Synchronization Modulo k in Dynamic Networks

Louis Penet de Monterno, Bernadette Charron-Bost, Stephan Merz

The mod k-synchronization Problem

Louis Penet de Monterno - Bernadette Charron-Bost

August 2020

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 classically assumed that each run of an algorithm is started by all nodes simultaneously, i.e., at the same round, or even at round one. For instance, most of synchronous consensus algorithms (e.g., [**PSL80**, **DS83**, **ST87**]), as well as many distributed algorithms for dynamic networks (e.g., [**KLO10**, **KMO11**]) require synchronous starts.

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 more 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 become *active* at an unpredictable round. The goal is to guarantee that the nodes, when 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 [**CBM18**] demonstrates that the firing squad problem is not solvable without a strong connectivity property of the network, namely, there exists some positive integer T such that communication graph within every period of T consecutive rounds is strongly connected and the delay T is known. In many situations, this connectivity property is not guaranteed: as an example, in the dynamic graphs corresponding to any system model with benign failures, a node that experiments permanent and complete send omissions is constantly a sink in the communication graph.

However, with a closer look at many distributed algorithms designed in the round-based model, we see that these algorithms actually do not require perfect 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 k, for some positive integer k. The corresponding synchronization problem, that we call mod k *synchronization*, is formally specified as follows:

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

mod **k-simultaneity:** If two nodes fire at round t and t', then $t' \equiv t \mod k$.

Indeed, let A be an algorithm organized into regular *phases* consisting of a fixed number k of consecutive rounds: the sending and transition functions of every node at round t are entirely determined by the value of t modulo k. Moreover, assume that A has been proved correct (with respect to some given specification) when all nodes start A synchronously (at round one) and with any dynamic graph in a family $\mathcal{G}$ that is stable under addition of arbitrary finite prefixes. For instance, the *ThreePhaseCommit* algorithm for non-blocking atomic commitment [**BHG**], as well as the consensus algorithms in [**DLS87**] or the *LastVoting* algorithm [**CBS09**] – corresponding to the consensus core of *Paxos* – fulfill all the above requirements for phases of length $k = 3$ and $k = 4$, respectively, and the family $\mathcal{G}$ of dynamic graphs in which there exists an infinite number of "good" communication patterns (eg., a sequence of $2k - 1$ consecutive communication graphs in which a majority of nodes is heard by all in each graph). The use of a mod k-synchronization algorithm on the top of 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 perfect synchronization can be weakening into synchronization modulo k is the development of the basic *rotating coordinator* strategy in the context of asynchronous starts. Roughly speaking, this strategy consists in the following: if nodes have unique identifiers in $\{1, \ldots, n\}$, the coordinator at round t is the node whose identifier is t modulo n. For that, each node u maintains a local counter c_u whose current value is the number of rounds where it has been active. 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 synchronous starts. Clearly, with the use of a mod n-synchronization algorithm in a preliminary phase and a counter for each node that now counts the number of rounds elapsed since the node fired, the above scheme correctly[1] implements the rotating coordinator strategy in the context of asynchronous starts,.

As a basic synchronization abstraction, a mod k-synchronization algorithm can be used as a subroutine by some parent algorithm. For instance, consider the use of a mod k-synchronization algorithm on the top of some algorithm $\mathcal{B}$ that has been proved correct (with respect to some given specification) in the case of synchronous starts

In this case, a node is passive if it has not yet completed the execution of $\mathcal{A}$, and the start signal on a node corresponds to the completion of $\mathcal{A}$ by this node. In this example a passive node sends null messages, just indicating that it has not completed the execution of $\mathcal{A}$.

To cope with this problem, we propose to

obtained when implementing synchronous rounds

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 [**CBS09**], in which point-to-point communications are organized into *synchronized rounds*: each node can send messages to all nodes and can receive messages sent by some of the nodes. Rounds are communication closed in the sense that no node receives messages in round t that are sent in a round different from t. The collection of *possible* communications (which nodes can communicate to 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 $\mathbb{G}(t) = (V, E_t)$, 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 $\mathrm{In}_u(t)$.

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 $\mathbb{G} = (\mathbb{G}(t))_{t\in\mathbb{N}}$ is called a *dynamic graph* [**CFQS12:TVG**].

In round t ($t = 1, 2, \ldots$), each node u successively (a) broadcasts messages determined by its state at the beginning of round t (b) receives *some* of the messages sent to it, and finally (c) undergoes an internal transition to a new state. A *local algorithm* for a node corresponds to a pair of a *sending function* that determines the 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* that are collections $\mathbb{S} = (s_u)_{u\in V}$, where each s_u is a positive integer or is equal to ∞.

The execution of the 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, sets up its local variables. In round t ($t = 1, 2 \ldots$), 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 message to be sent to all nodes, 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 go to a 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,

[1] With respect to a specification that lets a passive node to be a coordinator.

the node is integrally part of the system and can send messages, but these messages are empty with respect to the semantics of B.

2.2 Network model and start model

A *network model* is any non-empty set of dynamic graphs with a permanent self-loop at each node. We will focus on the specific network model of *centered* dynamic graphs $\mathbb{G}$ defined as follows: there exists some node γ in V such that every digraph $\mathbb{G}(t)$ has a spanning star centered at γ, i.e.,

$$\exists \gamma \in V, \forall u \in V, \forall t \in \mathbb{N},\ \gamma \in \mathrm{In}_u(t).$$

Note that there may be several fixed centers for $\mathbb{G}$.

As demonstrated in [**CBS09**], centered dynamic graphs in the Heard-Of model captures the classical model of synchronous systems with a (fixed) complete communication graph and at most $n-1$ faulty senders, including the case of crashes.

Similarly, we 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, yielding the model of complete start schedules. Synchronous starts correspond to complete start schedules $\mathbb{S} = (s_u)_{u \in V}$ with equal start rounds. The property of synchronous starts can be relaxed into mod k*-synchronous starts*, where k is any positive integer: for every pair of nodes u and v, it holds that $s_u \equiv s_v \mod k$.

3 The *SynchMod*$_k$ algorithm

In the *SynchMod*$_k$ algorithm, each node maintains a local clock modulo k with values in $\{\overline{1}, \ldots, \overline{k}\}$. It fires in the first round at which all the local clocks it has just heard of are all equal to $\overline{k}$ (line 11). The first time a node receives discrepant clocks from its neighbors, it tries to force firing by setting its clock to $\overline{k}$ (lines 16-17); thereafter, that just leads it to roll back its clock to 1 (line 19). Otherwise, it receives agreed values with its own clock and then increments it by one modulo k (line 13). Let us again stress on the fact that at each round t, every active node receives the value of its local clock. The pseudo-code of the local code of the agent u is given in Algorithm 1.

As we will see below, the difficult point in the correctness proof of the *SynchMod*$_k$ algorithm is liveness. However, right now, let us point out some properties of the algorithm that enable liveness. First, if all the nodes agree on the same value for their local clocks – in which case the system will be said to be *monovalent* – and if they are all active, then the system remains monovalent forever. Moreover, the common value of the local clocks is incremented by one at every later round and thus eventually reaches the value $\overline{k}$ (cf. Lemma 3.4). The key point of the algorithm and of its "forced firing procedure" lies in the fact that if all the communication graphs contain a star centered at γ, then when γ is active, its local clock necessarily becomes equal to $\overline{k}$, and every active node will eventually fire.

3.1 Notation and preliminary lemmas

In the rest of this section, we fix an execution σ of the *SynchMod*$_k$ algorithm associated to a complete activation schedule $\mathcal{A}$ and a centered dynamic graph $\mathbb{G} \in \mathcal{G}^c$. Let $s^{\max} = \max_{u \in \Pi} s(u) < \infty$ and let γ denote one center of $\mathbb{G}$.

For the correctness proof of *SynchMod*$_k$, we now introduce some additional definitions. Let S be any subset of $\mathbb{Z}/k\mathbb{Z}$. Round t in σ is said to be S*-valent* if S is the set of the clock values of active nodes at the end of round t, i.e.,

$$S = \{\overline{i} \in \mathbb{Z}/k\mathbb{Z} : \exists u \in \mathcal{A}_t,\ \overline{c}_u(t) = \overline{i}\,\}.$$

The system is said to be $\overline{i}$*-monovalent* if the system is $\{\overline{i}\}$-valent.

Let us define $\phi(u)$ to be the round number at which the node u fires, if any, and let $\phi(u) = \infty$ otherwise. Similarly, let us define $\tau(u)$ to be the round number at which the node u tries to force firing (line 16) if any, and let $\tau(u) = 0$ otherwise. It follows that $\tau^{\max} = \max_{u \in \Pi} \tau(u) < \infty$.

In the *SynchMod*$_k$ algorithm, the state of each node u is composed of three variables named $\overline{c}_u$, $fired_u$ and $tried_u$. For any round r, we denote $\overline{c}_u(r)$, $fired_u(r)$ and $tried_u(r)$ respectively the values of these variables in the execution σ.

Lemma 3.1. *If $\overline{c}_\gamma(t) = \overline{k}$, then the round $t+1$ is $\overline{1}$-monovalent.*

Algorithm 1 The $SynchMod_k$ algorithm

1: **Initialization:**
2: $\overline{c}_u \in \mathbb{Z}/k\mathbb{Z} \cup \{\perp\}$, initially $\perp$
3: $tried_u \leftarrow false$
4: $fired_u \leftarrow false$

5: **In each round** t**:**
6: send $\langle \overline{c}_u \rangle$ to all
7: receive incoming messages
8: **if** all the received messages are equal to $\overline{k}$ and $\neg fired_u$ **then**
9: $fired_u \leftarrow true$
10: **if** the received messages other than *null* and $\perp$ are all equal to $i \in \{\overline{1}, \ldots, \overline{k}\}$ **then**
11: $\overline{c}_u \leftarrow \overline{i+1}$
12: **else**
13: **if** $\neg tried_u$ and no received message is $\overline{k}$ **then**
14: $\overline{c}_u \leftarrow \overline{k}$
15: $tried_u \leftarrow true$
16: **else**
17: $\overline{c}_u \leftarrow \overline{1}$

Proof. If $\overline{c}_\gamma(t) = \overline{k}$, then the node γ sends $\overline{k}$ to all nodes in round $t+1$. Hence, any active node u at round $t+1$ receives $\overline{k}$ in this round, and so updates its clock $\overline{c}_u$ according either to line 13 of to line 19. In both cases, it holds that $\overline{c}_u(t+1) = \overline{1}$. □

Lemma 3.2. *If the center γ is active in round t and round t is $\overline{i}$-monovalent, then any subsequent round $t+h$ is $\overline{i+h}$-monovalent.*

Proof. The proof is by induction on $h \in \mathbb{N}$.

1. The base case $s = 0$ corresponds to the assumption in the lemma.
2. Induction step: assume that the round $t+h$ is $\overline{i+h}$-monovalent, and let u be any active node in round $t+h+1$. The center γ is active in round $t+h$, and thus sends the value $\overline{i+h}$ to u in round $t+h+1$. Therefore, the node u can receive only this value (in addition of null and $\perp$), and thus updates $\overline{c}_u$ according to line 13. It follows that $\overline{c}_u(t+h+1) = \overline{i+h+1}$ as required.

□

Lemma 3.3. *If the center γ is active at round t and $\overline{c}_\gamma(t) = \overline{k}$, then every node fires no later than round* $\max(t, s^{\max}) + k$.

Proof. Lemmas 3.1 and 3.2 show that the system is $\overline{1}$-monovalent in round $t+1$ and $\overline{i}$-monovalent in every following round $t+i$. It follows that there is a round r, with $r \in [\max(t, s^{\max}), \max(t, s^{\max}) + k - 1]$, that is $\overline{k}$-monovalent and at which every node is active. In round r, every node, including γ, sends $\overline{k}$. According to line 11, every node fires no later than round $r+1$. □

Lemma 3.4. *If round t is a monovalent round in which the center γ is active, then every node fires before round $max(t, s^{\max}) + k + 1$.*

Proof. Lemma 3.2 guarantees that for every non negative integer i, the round $t+i$ is monovalent. Hence, there exists some round r, with $r \in [max(t, s^{\max}), max(t, s^{\max}) + k - 1]$ that is $\overline{k}$-monovalent. In this round, every node, including γ, sends $\overline{k}$. According to line 11, every node fires no later than round $r+1$. □

Lemma 3.5. *Any round $t > \max(s(\gamma), \tau^{\max}) + 1$ is either monovalent or S-valent with $S = \{1, \overline{i}\}$.*

Proof. Let $t \geq \max(t_s(\gamma), \tau^{\max}) + 1$, and let u be an active node at round t. Since $t \geq s(\gamma) + 1$, we have $\overline{c}_\gamma(t-1) = \overline{i-1} \in \mathbb{Z}/k\mathbb{Z}$, and the node u receives the value $\overline{i-1}$ at round t. There are two cases to consider.

1. The node u receives no value else than $\overline{i-1}$ and $\perp$ at round t. Then u executes line 13, and $\overline{c}_u(t) = \overline{i}$, i.e., round t is $\overline{i}$-monovalent.

2. Otherwise, u executes line 19 since it has already tried to force firing ($t > \tau(u)$). Therefore, $\overline{c}_u(t) = \overline{1}$, which shows that round t is $\{1, \overline{i}\,\}$-valent.

□

3.2 Correctness proof

We are now in position to prove the correctness of the *SynchMod*$_k$ algorithm for any integer k greater than 2 under the conditions of a complete activation schedule and a centered dynamic graph. The safety property is a direct consequence of the above lemmas. For the liveness property, Lemmas 3.3 and 3.4 lead us to define the notion of a *good round* as either a monovalent round or a round in which the counter of any center reaches the value $\overline{k}$: liveness is then enforced by the existence of a good round.

Theorem 1. *Under the conditions of complete activation schedules and centered dynamic graphs, the* SynchMod$_k$ *algorithm solves the* mod k*-synchronization problem for any integer k greater than 2.*

Proof. Let α be an execution of the *SynchMod*$_k$ algorithm with a dynamic graph centered at γ and a complete activation schedule.

For the safety property, let us assume that some nodes fire in α, and let u be the first node that fires in round $\phi(u)$. Since γ is an incoming neighbor of u, the firing rule (line 11) implies that γ is active and has sent the value $\overline{k}$ in round $\phi(u)$. By Lemma 3.1, round $\phi(u)$ is $\overline{1}$-monovalent, and Lemma 3.2 shows that for every round $\phi(u) + i$ is $\overline{i}$-monovalent. If a node v fires in a later round $\phi(v) > \phi(u)$, the round $\phi(v) - 1$ is $\overline{k}$-monovalent (line 11); and thus $\phi(u)$ and $\phi(v)$ are congruent modulo k.

For the liveness property, it suffices to prove that α contains a good round. Let us first observe that if γ tries to force firing (line 17) at round t, then $\overline{c}_\gamma(t) = \overline{k}$ and round t is a good round. Thus let us assume $\tau(\gamma) = 0$, and let $t \geq \max(s(\gamma), \tau^{\max}) + 1$. Lemma 3.5 shows that either (1) round t is monovalent, or (2) $\overline{c}_\gamma(t) = \overline{i} \neq \overline{1}$ and round t is $\{\overline{1}, \overline{i}\,\}$-valent, or (3) $\overline{c}_\gamma(t) = \overline{1}$ and round t is $\{\overline{1}, \overline{i}\,\}$-valent.

1. In case (1), round t is a good round.

2. In case (2), the center γ only receives the value $\overline{c}_\gamma(t) = \overline{i}$ at round $t+1$ since it never tries to force firing. Hence, we have $\overline{c}_\gamma(t+1) = \overline{i+1}$, and round $t+1$ is $\{\overline{1}, \overline{i+1}\,\}$-valent. Repeating this argument yields $\overline{c}_\gamma(t+k-i) = \overline{k}$. Hence, round $t+k-i$ is a good round.

3. For case (3), we consider the following two subcases:

   (a) Node γ does not receive the value $\overline{i}$ at round $t+1$, and so $\overline{c}_\gamma(t+1) = \overline{2}$. Since $t > \tau^{\max}$, we have $\overline{c}_u(t+1) = \overline{1}$ or $\overline{c}_u(t+1) = \overline{2}$ for every node u. Moreover, round $t+1$ is $\{\overline{1}, \overline{2}\,\}$-valent and meets the above case (2). It follows that round $t+k-1$ is a good round.

   (b) Node γ receives the value $\overline{i}$ at round $t+1$. Since $\tau(\gamma) = 0$, this case may occur only if $\overline{i} = \overline{k}$. In this case, $\overline{c}_u(t+1) = \overline{1}$, and round $t+1$ is either $\{\overline{1}\}$-monovalent or $\{\overline{1}, \overline{2}\,\}$-valent. In the latter situation, round $t+1$ meets case (3) with $\overline{i} = \overline{2}$, and hence case (3.a) when $k > 2$. Therefore, either round $t+1$ or round $t+k$ is a good round.

It follows that if $k > 2$, then the execution α contains a good round, and Lemmas 3.3 and 3.4 imply the liveness property of the *SynchMod*$_k$ algorithm. □

3.3 The mod 2-synchronization

Unfortunately, the *SynchMod*$_k$ algorithm does not work when $k = 2$, as demonstrated by its execution with three nodes and the two-periodic dynamic graph $G, H, G, H \cdots, G, H, \cdots$, where G is the communication graph in every even round, and H is the communication graphs in every odd round. G and H are given in Figure below. The starts schedule is defined by $\mathcal{A}_0 = \emptyset$, $\mathcal{A}_1 = \{u_1\}$ and $\mathcal{A}_i = \Pi$ for any $i > 1$. The system cycles between two configuration and no node ever fire. That violates the termination property.

However, the mod k-synchronization is trivially reducible to the mod k'-synchronization if k divides k'. Hence, the mod 2-synchronization is solvable with the *SynchMod*$_4$ algorithm in the class of dynamic graphs with a fixed center.

Then, a natural question is whether there exists a better and more direct algorithm for the mod 2-synchronization problem in the class of centered dynamic graphs, using a different algorithmic approach.

4 A possible optimisation

There exist some scenarios where the majority of nodes are active, and some of them have already fired. Because of some remaining passive node, some active nodes refrain from firing. In the worst case, a single passive node can prevent some active node from firing, during an arbitrary long time. In this section, we propose an optimisation which improves time-complexity in the average case. When a node fires, it learns that the system is monovalent. During the subsequent rounds, it tries to share this knowledge using gossip. A node knowing that the system is monovalent may fire as soon as its counters is equal to $\overline{1}$, in spite of the null messages it might receive.

Algorithm 2 The *SynchMod*$_k$ algorithm

1: **Initialization:**
2: $\overline{c}_u \in \mathbb{Z}/k\mathbb{Z} \cup \{\perp\}$, initially $\perp$
3: $tried_u \leftarrow false$
4: $fired_u \leftarrow false$
5: $monovalent_u \leftarrow false$

6: **In each round** t**:**
7: send $\langle \overline{c}_u, monovalent_u \rangle$ to all
8: receive incoming messages
9: $monovalent_u \leftarrow monovalent_u \vee$ all the received messages are equal to $\overline{k} \vee$ one true flag received
10: **if** $monovalent_u \wedge \overline{c}_u = k$ **then**
11: $fired_u \leftarrow true$
12: **if** the received messages other than *null* and $\perp$ are all equal to $i \in \{\overline{1}, \ldots, \overline{k}\}$ **then**
13: $\overline{c}_u \leftarrow \overline{i+1}$
14: **else**
15: **if** $\neg tried_u$ and no received message is $\overline{k}$ **then**
16: $\overline{c}_u \leftarrow \overline{k}$
17: $tried_u \leftarrow true$
18: **else**
19: $\overline{c}_u \leftarrow \overline{1}$

Theorem 2. *Under the conditions of complete activation schedules and centered dynamic graphs, the optimized* SynchMod$_k$ *algorithm solves the* mod k*-synchronization problem for any integer k greater than 2. Moreover, in any execution of the optimized algorithm, no node can fire later than in the corresponding execution of the non-optimized algorithm.*

The proof of the first part of this theorem is exactly the same as the proof of the non-optimized algorithm. Moreover, the proof of the second part results from a simple observation of the line 9.

5 Use-cases of mod n-synchronization

5.1 Algorithms without safety assumption

Let us consider the LastVoting algorithm, which is a round-based adaptation of Paxos. As pointed out in the introduction, this algorithm is incorrect with asynchronous starts, for structural reasons. A prior mod 4-synchronization can solve the main structural issue. The question is now how should the algorithm deal with null messages, and whether the algorithm is still correct with mod 4-synchronization instead of full synchronization.

The proof of liveness relies on an assumption on the communication graph. However, no assumption is required for safety, and that makes the above questions easy to answer. We define the $SynchMod_4 \Diamond LastVoting$ algorithm with the following construction:

A passive node sends null to all, until it gets active. Then it executes $SynchMod_k$ algorithm. It proceeds $SynchMod_k$ -related messages, and ignores LastVoting-related messages. If it fires in round r, then, in round $t+1$ it starts LastVoting algorithm in parallel with $SynchMod_k$. If a node u sends to v a message related to $SynchMod_k$ without message related to LastVoting, v learns that u has not fired yet. The LastVoting instance of v must interpret that absence of message as a null message from u. Without adaptation of LastVoting algorithm, the LastVoting instances proceeds null message as a message loss. With this construction, the execution of LastVoting is equivalent to an execution where the outgoing edges of passive nodes have been removed from the communication graphs.

Since the safety does not relies on any assumption on the dynamic graph, the safety proof written with synchronous starts is still valid with this construction. If we assume that no node remains passive forever, the liveness proof written with synchronous starts is still valid, at least from a round where every node is active. Thus, we can prove that the $SynchMod_4 \Diamond LastVoting$ algorithm solves the consensus problem in an asynchronous-starts-tolerant way.

The literature provides an algorithm called ThreePhaseCommit – in short 3PC – with solves the database commit problem. Using the same construction, we can prove that the $SynchMod_3 \Diamond 3PC$ solves the database commit problem in an asynchronous-starts-tolerant way.

5.2 Algorithms with safety assumption

When the safety proof of an algorithm relies on some assumption, the above-mentioned procedure may be incorrect. In that case,

5.3 mod k-synchronization and coordinated algorithms

Some algorithms like Paxos rely on the existence of a shared coordinator in each round. A simple implementation consists in setting a rotating coordinator: each node holds the list of nodes. In the first round, the chosen coordinator is the first node in the list. In the i^{th} round, the chosen coordinator is the $i \ mod \ k^{th}$ node on the list. This works out-of-the box when the starts are assumed to be synchronous. However, when the starts are asynchronous, a prior mod n-synchronization is required, where n is the number of nodes in the shared list. This is another typical use case of the $SynchMod_k$ algorithm.

6 Conclusion and future work

As any complex reasoning by cases, the correctness proof of the $SynchMod_k$ algorithm, and more specifically the proof of the liveness property, is very error prone. This is a typical example of the relevance of formal verification for distributed algorithms. Indeed, in a later work [], we used the interactive theorem prover Isabelle to encode the complete proof of Theorem 2, and thus obtained a certificate for $SynchMod_k$'s correctness when k is greater than 2.

Since mod 2-synchronization is reducible to mod 4-synchronization, our algorithm solves the mod k-synchronization problem for any positive integer k in the class of dynamic graphs with a fixed center. This class of dynamic graphs plays a crucial role regarding benign failures as it captures the synchronous model with at most $n-1$ faulty senders, including the one with at most $n-1$ crashes. In the wilder context of dynamic graphs, a natural question is whether the problem is still solvable under weaker connectivity assumptions, in particular, in the class of dynamic graphs with a fixed root, i.e., with a time-varying spanning tree at each round rooted at a fixed node.

7 Appendix: Non-uniform firing-squad algorithm

The goal of this section is to provide an adaptation of the firing-squad algorithm to solve the relaxed problem called *non-uniform synchronization*. A subset $S \subseteq \Pi$ is the subset of correct nodes. A correct node cannot crash, and any messages sent by $u \in S$ always reach its destination. An incorrect node $u \in \Pi \setminus S$ which crashes in round t must :

- be correct until round $t-1$. All of its outgoing messages reach their destination.
- fail in round t : only a subset of its outgoing messages reach their destination.
- be quiet in round $t+1$ and later.

We define the non-uniform synchronization problem as the conjunction of two properties:

Safety: If a correct node fires in round t, every correct node do so.

Liveness: At least one correct node fires.

We define below the firing-squad algorithm adapted for a system with at most f failures. In this algorithm, each node maintains a counter. When every correct node is active, no node can receive a null message, and the nodes start incrementing their counters. In the case where a node crashes in round s^{max}, there might be a discrepancy of one unit between the counter of the nodes. This discrepancy might be preserved if a node crashes in round $s^{max}+1$. However, the upper bond on the number of crash, and the sharing of counter values between nodes in each round guarantee that the counter values of every node will eventually be synchronized.

Algorithm 3 The firing-squad algorithm

```
1:  Initialization:
2:      i_u ∈ ℕ, initially 0

3:  In each round t:
4:      send i_u to all
5:      receive incoming messages
6:      if no null received then
7:          i_u ← 1 + min {i_v(t − 1), v ∈ HO(u, t)}
8:      else
9:          i_u ← 0
10:     if i_u ≥ f + 2 then
11:         fire
```

7.1 Proof

Lemma 7.1. *If no node crash during a round $r \geq s^{max}$, every correct node must hold the same counter value.*

Proof. In such a round r, every correct node sends to all its counter. Since no crash happen, the remaining node stays quiet. Then, every node receive the same sets of values. The line 7 guarantees that, at the end of the round, every node holds the same set. □

Lemma 7.2. *If at a round $r \geq s^{max}$, every correct node hold the same counter value, in every subsequent round, every node will hold the same counter value.*

Proof. This property can be proved by induction over the round number. Initially, we assume that every correct node hold the same counter value. In every subsequent round, the set of received values will be the same singleton for every node. Then, using the 7, the induction holds. □

Theorem 3. *The firing-squad algorithm solves the non-uniform synchronization problem if no node remains passive forever, and at most f crash happen.*

Proof. If no node remains passive forever, s^{max} is finite.

Safety: Since at most f crash may happen, there exists a round r between s^{max} and $s^{max} + f + 1$ in which no crash happen. Then, using lemmas 7.1 and 7.2, we obtain that, in round $s^{max} + f + 1$, every node holds the same counter value. Moreover, no node can reach $i_u \geq f + 2$ before round $s^{max} + f + 1$. That proves safety.

Liveness: Beyond round $s^{max} + 1$, no node can receive a null message. We can prove that, in round $s^{max} + k$, the value k is a lower bound of the counter values in the system. Thus, nodes must reach a counter value of $f + 2$. That proves liveness.

□