 1 + c_v(t-1) \leq 1 + c_v^{pre}(t-1).$$

(i) *$level_u(t) \leq force_u(t)$.*

Proof.

(a) The value of $level_u(t+1)$ is equal to $level_u(t)$, unless line 21 or 28 is executed in round $t+1$. In that case, $level_u(t+1) = level_u(t) + 1$.

(b) If c_u is nonzero at the end of round t, then lines 24 and 30 cannot be executed during round t. Therefore, lines 23 and 29 are not executed either. Since no other lines starting at line 19 modify the variables $force_u$ or c_u, it follows that $force_u(t) = force_u^{pre}(t)$ and $c_u(t) = c_u^{pre}(t)$.

(c) The assignments in lines 22 and 27 ensure that $c_u(t)$ is 0, and they are executed only if $c_u^{pre}(t) \equiv_P 0$.

(d) Firstly, $c_v^{pre}(t-1)$ is well-defined because $In_u(t) = In_u^a(t)$ (see line 10). Moreover, the set $\{c_v(t-1), v \in In_u(t)\}$ contains integers which are mutually congruent modulo P (see line 11). Using claim 1.c and line 18
$$c_v^{pre}(t-1) + 1 \equiv_P c_v(t-1) + 1 \equiv_P c_u^{pre}(t).$$

(e) By claim 1.d, every incoming neighbor v of u is active in round $t-1$ and satisfies $c_v^{pre}(t-1) \not\equiv_P c_u^{pre}(t) - 1 \equiv_P 0$. Then the conditional statement starting in line 19 is not executed by v in round $t-1$. Then, both $synch_v(t-1)$ and $synch_v^{pre}(t-1)$ hold.

(f) Assume that $c_u(t) \not\equiv_P 1 \wedge ready_u^{pre}(t) \wedge synch_u^{pre}(t)$. Using the previous proof, every incoming neighbor v is active in round $t-1$ and the conditional statement starting in line 19 is not executed by v in round $t-1$. Finally, by line 16, v satisfies $ready_v^{pre}(t-1)$.

(g) This property follows directly from lines 17, 23 and 29.

(h) If $force_v^{pre}(t) = force_v^{pre}(t-1)$, then $force_v(t) = force_u^{pre}(t-1)$ by Lemma 1.g. Then $c_u^{pre}(t) \leq 1 + c_v(t-1) \leq 1 + c_v^{pre}(t-1)$ by line 18.

(i) We prove by induction on $t \geq s_u - 1$ that $\forall t \geq s_u - 1$, $level_u(t) \leq force_u(t)$.

   (a) If $t = s_u - 1$, then u is in the initial state in round t. Then $level_u(t) = force_u(t) = 0$.

   (b) Assume now that $level_u(t) \leq force_u(t)$. If u levels up in round $t+1$, then $level_u(t+1) \leq force_u(t+1)$ by lines 21, 23, 28. Otherwise, by Lemma 1.g and by induction hypothesis, $level_u(t+1) = level_u(t) \leq force_u(t) \leq force_u(t+1)$. □

Lemma 2. *No node can perform a level-up event action in round $P-1$ or earlier.*

Proof. We prove by induction on t that:

$$\forall t < P, \forall u \in V, t \geq s_u - 1 \Rightarrow c_u(t) \leq t \wedge force_u(t) = 0 \wedge \neg synch_u(t).$$

1. For the base case, any node active from the first round is in initial state in round 0:
$$c_u(0) = 0 \wedge force_u(0) = 0 \wedge \neg synch_u(0).$$

2. Let t be some integer in $\{0, \ldots, P-2\}$. Let u be some node which is active in round $t+2$. Either u is in its initial state in round $t+1$, and then clearly $c_u(t+1) = 0 \wedge force_u(t+1) = 0 \wedge \neg synch_u(t+1)$. Or u is active in round $t+1$. By induction hypothesis, every active incoming neighbor v of u in round $t+1$ has $force_v(t) = 0 \wedge \neg synch_v(t)$. Then $force_u^{pre}(t+1) = 0 \wedge \neg synch_u^{pre}(t+1)$. Using the induction hypothesis and line 18, we have $t+1 \geq c_u(t) + 1 \geq c_u^{pre}(t+1)$.

   Then $c_u^{pre}(t+1) \in \{1, \ldots, P-1\}$. By line 19, the variables of u are not modified in round $t+1$ after line 19. From previous claims, we obtain that $c_u(t+1) \leq t+1 \wedge force_u(t+1) = 0 \wedge \neg synch_u(t+1)$.

Using $\neg synch_u(t)$ and line 32, we obtain that a level-up event is impossible for any u, for $t < P$. □

We now show a few properties on the incoming neighbors of nodes that reach level 1 or 2. This situation is illustrated by Fig. 1.

Lemma 3. *Let i be an integer, $0 \leq i < P$, and let u and v be two nodes such that $u \in In_v(t-P+i+1 : t)$. If v is active and moves to level 1 or 2 in round t, then*

(a) *u is active in round $t-P+i$.*

(b) *$c_u^{pre}(t-P+i) \equiv_P i$.*

(c) *If $ready_v^{pre}(t)$ is true and $i > 0$, then $ready_u^{pre}(t-P+i)$ is true as well.*

Proof. By Lemma 2, $t \geq P$. Let $u = w_{t-P+i}, \ldots, w_t = v$ denote some $u \triangleright v$ path in the interval $[t-P+i+1, t]$. By a backward induction, we show that, for any $j \in \{i, \ldots, P\}$, the node w_{t-P+j} is active at round $t-P+j$ and

$$\begin{array}{ll} & c_{w_{t-P+j}}^{pre}(t-P+j) \equiv_P j \\ \wedge & j > 0 \Rightarrow synch_{w_{t-P+j}}^{pre}(t-P+j) \\ \wedge & j > 0 \wedge ready_v^{pre}(t) \Rightarrow ready_{w_{t-P+j}}^{pre}(t-P+j). \end{array}$$

1. The base case (i.e., $j = P$ and $w_{t-P+j} = v$) comes from lines 19, 20, and 27.

2. For the inductive case, we assume that $c_{w_{t-P+j+1}}^{pre}(t-P+j+1) \equiv_P j+1$ as well as $synch_{w_{t-P+j+1}}^{pre}(t-P+j+1)$, and that whenever $ready_v^{pre}(t)$ holds, then $ready_{w_{t-P+j+1}}^{pre}(t-P+j+1)$ holds as well. By Lemma 1.d, w_{t-P+j} is active in round $t-P+j$ and $c_{w_{t-P+j}}^{pre}(t-P+j) \equiv_P j$. If $j > 0$, we obtain $synch_{w_{t-P+j}}^{pre}(t-P+j)$ by Lemma 1.e, and by Lemma 1.f, $ready_v^{pre}(t)$ implies $ready_{w_{t-P+j}}^{pre}(t-P+j)$. ☐

Lemma 4. *If some node u reaches level 2 in round t_u, then γ is already in level 1 in round t_u.*

Proof. Let v be some node which reaches level 2 in round t_v. By Lemma 2, $t_v \geq P$. By line 27, we have $c_v^{pre}(t) \equiv_P 0 \wedge synch_v^{pre}(t) \wedge ready_v^{pre}(t)$. We consider a $\gamma \triangleright u$ path in the interval $[t_v - P+1, t_v]$, noted $w_{t_v-P}, w_{t_v-P+1}, \cdots w_{t_v}$. Applying Lemma 3 with $i = 1$ and $u = w_{t_v-P+1}$, we obtain that w_{t_v-P+1} is active in round $t_v - P + 1$ and $ready_{w_{t_v-P+1}}^{pre}(t_v - P + 1)$. Then $ready_\gamma(t_v - P)$ is true using line 16. Applying Lemma 3 with $i = 0$ and $u = \gamma$, we obtain that γ is active in round $t_v - P$ and $c_\gamma^{pre}(t_v - P) \equiv_P 0$. Finally, $level_\gamma(t_v - P) > 0$ using line 19 and 33. ☐

Lemma 5. *If γ reaches level 1 in round t_γ, no node can reach level 1 or 2 in any of the rounds $t_\gamma + 1, \ldots, t_\gamma + P - 1$.*

Proof. By Lemma 2, $t_\gamma \geq P$. We assume that some node u levels up in round $t_\gamma + i$ where $j \in \{1, \ldots, P-1\}$. Using $\mathbb{G} \in \mathcal{G}_\Delta^* \subseteq \mathcal{G}_P^*$, we have

$$\gamma \in In_u(t_\gamma - P + j + 1 : t_\gamma + j).$$

Applying Lemma 3.b with $i = 0$, we get $c_\gamma^{pre}(t_\gamma - P + j) \equiv_P 0$. The presence of the self-loops implies the existence of a $\gamma \triangleright \gamma$ path in the interval $[t_\gamma - P + j + 1, t_\gamma]$. Applying Lemma 3.b with $i = j$, we get $c_\gamma^{pre}(t_\gamma - P + j) \equiv_P j$. We get a contradiction from $j \equiv_P 0$. $\square$

Lemma 6. *Let u be some node, and t be some round in which u is active. There exists some node w which reached a level equal to $\mathit{force}_u^{pre}(t)$ in round $t - c_u^{pre}(t)$. Moreover, an active $w \triangleright u$ path exists in the interval $[t - c_u^{pre}(t) + 1, t]$.*

Proof. We show this lemma by induction on $c_u^{pre}(t)$.

1. As $c_u^{pre}(t) \geq 1$ by line 18, the induction begins at $c_u^{pre}(t) = 1$. In that case, u received a message $\langle 0, *, \mathit{force}_u^{pre}(t), * \rangle$ from some node v (see lines 17 and 18). Then v reached a level equal to $\mathit{force}_u^{pre}(t)$ in round $t-1$. Because $v \in \mathit{In}_u(t)$, an active $v \triangleright u$ path exists in the interval $[t, t]$.

2. Let us fix some $c_u^{pre}(t) > 1$. Then, u received a message $\langle c_u^{pre}(t) - 1, *, \mathit{force}_u^{pre}(t), * \rangle$ from some node v. From Lemma 1.b, $c_u^{pre}(t) - 1 = c_v^{pre}(t-1)$ and $\mathit{force}_u^{pre}(t) = \mathit{force}_v^{pre}(t-1)$. Applying the induction hypothesis to v in round $t - 1$, we obtain some node w which reaches a level equal to $\mathit{force}_u^{pre}(t)$ in round $t - c_u^{pre}(t)$. We also obtain an active $w \triangleright u$ path in the interval $[t - c_u^{pre}(t) + 1, t]$. $\square$

We define the set

$$Z \overset{\text{def}}{=} \{(f, t), \exists u \in V, \mathit{level}_u(t) = f \wedge \mathit{level}_u(t-1) \neq f\}. \tag{1}$$

This set represents the finite set of level-up events. Using Lemma 6, any node u satisfies $z_u(t) = (\mathit{force}_u^{pre}(t), t - c_u^{pre}(t)) \in Z$ in every round $t \geq s_u$ in which u is active. We order Z lexicographically. The following lemma proves that u records the most recent strongest level-up event of its view.

Lemma 7. *If there exists an active $u \triangleright v$ path between two nodes u and v in the interval $[t + 1, t']$, then $z_u(t) \leq z_v(t')$. Moreover, if u reached a level equal to f in round t, then $(f, t) \leq z_v(t')$.*

Proof. Using claims 1.g and 1.h, we have, for any integer t and any node w:

$$z_w(t+1) \geq \max_{w' \in \mathit{In}_w^a(t+1)} z_{w'}(t). \tag{2}$$

Given an active $u \triangleright v$ path between u and v in the interval $[t + 1, t']$, the main claim of the lemma follows from Eq. 2, applied to each node in this path. In the special case where u reached a level equal to f in round t, any outgoing neighbor w of u satisfies

$$z_w(t+1) \geq (f, t).$$

This inequality also comes from claims 1.g and 1.h. By Eq. 2, we obtain that $(f, t) \leq z_v(t')$, as required. $\square$

Lemma 8. *If γ reaches level 1 in some round t_γ, whereas some u reaches level 1 or 2 in some round $t_u \geq t_\gamma$, then $t_u \equiv_P t_\gamma$.*

Proof. By contradiction, we consider the earliest node u which levels up in some round $t_u \geq t_\gamma$ with $t_u \not\equiv_P t_\gamma$. By Lemma 5, $t_u \geq t_\gamma + P$. There exists a $\gamma \triangleright u$ path in the interval $[t_u - P + 1, t_u]$, and this path is active by Lemma 3.a. Using Lemma 7, the self-loop of γ, and this active path, we obtain

$$(1, t_\gamma) \leq z_\gamma(t_u - P) \leq z_u(t_u). \tag{3}$$

Lemma 6 implies the existence of a node v which reached a level equal to $\mathit{force}_u^{pre}(t_u)$ in some round $t_v = t_u - c_u^{pre}(t_u)$. In the case $\mathit{force}_u^{pre}(t_u) = 2$, from Lemma 4, we obtain $t_v \geq t_\gamma$. Otherwise, using $(1, t_\gamma) \leq z_u(t_u)$, we also have $t_v \geq t_\gamma$.

By line 19, we have $c_u^{pre}(t_u) \equiv_P 0$. Recalling $t_v = t_u - c_u^{pre}(t_u)$, we obtain $t_v \equiv_P t_u \not\equiv_P t_\gamma$. This contradicts the fact that u was the earliest node satisfying $t_u \geq t_\gamma$ and $t_u \not\equiv_P t_\gamma$. □

Lemma 9. *If every node is active in round t, and $z_\gamma(t) = z_\gamma(t + 3P)$, then γ is in level 1 in round $t + 3P$.*

Proof. Let t^0 be a round in which every node is active. By Lemma 7, the sequence $(z_\gamma(t))_{t>s(\gamma)}$ is non-decreasing. By assumption, it remains constant between the rounds t and $t + 3P$. Then, there exists some round $t^1 \in \{t^0, t^0 + 1, \dots t^0 + P - 1\}$ such that $c_\gamma^{pre}(t^1) \equiv_P 0$. Then we prove by induction on i the following invariant:

$$\forall i < P, \forall u \in \mathit{In}_\gamma(t^1 + P + i + 1 : t^1 + 2P),\ c_u(t^1 + P + i) \equiv_P i \quad \text{and } \mathit{synch}_u(t^1 + P + i) \text{ holds}$$

1. Base case: we fix some node $u \in \mathit{In}_\gamma(t^1 + P + 1 : t^1 + 2P)$. By Lemma 7, $z_\gamma(t^1) \geq z_u(t^1 + P) \geq z_\gamma(t^1 + 2P)$. Moreover, by assumption, $z_\gamma(t^1) = z_\gamma(t^1+2P)$. Using successively Claim 1.c, the equality $z_\gamma(t^1) = z_\gamma(t^1+2P)$ and the definition of t^1, we obtain
$$c_u(t^1 + P) \equiv_P c_u^{pre}(t^1 + P) \equiv_P c_\gamma^{pre}(t^1) \equiv_P 0. \tag{4}$$
Moreover, $\mathit{synch}_u(t^1 + P)$ holds by line 32.

2. Induction case: we fix some integer $i < P - 1$ and some node u such that $u \in \mathit{In}_\gamma(t^1 + P + i + 2 : t^1 + 2P)$. In round $t^1 + P + i + 1$, every incoming neighbor v of u belongs to $\mathit{In}_\gamma(t^1 + P + i + 1 : t^1 + 2P)$, and hence, by induction hypothesis, satisfies $c_v \equiv_P i$ and synch_v in round $t^1 + P + i$. By lines 18 and 11, $c_u(t^1 + P + i + 1) \equiv_P i + 1$ and $\mathit{synch}_u(t^1 + P + i + 1)$ holds, as required.

Finally, by choosing $i = P - 1$, the previous invariant, lines 18 and 11 imply that $c_\gamma^{pre}(t^1 + 2P) \equiv_P 0$ and $\mathit{synch}_\gamma^{pre}(t^1 + 2P)$ is true. By lines 19 and 20, γ moves to level 1 in round $t^1 + 2P$ at the latest. □

Lemma 10. *Assuming a dynamic graph satisfying* $\mathrm{rad}(\mathbb{G}) \leq P$*, any execution of the SynchMod*$_P$ *algorithm satisfies* mod P*-simultaneity.*

Proof. We fix some node u, and we assume that u reaches level 2 in round t_u. Let γ be a central node whose eccentricity is at most P. From Lemma 4, we obtain $t_u \geq t_\gamma$, where t_γ is the round in which γ reaches level 1. By Lemma 8, $t_u \equiv_P t_\gamma$. That proves mod P-simultaneity. □

Lemma 11. *Under the assumptions of a complete activation schedule and of a dynamic graph satisfying* $\mathrm{rad}(\mathbb{G}) \leq P$*, any execution of the SynchMod*$_P$ *algorithm terminates. Moreover, every node fires* $6nP$ *rounds after the activation of all nodes, at the latest, where* n *is the cardinality of* V*.*

Proof. Recall that s^{max} denotes the round from which every node is active. Let γ be a central node whose eccentricity is at most P. Let t_γ be the round, if any, in which γ moves to level 1. The proof consists in two parts: First, we show that t_γ exists and is bounded by $t^{max} = s^{max} + 3P(2n-1)$. Then we deduce the termination property of the *SynchMod*$_P$ algorithm.

We assume by contradiction that γ is still at level 0 in round t^{max}. By Lemma 4, each node other than γ is at most at level 1 in round t^{max}. Then, at most $2n-1$ level-up events occurred in round t^{max} and beforehand. We consider the sequence $(\hat{z}_i)_{i\in\mathbb{N}}$ where $\hat{z}_i = z_\gamma(s^{max} + 3P \times i)$. This sequence is non-decreasing by Lemma 7. In addition, by Lemma 9, this sequence is strictly increasing as long as γ is at level 0. We obtain a contradiction by the pigeonhole principle: the first $2n$ elements of the sequence $(\hat{z}_i)_{i\in\mathbb{N}}$ are distinct, whereas only $2n-1$ level-up events happened before round t^{max}. This ends the first part of the proof.

We achieve the second part of the proof using two invariants. First, we prove the following invariant by induction over i.

$$\forall i \in \mathbb{N}, \forall u \in V, c_u(t_\gamma + P + i) \equiv_P i \text{ and } \mathit{synch}_u(t_\gamma + P + i) \text{ holds} \tag{5}$$

1. Base case. By Claim 1.c and Lemma 8, as $z_u(t_\gamma + P) \geq z_\gamma(t_\gamma)$, we obtain
$$c_u(t_\gamma + P) \equiv_P c_u^{pre}(t_\gamma + P) \equiv_P c_\gamma^{pre}(t_\gamma) \equiv_P 0.$$
Moreover, $\mathit{synch}_u(t_\gamma + P)$ holds by line 32.

2. The inductive case holds by lines 18 and 11, using the induction hypothesis.

Given a node u, we apply Eq. 5 to each of u's incoming neighbors in round $t_\gamma + 2P$. We obtain $c_u^{pre}(t_\gamma + 2P) \equiv_P 0$ and $\mathit{synch}_u^{pre}(t_\gamma + 2P)$, and hence u reaches level 1 in round $t_\gamma + 2P$ at the latest. We prove another invariant by induction over i.

$$\forall i \in \mathbb{N}, \forall u \in V, \mathit{ready}_u(t_\gamma + 2P + i) \text{ holds} \tag{6}$$

The base case holds by line 33, and the inductive case holds by line 16. Finally, using Eq. 5 and 6, every node fires in round $t_\gamma + 3P \leq s^{max} + 6Pn$ at the latest. $\square$

The previous two lemmas yield the following correctness theorem:

Theorem 12. *Under the assumptions of a complete activation schedule and of a dynamic graph satisfying* $\mathrm{rad}(\mathbb{G}) \leq P$*, the SynchMod*$_P$ *algorithm solves the* $\mathrm{mod}\, P$*-synchronization problem for any integer* P *greater than 2. Moreover, every node fires* $6nP$ *rounds after the activation of all nodes, at the latest.*

3.4. Solvability Results

We show that the $\mathrm{mod}\, P$-synchronization problem is always solvable, regardless of the value of P, if the bound Δ on the delay is known: for each possible Δ, we can exhibit an algorithm which solves $\mathrm{mod}\, P$-synchronization in any dynamic graph satisfying $\mathrm{rad}(\mathbb{G}) \leq \Delta$.

Corollary 13. *For any positive integer* P*, the* $\mathrm{mod}\, P$*-synchronization problem is solvable in each network model* $\mathcal{G}^*_\Delta$ *in any complete activation schedule.*

Proof. Depending on the relative values of P and Δ, we consider the following cases:

1. $P = 1$. The $\mathrm{mod}\, P$-simultaneity property is a tautology in this case. The problem is trivially solvable in any network model, in particular $\mathcal{G}^*_\Delta$.

2. $\Delta \leq P$ and $P > 2$. By Theorem 12, the *SynchMod*$_P$ algorithm solves the $\mathrm{mod}\, P$-synchronization problem in $\mathcal{G}^*_\Delta$.

3. $\Delta \leq P = 2$. Theorem 12 shows that the *SynchMod*$_4$ algorithm achieves $\mathrm{mod}\, 4$-synchronization in $\mathcal{G}^*_2$, and hence achieves $\mathrm{mod}\, 2$-synchronization in $\mathcal{G}^*_2$.

4. $\Delta > P$. We have $\Delta \leq \lceil \frac{\Delta}{P} \rceil \cdot P$. By Theorem 12, the $\mathrm{mod}\, \lceil \frac{\Delta}{P} \rceil \cdot P$-synchronization problem is solvable in $\mathcal{G}^*_\Delta$ using *SynchMod*$_{\lceil \frac{\Delta}{P} \rceil \cdot P}$. The $\mathrm{mod}\, P$-synchronization problem is also solvable in $\mathcal{G}^*_\Delta$, *a fortiori.* $\square$

In contrast, we show that the $\mathrm{mod}\, P$-synchronization problem is not solvable if the delay Δ is unknown to the nodes.

Theorem 14. *If* $P > 1$*, then the* $\mathrm{mod}\, P$*-synchronization problem is not solvable in the network model* $\bigcup_{i \in \mathbb{N}} \mathcal{G}^*_i$*.*

Proof. By contradiction, assume that an algorithm A solves the problem in the above-mentioned network model. We consider any system and we fix two nodes u and v in this system. We denote I the digraph only containing self-loops. We denote C_u and C_v the digraphs only containing self-loops and a star centered in u and v respectively. We construct four executions of A:

1. Every node starts in round 1. The dynamic graph is equal to C_u at each round. This dynamic graph belongs to $\mathcal{G}_1^*$. Using the termination of A, u fires in some round f_u.

2. Every node starts in round 1. The dynamic graph is equal to C_v at each round. This dynamic graph belongs to $\mathcal{G}_1^*$. Using the termination of A, v fires in some round f_v.

3. Every node starts in round 1. During the first f_u+f_v rounds, the communication graph is equal to I. In every subsequent round, the communication graph is equal to C_u. This dynamic graph belongs to $\mathcal{G}^*_{1+f_u+f_v}$.

4. The node u starts in round 1, whereas every other node starts in round 2. During the first f_u+f_v rounds, the communication graph is equal to I. In every subsequent round, the communication graph is equal to C_u. This dynamic graph belongs to $\mathcal{G}^*_{1+f_u+f_v}$.

From the point of view of u, the third execution is indistinguishable from the first execution. Therefore, u fires in round f_u in the third execution. From the point of view of v, the third execution is indistinguishable from the second execution during the first f_v rounds. Thus, v fires in round f_v in the third execution. Using the mod P-simultaneity of A in the third execution, we obtain:

$$f_u \equiv_P f_v.$$

Similarly, u fires in round f_u and v fires in round $1+f_v$ in the fourth execution. Using the mod P-simultaneity of A in the fourth execution, we obtain:

$$f_u \equiv_P f_v + 1.$$

Since we assumed $P > 1$, a contradiction is reached. □

3.5. Reducing Memory Usage

For all nodes u and all rounds t, we have $(\mathit{force}_u^{pre}(t), t - c_u^{pre}(t)) \in Z$ by Lemma 6. Since Z is finite, $c_u^{pre}(t)$ tends to infinity as t tends to infinity. We present below an idea (inspired by [4]) which can alleviate this issue: in each execution of Algorithm 1, total memory usage increases forever, whereas in each execution of Algorithm 2, total memory usage grows during some arbitrarily long initial period, and then drops and remains bounded forever.

The idea is as follows: as soon as $\mathit{force}_u(t) = 2$, the node u knows that some node v fired in round $t - c_u(t)$ (see Lemma 6). Then u may fire in any round $t' \equiv_P t - c_u(t)$. At this point, the transition function can thus be simplified as in Algorithm 2.

Theorem 15. *Under the assumptions of a complete activation schedule and of a dynamic graph satisfying* $\mathrm{rad}(\mathbb{G}) \leq P$*, Algorithm 2 solves the* mod P*-synchronization problem. Moreover, in each execution of Algorithm 2, the memory usage of each node is bounded.*

Algorithm 2: The *OptSynchMod*$_P$ algorithm

Initialization:
  initialize with *SynchMod*$_P$'s initial state

At each round:
  if $force_u = 2$ **then**
    send $\langle c_u, true, 2, true \rangle$ to all
    $c_u \leftarrow 1 + c_u \bmod P$
    if $level_u < 2 \wedge c_u = 0$ **then**
      $level_u \leftarrow 2$
    end
  end
  else
    apply *SynchMod*$_P$'s transition function
  end

However, it is not possible to provide a bound on the memory usage that would hold for all executions of Algorithm 2. Indeed, the synchronization is guaranteed to happen only once all nodes have become active. Before this point, the memory usage of active nodes will steadily grow.

4. Conclusion and Future Work

In this paper, we introduced the mod P-synchronization problem, and we presented an algorithm for solving this problem. We provided a detailed proof of correctness of this algorithm, and we showed that the time complexity is bounded by $6Pn$, where n is the number of nodes in the system. We did not prove the tightness of this bound, but we believe that an execution whose time complexity is close to $6Pn$ exists. This could be part of a future work. This time complexity seems problematic when n is large. Typically, IoT networks can contain a few hundred nodes. However, keep in mind that $6Pn$ is the time complexity of the worst-case scenario. We believe that, in the average case, the time complexity is much better, including in large networks. This issue could also be further studied in the future.

References

[1] Arora, A., Dolev, S., Gouda, M.G.: Maintaining digital clocks in step. Parallel Processing Letters **1**, 11–18 (1991)

[2] Awerbuch, B., Kutten, S., Mansour, Y., Patt-Shamir, B., Varghese, G.: A time-optimal self-stabilizing synchronizer using a phase clock. IEEE Transactions on Dependable and Secure Computing **4**(3), 180–190 (2007)

[3] Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)

[4] Boldi, P., Vigna, S.: Universal dynamic synchronous self–stabilization. Distributed Computing **15**(3), 137–153 (2002)

[5] Boulinier, C., Petit, F., Villain, V.: Synchronous vs. asynchronous unison. Algorithmica **51**(1), 61–80 (2008)

[6] Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. In: International Conference on Ad-Hoc Networks and Wireless. Lecture Notes in Computer Science, vol. 6811, pp. 346–359. Springer (2011)

[7] Charron-Bost, B., Moran, S.: The firing squad problem revisited. Theoretical Computer Science **793**, 100–112 (2019)

[8] Charron-Bost, B., Moran, S.: MinMax algorithms for stabilizing consensus. Distributed Computing **34**, 195–206 (2021)

[9] Charron-Bost, B., Schiper, A.: The Heard-Of model: computing in distributed systems with benign faults. Distributed Computing **22**(1), 49–71 (2009)

[10] Couvreur, J.M., Francez, N., Gouda, M.G.: Asynchronous unison. In: ICDCS. vol. 92, pp. 486–493 (1992)

[11] Debrat, H., Merz, S.: Verifying fault-tolerant distributed algorithms in the heard-of model. Archive of Formal Proofs (2012), http://isa-afp.org/entries/Heard_Of.html, Formal proof development

[12] Dolev, D., Strong, H.R.: Authenticated algorithms for Byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)

[13] Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)

[14] Herman, T., Ghosh, S.: Stabilizing phase-clocks. Inf. Process. Lett. **54**(5), 259–265 (1995)

[15] Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: Proceedings of the 42nd ACM Symposium on Theory of Computing. pp. 513–522 (2010)

[16] Kuhn, F., Moses, Y., Oshman, R.: Coordinated consensus in dynamic networks. In: Proceedings 30th ACM Symposium Principles of Distributed Computing. pp. 1–10. ACM (2011)

[17] Lamport, L.: The part-time parliament. ACM Transactions on Computer Systems **16**(2), 133–169 (May 1998)

[18] Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL. A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)

[19] Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)

[20] Penet de Monterno, L., Charron-Bost, B., Merz, S.: Synchronization modulo k in dynamic networks. In: 23rd International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS. Lecture Notes on Computer Science, vol. 13046, pp. 425–439. Springer (2021)

[21] Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distributed Computing **2**(2), 80–94 (1987)

Appendix A. Some Extra Formal Definition

Below are the formal definitions of the state space and the initial state of the $SynchMod_P$ algorithm in the syntax of Isabelle/HOL.

record *locState* =
  x :: nat
  synch :: bool
  ready :: bool
  force :: nat — *force* $\in \{0, 1, 2\}$
  level :: nat — *level* $\in \{0, 1, 2\}$
definition *initState* **where**
  initState ≡ (| *x = 0, synch = False, ready = False, force = 0, level = 0* |)

We define a datatype for messages sent between two nodes u and v: messages either carry a value of some type *'msg*, or are equal to *Null* if u is passive, or to *Void* if u is not an incoming neighbor of v.
datatype *'msg message = Content 'msg | Null | Void*

Then we provide the transition function of the $SynchMod_P$ algorithm. The argument *msgs* is a function that maps each node to the message received from this node.

definition *nextState :: nat ⇒ locState ⇒ (Proc ⇒ locState message) ⇒ locState* **where**
  nextState P s msgs ≡
    let *synch_pre = (∀ p. msgs p ≠ Void ⟶*
      (∃ m. msgs p = Content m ∧ synch m ∧ c m mod P = c s mod P)) **in**
    let *ready_pre = (∀ p m. msgs p = Content m ⟶ ready m)* **in**
    let *force_pre = (Max (P_mod.forceMsgs ‘ range msgs))* **in**
    let *c_pre = Suc (LEAST v. ∃ m p. msgs p = Content m ∧ force m = force_pre ∧ c m = v)* **in**
    if *c_pre mod P = 0* **then**
      if *level s = 0 ∧ synch_pre* **then**
        if *force_pre ≤ 1* **then**

```
        ⦇ c = 0, synch = True, ready = True,
          force = 1, level = 1 ⦈
      else
        ⦇ c = c_pre, synch = True, ready = True,
          force = force_pre, level = 1 ⦈
    else
    if level s = 1 ∧ synch_pre ∧ ready_pre then
      ⦇ c = 0, synch = True, ready = True,
        force = 2, level = 2 ⦈
    else
      ⦇ c = c_pre, synch = True, ready = level s > 0,
        force = force_pre, level = level s ⦈
  else
    ⦇ c = c_pre, synch = synch_pre, ready = ready_pre,
      force = force_pre, level = level s ⦈
```

The following two definitions state the correctness properties of the algorithm. In these definitions, *rho* denotes an execution, modeled as a sequence of global states, i.e. functions from nodes u to either *Asleep* (representing the fact that node u is still passive) or *Active* s for some local state s.

```
definition liveness where      — termination
  liveness rho ≡ ∀ u. ∃ t s. rho t u = Active s ∧ level s = 2
definition safety where      — mod P-simultaneity
  safety rho ≡ ∃ c. ∀ u t s ss.
    rho t u = Active s ⟶ level s < 2 ⟶
    rho (Suc t) u = Active ss ⟶ level ss = 2 ⟶ t mod P = c
```

The correctness of the $SynchMod_P$ algorithm is proved under the following assumptions:

```
assumes ∀ u t. path In gamma u t P       — gamma's eccentricity is at most P
  and ∀ u t. u ∈ In t u                  — the graph contains self-loops
  and HORun (HOMachine P) rho In         — rho is an execution
  and ∀ p. ∃ t. rho t p ≠ Asleep         — the schedule is complete
  and P > 2
```

The predicate *HORun* above is defined in [11] and characterizes executions of an algorithm as described above. Since this definition was originally written for synchronous starts, we adapted it to describe asynchronous starts.