# Brief Announcement: Unifying Partial Synchrony

**Andrei Constantinescu**
ETH Zürich, Switzerland

**Diana Ghinea**
ETH Zürich, Switzerland

**Jakub Sliwinski**
ETH Zürich, Switzerland

**Roger Wattenhofer**
ETH Zürich, Switzerland

## Abstract

The distributed computing literature considers multiple options for modeling communication. Most simply, communication is categorized as either synchronous or asynchronous. Synchronous communication assumes that messages get delivered within a publicly known timeframe and that parties' clocks are synchronized. Asynchronous communication, on the other hand, only assumes that messages get delivered eventually. A more nuanced approach, or a middle ground between the two extremes, is given by the *partially synchronous model*, which is arguably the most realistic option. This model comes in two commonly considered flavors:

(i) The Global Stabilization Time (GST) model: after an (unknown) amount of time, the network becomes synchronous. This captures scenarios where network issues are transient.

(ii) The Unknown Latency (UL) model: the network is, in fact, synchronous, but the message delay bound is unknown.

This work formally establishes that any time-agnostic property that can be achieved by a protocol in the UL model can also be achieved by a (possibly different) protocol in the GST model. By time-agnostic, we mean properties that can depend on the order in which events happen but not on time as measured by the parties. Most properties considered in distributed computing are time-agnostic. The converse was already known, even without the time-agnostic requirement, so our result shows that the two network conditions are, under one sensible assumption, equally demanding.

**2012 ACM Subject Classification** Theory of computation → Distributed computing models

**Keywords and phrases** partial synchrony, unknown latency, global stabilization time

**Digital Object Identifier** 10.4230/LIPIcs.DISC.2024.51

**Category** Brief Announcement

## 1 Introduction

Distributed computing systems underpin a vast array of contemporary technological advancements, ranging from cloud computing platforms to blockchain networks. These systems rely on protocols to ensure consistency and reliability even when faced with challenges such as message delays and node failures. A cornerstone of designing robust protocols lies in understanding the communication model assumed by the distributed system. Within the distributed computing literature, the synchronous and asynchronous communication models remain the two best-established paradigms. The synchronous model assumes a publicly known upper bound $\Delta$ on message delays and that parties hold synchronized clocks. This idealized setting facilitates the design of elegant round-based protocols that often achieve very high resilience thresholds. However, the synchronous model exhibits a fundamental limitation: any deviation from the assumed message delay bound $\Delta$ can render synchronous protocols entirely ineffective, potentially leading to complete breakdowns of the protocols.

The asynchronous model, on the other hand, only assumes that messages get delivered eventually. This inherent flexibility empowers the asynchronous model to support protocols that can gracefully adapt to any network conditions. However, asynchronous protocols typically exhibit lower resilience thresholds compared to their synchronous counterparts, and even achieving agreement when parties might crash is impossible without randomization [3]. Hence, neither of these two extremes perfectly captures real-world systems: the synchronous model's assumptions are too strong, while the asynchronous model is too pessimistic. In this work, we are concerned with a middle ground between the two: the *partially synchronous* model, a nuanced paradigm that bridges the gap between the two, introduced by Dwork, Lynch, and Stockmeyer [2]. The work of [2] proposes two definitions for the partially synchronous model, described below (see the next section for the formal definitions).

**The Global Stabilization Time (GST) model.** This variant acknowledges that there might be periods of unpredictable delays due to network congestion or outages but also assumes that these disruptions eventually resolve and the system stabilizes. [2] explains how this intuition can be faithfully captured with a simple model, known as the Global Stabilization Time (GST) model: there is an unknown 'Global Stabilization Time' $\mathsf{T}$ after which the system behaves synchronously for a publicly-known message delivery bound $\Delta$. In particular, there is a publicly known amount of time $\Delta$ such that every message sent at time $\mathsf{t}$ is delivered by time $\max(\mathsf{t}, \mathsf{T}) + \Delta$.

**The Unknown Latency (UL) model.** In this variant, the system is, in fact, always synchronous: there is a value $\Delta$ such that every message sent by time $\mathsf{t}$ is delivered by time $\mathsf{t} + \Delta$. However, as opposed to the synchronous model, the value of $\Delta$ is unknown to the protocol.

**The relationship between the two.** The two models are conjectured to be equivalent, in the sense that any property that can be achieved by a protocol in one can also be achieved in the other. In fact, there is an elegant folklore reduction from the UL model to the GST model, presented in [1], which we explain next. Consider a protocol $\Pi$ achieving some property $\mathsf{X}$ in the UL model. Let us run $\Pi$ in the GST model, where a value of $\Delta$ is provided and guaranteed to hold eventually, but $\Pi$ ignores it. Consider any execution $\varepsilon$ of $\Pi$ in this setting: the model ensures that there is a time $\mathsf{T}$ such that all messages in $\varepsilon$ sent at time $\mathsf{t}$ get delivered by time $\max(\mathsf{t}, \mathsf{T}) + \Delta$. Hence, in $\varepsilon$ all messages get delivered within time $\mathsf{T} + \Delta$, so from the parties' perspective, they might just as well be running in UL with the unknown bound on message delay being $\mathsf{T} + \Delta$. Hence, $\varepsilon$ is also a legal execution of $\Pi$ in UL. As a result, the set of executions of $\Pi$ in GST is a subset of its set of executions in UL. Since $\Pi$ satisfies $\mathsf{X}$ in UL, it also satisfies $\mathsf{X}$ in GST.

The same blog post [1] also explains the reverse direction: a protocol designed for the GST model can be transformed into an equivalent protocol for the UL model, but only if it satisfies a certain property, namely, that the protocol's guarantees are still maintained if the assumed value of $\Delta$ changes dynamically. This way, one may increment the assumed $\Delta$ whenever a timeout of the protocol expires, and eventually, the assumed $\Delta$ exceeds the real one. However, as [1] points out, assuming this property is with loss of generality. Whether the converse holds is still an open question.[1]

**Our contribution.** In this work, we answer this question in the affirmative under the relatively minor technical assumption of only considering 'time-agnostic' properties. A protocol property

[1] The incorrect proof also implicitly assumes the time-agnostic property which we introduce in the next paragraph.

is *time-agnostic* if whether it holds for a given execution of a protocol can only depend on the relative order in which events happened, but not on time as measured by the parties in the execution. We note that most properties considered in distributed computing are indeed time-agnostic; e.g., whether some consensus protocol satisfies a given agreement conditions. Bounds on message complexity can also be accommodated, but the same is not true about running time. Additionally, we will only show our result assuming that the environment provides a global perfect clock to the parties, that is their only way of telling time. On a similar note, we consider randomized protocols, but do not consider probabilistic properties, such as "with probability at least 0.5 all parties terminate". We leave a formal argument considering imperfect clocks and probabilistic properties for future work. On the other hand, our proof works in adversarial settings, i.e., crashes or byzantine behavior. The key idea in our proof is that, instead of estimating the actual value of $\Delta$ in the UL model, like in the argument of [1], we continuously slow down the parties' clocks. This is achieved by the parties applying a wrapper function on top of the time measurements returned by the system clock. This way, the parties simulate running the GST protocol with a continuously increasing value of $\Delta$, which eventually exceeds the actual unknown $\Delta$ of the UL model, hence allowing the guarantees of the GST protocol that we are running to apply.

## 2 Preliminaries

We consider a fixed set of $n$ parties in a network, where links model communication channels. The parties are running a (possibly randomized) protocol over the network. For each party, the protocol is specified by a state transition diagram, where a party's state is defined by its local variables. The initial state of a party is then defined by any initial inputs and random coins. The transitions are deterministic (but may depend on the party's random coins). Without loss of generality, a party's transition to another state is triggered by the receipt of a message, or specific changes in time (e.g., waiting a predefined amount of time). State changes are instantaneous and include all required local computations and the corresponding sending of messages (i.e., these instructions are executed atomically). The receipt of messages, on the other hand, will be controlled by the *message system*, which we discuss below.

**Messages.** Messages are held in a global *message system*: this maintains a set containing tuples $(P_s, P_r, m, c)$, where $P_s$ is the sender of the message, $P_r$ is a receiver of the message, $m$ is the content of the message, and $c$ is a unique identifier assigned by the message system. The message system is controlled by the adversary and may decide when to deliver these messages (subject to the constraints of the communication model). For simplicity, we assume that the message system keeps delivering messages even after the receivers have terminated (if the protocol allows it) or crashed. Otherwise, claims of the form "eventually all messages get delivered within $\Delta$ time units" would not be meaningful for terminated receivers.

**Global clock.** We assume that parties have access to a common *global clock* denoted by $\textsc{Clock}$, which is their only source of time. Abstractly, $\textsc{Clock}$ is represented by an increasing and continuous function $\textsc{Clock} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ that maps real time to *system time*. In particular, at real time $t$, the parties can atomically query the global clock to read off a 'system time' of $\textsc{Clock}(t)$. Neither the parties nor the adversary have access to the actual definition of the function $\textsc{Clock}$. Instead, they can only use the global clock as an oracle to receive the current *system time*. Depending on the environment, the system time may coincide with real time, in which case $\textsc{Clock}$ is $\textsc{RealClock}(t) = t$, but this is not necessarily the case.

**Protocol execution models.** A protocol execution model $M$ captures all requirements and guarantees of the environment under which a protocol runs. For our scope, communication in $M$ always happens through message passing as already described. Moreover, $M$ specifies a global clock function $\textsc{Clock}$ that the parties use to tell time. Other aspects of the execution environment can appear as part of the guarantees of $M$, such as message delay bounds or other timing constraints. Two examples of such models $M$ are $\mathrm{GST}(\Delta, \textsc{Clock})$ and $\mathrm{UL}(\Delta, \textsc{Clock})$, formally introduced below. Note that the guarantees of a fixed model $M$ are concrete: e.g., messages are delivered within $\Delta$ time units for a fixed $\Delta$; in contrast, often in the literature, models usually refer to families of models (in this particular case, parameterized by $\Delta$). Last but not least, $M$ specifies the power of the adversary. On top of controlling the scheduler within the model's timing constraints, the adversary might, for instance, make parties crash, fail to send certain messages or deviate from the protocol arbitrarily (i.e., byzantine behavior). Model $M$ should specify precisely which faults are possible and under what circumstances (e.g., if the adversary is adaptive, computationally bounded, and how many parties it can corrupt). The parties are not aware of the clock function used: this is supplied by the environment as an oracle, with no access to its implementation. More abstractly, a model $M$ specifies for each protocol $\Pi$ its set of legal executions $\varepsilon$, defined next.

**Executions.** Consider a protocol $\Pi$ running in a model $M$ where parties measure time using function $\textsc{Clock}$. An execution of $\Pi$ is defined by the parties' initial states and a (possibly infinite) collection of events, denoted by $\textsc{events}(\varepsilon)$. Each event in $\textsc{events}(\varepsilon)$ is a tuple ($t$, $\textsc{ReceivedMsgs}$, $P$, $q$, $\textsc{SentMsgs}$) signifying that, at system time $t$ (i.e., as observed by the parties using function $\textsc{Clock}$), party $P$ received the (possibly empty) multiset of messages $\textsc{ReceivedMsgs}$ from the message system, $P$'s state became $q$ (possibly the same state it was already in), and $P$ sent the (possibly empty) multiset of messages $\textsc{SentMsgs}$ to the message system. We say that a message $\textsc{msg} = (P_s, P_r, m, c)$ was sent at system time $t$ in execution $\varepsilon$ if $\textsc{events}(\varepsilon)$ contains some event $(t, \textsc{ReceivedMsgs}, P_s, q, \textsc{SentMsgs})$ with $\textsc{msg} \in \textsc{SentMsgs}$. Similarly, we say that a message $\textsc{msg} = (P_s, P_r, m, c)$ was received at system time $t$ in execution $\varepsilon$ if $\textsc{events}(\varepsilon)$ contains some event $(t, \textsc{ReceivedMsgs}, P_r, q, \textsc{SentMsgs})$ with $\textsc{msg} \in \textsc{ReceivedMsgs}$. Note that a message sent/received at system time $t$ is sent/received at real time $\textsc{Clock}^{-1}(t)$. We have made the deliberate choice to timestamp executions in system time as this is the perspective that parties perceive them from. This will allow us to map between executions with different clock functions in our main result.

**The GST model.** The GST model has as parameters a clock function $\textsc{Clock}$ that the environment provides to the parties to tell the time when running a protocol, and $\Delta$, to be supplied to protocols designed for the model when instantiated for a specific $\Delta$. We write $\mathrm{GST}(\Delta, \textsc{Clock})$ for the model instantiated with specific parameters $\Delta$ and $\textsc{Clock}$. The model guarantees that, for every protocol $\Pi$ and every execution $\varepsilon$ of $\Pi$ in the model, there exists a time $T$ measured in real time such that every message in $\varepsilon$ sent at real time $t$ is received by real time $\max(t, T) + \Delta$. The model can be altered to give the adversary more power than controlling the scheduler; e.g., to corrupt parties.

**The UL model.** The UL model has as parameters a clock function $\textsc{Clock}$ that the environment similarly provides to the parties, and $\Delta$, not to be supplied to protocols designed for the model. We write $\mathrm{UL}(\Delta, \textsc{Clock})$ for the model instantiated with specific parameters $\Delta$ and $\textsc{Clock}$. The model guarantees that, for every protocol $\Pi$ and every execution $\varepsilon$ of $\Pi$ in the model, any message in $\varepsilon$ sent at real time $t$ is received by real time $t + \Delta$. This model can also be altered to give more power to the adversary.

**Protocol properties.** We define a protocol property as a set of allowed executions; e.g., the property that all parties eventually terminate, or that they produce some outputs. A protocol achieves a property in a model $M$ if all its legal executions in that model satisfy the property, i.e., are in the set of executions allowed by the property. Note that modeling certain properties this way is non-trivial, as executions alone do not contain, e.g., who are the byzantine parties and when they were corrupted. However, even such properties can be modeled: executions may contain changes of states that do not follow from the protocol's state transition to model parties misbehaving, or one can modify executions to include corruption events to make the process more transparent. In this paper, we are concerned with time-agnostic properties, defined next. We call two executions $\varepsilon, \varepsilon'$ *equivalent* if they differ only in the timestamps of the events and agree on the relative order of the events. A property $X$ is *time-agnostic* if for any two equivalent executions $\varepsilon, \varepsilon'$ it holds that $\varepsilon \in X \iff \varepsilon' \in X$.

**Augmented models.** Our result will be very general: we will consider an arbitrary protocol $\Pi$ designed for the GST model, instantiated with a publicly-known eventual message delay bound of 1, that satisfies a given time-agnostic property in $\text{GST}(1, \textsc{RealClock})$. We will show how $\Pi$ can be transformed into a protocol $\Pi'$ that only depends on $\Pi$ that satisfies the same property in $\text{UL}(\Delta, \textsc{RealClock})$, irrespective of the value of $\Delta$. Moreover, if we augment both the GST model and the UL model with the same kind of additional power for the adversary, the same statement holds, with the same proof. For simplicity, in the following, we assume the basic models, but we note that we also get the result for a plethora of more interesting fault settings, e.g., byzantine faults and crashes.

## 3 Our Reduction

This section presents the proof of our main result, stated below.

▶ **Theorem 1.** *Any time-agnostic property that can be achieved by a protocol in the GST model can also be achieved by a protocol in the UL model.*

As previously mentioned, the key idea behind our reduction will be slowing down time. Given a protocol $\Pi$ achieving a time-agnostic property $X$ in $\text{GST}(1, \textsc{RealClock})$, we construct a protocol $\Pi'$ such that any execution $\varepsilon'$ of $\Pi'$ in $\text{UL}(\Delta, \textsc{RealClock})$ for some $\Delta$ unbeknownst to the protocol is equivalent to a legal execution $\varepsilon$ of $\Pi$ in $\text{GST}(1, \textsc{RealClock})$, hence also achieving property $X$. Protocol $\Pi'$ will simulate running $\Pi$ with a modified system clock that continuously slows down, so that equal intervals of time measured in the simulated system will represent longer and longer spans of real time. Moreover, we need that the modified system clock eventually gets arbitrarily slow. This way, since $\Pi$ is designed to have property $X$ if, once sufficient time passes, every message gets delivered within 1 unit of system time, this will eventually be the case: the clock gets slow enough for 1 unit of system time to correspond to a span of real time exceeding the unknown message delay bound.

More specifically, $\Pi'$ will simulate $\Pi$ running with system clock $\textsc{SlowClock} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ given by $\textsc{SlowClock}(t) = \sqrt{t}$ (any increasing function whose derivative tends to 0 as $t \to \infty$ will suffice). Whenever a party in the simulated $\Pi$ queries the global clock and the answer would have normally been (real time) $t$, $\Pi'$ replaces the answer with $\textsc{SlowClock}(t)$: from the perspective of the simulated $\Pi$, the system clock is $\textsc{SlowClock}$.

The first lemma below shows the required result assuming that $\Pi$ is running standalone but with system clock $\textsc{SlowClock}$. The second lemma lifts it to the protocol $\Pi'$ that runs with system clock $\textsc{RealClock}$, but simulates $\Pi$ running with system clock $\textsc{SlowClock}$. A short discussion of why this implies Theorem 1 follows.

▶ **Lemma 2.** *Consider a protocol $\Pi$ and a legal execution $\varepsilon$ of $\Pi$ in $UL(\Delta, \textsc{SlowClock})$. Then, $\varepsilon$ is a legal execution of $\Pi$ in $GST(1, \textsc{RealClock})$.*

**Proof.** Consider an execution $\varepsilon$ of $\Pi$ in $\mathrm{UL}(\Delta, \textsc{SlowClock})$, which guarantees that any message sent at real time $t$ is delivered by real time $t+\Delta$. From the perspective of the parties, however, time is measured using $\textsc{SlowClock}$, so in $\varepsilon$, the parties observe that any message sent at system time $\textsc{SlowClock}(t)$ is delivered by system time $\textsc{SlowClock}(t+\Delta)$. Let us consider $\textsc{SlowClock}(t+\Delta)-\textsc{SlowClock}(t) = \sqrt{t+\Delta}-\sqrt{t}$ to understand how the message delay observed by the parties evolves with $t$. Taking the derivative, the function is strictly decreasing with $t$, so the observed network delay gets smaller and smaller as time passes. Subsequently, let us find a bound $t_0$ on $t$ such that starting at real time $t_0$, the observed network delay is bounded by 1; i.e., let us solve $\sqrt{t+\Delta}-\sqrt{t} \leqslant 1$. If $\Delta < 1$, this happens for $t \geqslant 0$. Otherwise, $\Delta \geqslant 1$, and this happens for $t \geqslant \frac{1}{4}(\Delta-1)^2$. Hence, starting at real time $\frac{1}{4}(\max\{1,\Delta\}-1)^2$, the observed (system) network delay is bounded by 1. Writing the same in terms of system time, starting at system time $T := \sqrt{\frac{1}{4}(\max\{1,\Delta\}-1)^2} = \frac{1}{2}(\max\{1,\Delta\}-1)$, the system network delay is bounded by 1. In particular, this means that a message sent at system time $t$ in $\varepsilon$ is delivered by system time $\max\{t, T\}+1$. Hence, $\varepsilon$ could just as well be an execution of $\Pi$ in $\mathrm{GST}(1, \textsc{RealClock})$ with global stabilization time $T$ because the parties and the adversary are unaware of the clock function used. ◀

▶ **Lemma 3.** *If protocol $\Pi$ achieves a time-agnostic property $X$ in $\mathsf{GST}(1, \textsc{RealClock})$, there is a protocol $\Pi'$ depending only on $\Pi$ that achieves $X$ in UL($\Delta$, $\textsc{RealClock}$) $\forall\Delta \geqslant 0$.*

**Proof.** In protocol $\Pi'$ parties run protocol $\Pi$ but apply the function $\textsc{SlowClock} : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ as a wrapper over the global clock's responses to the queries. In particular, whenever a party queries the global clock in $\Pi$ and the time returned is $t$, the party evaluates $\textsc{SlowClock}(t)$ and takes this as the answer instead. Every execution $\varepsilon'$ of $\Pi'$ in some model $M$ corresponds to an equivalent execution $\varepsilon$ of $\Pi$ in $M$ where the clock function provided by the environment is composed with $\textsc{SlowClock}$. Namely, every event $e$ present in $\varepsilon$ and $\varepsilon'$ is timestamped $t$ in $\varepsilon$ and $\textsc{SlowClock}(t)$ in $\varepsilon'$.

Hence, for any $\Delta \geqslant 0$, any legal execution $\varepsilon'$ of $\Pi'$ in $\mathrm{UL}(\Delta, \textsc{RealClock})$ corresponds to a legal execution $\varepsilon$ of $\Pi$ in $\mathrm{UL}(\Delta, \textsc{SlowClock})$ that is equivalent to $\varepsilon'$. By Lemma 2, $\varepsilon$ is also a legal execution of $\Pi$ in $\mathrm{GST}(1, \textsc{RealClock})$. Since $\Pi$ achieves property $X$ in $\mathrm{GST}(1, \textsc{RealClock})$, it follows that $\varepsilon \in X$, and hence, since $X$ is time-agnostic, $\varepsilon' \in X$. Since $\varepsilon'$ was an arbitrary execution of $\Pi'$ in $\mathrm{UL}(\Delta, \textsc{RealClock})$ and $\Delta \geqslant 0$ was arbitrary, it follows that $\Pi'$ satisfies property $X$ in $\mathrm{UL}(\Delta, \textsc{RealClock})$ for all $\Delta \geqslant 0$. ◀

**Proof of Theorem 1.** If $\mathbf{\Pi}$ denotes the set of all protocols, a protocol designed for the GST model is, in fact, a protocol family $\Pi : \mathbb{R}_{\geqslant 0} \to \mathbf{\Pi}$, one for each potential value of the publicly-known eventual message delay bound. $\Pi$ achieves a property in $\mathrm{GST}(\Delta, \textsc{RealClock})$ for some $\Delta \geqslant 0$ iff all executions of $\Pi(\Delta)$ achieve this property in $\mathrm{GST}(\Delta, \textsc{RealClock})$.

Let $\Pi$ be a protocol achieving a time-agnostic property $X$ in the GST model. We only need that $\Pi(1)$ satisfies $X$ in $\mathrm{GST}(1, \textsc{RealClock})$: applying Lemma 3 to $\Pi(1)$, we get that $\Pi'$ satisfies $X$ in $\mathrm{UL}(\Delta, \textsc{RealClock})$ for all $\Delta \geqslant 0$, implying the conclusion. ◀

## References


1 Ittai Abraham. Flavours of partial synchrony, 2019. URL: `https://decentralizedthoughts.github.io/2019-09-13-flavours-of-partial-synchrony/`.

2 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, apr 1988. doi:10.1145/42282.42283.

3 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, apr 1985. doi:10.1145/3149.214121.