Adaptive Failure Detection in Logical Time: The Eventual Relative-Speed Model over Causal Blocks

Raimundo José de Araújo Macêdo
macedo@ufba.br

Distributed Systems Laboratory (LaSiD)
DCI/IC – Federal University of Bahia (UFBA)

Abstract. *In this paper, we introduce the Eventual Relative-Speed (ERS) system model, a logical-time abstraction for distributed systems that lack an absolute physical time reference. In the ERS model, the relative progress of processes is governed by a finite parameter Γ, which characterizes the logical-time divergence among correct processes and holds only after an unknown Global Stabilization Time (GST).*

We propose a unified Causal Progress and Failure Detection algorithm built atop the Causal Blocks framework. By employing an adaptive suspicion threshold η_i, our algorithm ensures causal consistency and satisfies the properties of an Eventually Perfect ($\diamond\mathcal{P}$) failure detector. Because it operates entirely in logical time, the system is intrinsically adaptive to fluctuations in relative speed—such as slowing or accelerating—without requiring the dynamic calibration of physical timeouts.

We formally prove that the detector converges to Eventual Strong Accuracy as it adapts to transient delays, eventually stabilizing within the Γ bound. This provides a robust foundation for distributed computing in environments subject to arbitrary global load fluctuations.

1. Introduction

Failure detection is a fundamental building block for reliable distributed systems, supporting critical services such as group membership, state-machine replication, and distributed coordination. Since the seminal work of [Chandra and Toueg 1996], failure detectors have been studied as primary abstractions that encapsulate timing assumptions, thus enabling the design of robust distributed algorithms in the face of uncertainty [Raynal 2005, Freiling et al. 2011].

Traditional failure detectors are typically grounded in *physical time*, relying on periodic heartbeat messages and timeout mechanisms to infer process crashes [Chen et al. 2002]. While effective in stable environments, these approaches are inherently sensitive to network jitter and require meticulous calibration of timeout values. In dynamic settings where message delays and processing speeds fluctuate significantly, fixed timeouts often lead to a forced trade-off: either triggering frequent false suspicions during transient overloads or unacceptably delaying the detection of genuine failures. To cope with the inherently non-linear nature of such environments, we and several other authors have proposed complex adjustment techniques to maximize the performance of failure detectors [de Araújo Macêdo 2000, Chen et al. 2002, de Araújo Macêdo and e Lima 2004, Falai and Bondavalli 2005, Sá and de Araújo Macêdo 2010].

This paper explores an alternative paradigm for handling the dynamic nature of distributed systems, based on *logical time*, leveraging the structural properties of the Causal Blocks communication model [de Araújo Macêdo 1994]. Causal Blocks provide a logical-clock abstraction that, in the spirit of Lamport clocks [Lamport 1978], captures causality through monotonically increasing block numbers. While this abstraction has been successfully employed to design various group communication protocols (e.g., [Macêdo et al. 1993, Ezhilchelvan et al. 1995, de Araújo Macêdo and Freitas 2009]), its potential as a substrate for failure detection in "time-free" environments remains largely unexplored.

Our motivation stems from the need for distributed applications to maintain *strong state consistency* across participants in environments where processing and communication loads can fluctuate over time. Examples include active state-machine replication, where replicas must process an identical sequence of commands [Schneider 1990], and cooperative autonomous vehicle networks, where coordinated decision-making depends on a consistent view of the shared environment [Zhou et al. 2025]. In such systems, maintaining a coherent notion of progress is essential; any significant deviation in a process's logical progression must be reliably detected to preserve system integrity.

However, accurately detecting such deviations in practice is challenging. In many deployments, network congestion or resource contention affects the system holistically rather than impacting individual processes in isolation. As a result, absolute measures of progress often fail to reflect the actual activity of cooperating processes. This observation motivates the adoption of a relative-speed model, wherein the progress of correct processes may vary in absolute terms but remains bounded relative to one another. Under this model, reasoning in terms of logical time is particularly appealing: instead of relying on physical time, the system monitors the relative progression of processes through logical timestamps. A process that fails to keep pace with the others can thus be identified as faulty solely on the basis of its logical lag.

Building on this intuition, we introduce the Eventually Relative-Speed (ERS) model. In this specification, we assume that for every run of the system, there exists a finite constant $\Gamma \geq 1$ that bounds the relative logical progress between any two correct processes. The value of Γ is unknown to the processes, and this bound is only guaranteed to hold after an unknown Global Stabilization Time (GST). This model is the direct logical-time equivalent of the partially synchronous model defined by Dwork, Lynch, and Stockmeyer (DLS) [Dwork et al. 1988]—specifically the variant where both the bounds and the stabilization time are unknown—yet it entirely removes absolute physical time from the equation.

As a proof of concept of the application of the **ERS** Model, we extend the Causal Blocks framework [de Araújo Macêdo 1994] with a *periodic block creation mechanism* that ensures the continuous progression of logical time and enables the detection of processes that cease contributing to the causal structure.

The main contributions of this paper are as follows: (1) formalizing the **ERS** Model, a "time-free" alternative to partial synchrony where relative progress rates stabilize after an unknown Global Stabilization Time (GST); (2) introducing an *Adaptive Logical-Time Failure Detector* over Causal Blocks, using block numbers to implicitly

measure activity; (3) proposing a *Watchdog mechanism* for continuous logical progress; (4) formally proving the detector achieves Eventually Perfect ($\diamond\mathcal{P}$) properties by adapting to pre-GST transient delays via a dynamic threshold (η_i); and (5) discussing practical implementation issues and resource constraints.

The remainder of this article is structured as follows. Section 2 presents the *ERS* System Model. Section 3 discusses related work. Section 4 presents the proposed Logical-Time-Based Adaptive Failure Detector and its corresponding correctness proofs. Section 5 discusses implementation trade-offs, and Section 6 provides concluding remarks and discusses future work.

2. System and Computation Model

We consider a distributed system composed of a finite set of processes $\Pi = \{p_1, p_2, \ldots, p_n\}$. Each process executes an infinite sequence of atomic steps. A step consists of a local computation, the transmission of a message, or the reception of a message. We assume a purely distributed environment; specifically, the model does not assume the existence of global clocks or shared memory.

Processes communicate by exchanging messages in FIFO order over a reliable but asynchronous network. While the network guarantees that no messages are lost, it imposes arbitrary, yet finite, delays.

2.1. Process Groups and Views

For analytical convenience, we assume that all processes in the system form a single, unique group, with a communication structure organized into a **View** $V \subseteq \Pi$ [1]. This group evolves through a sequence of views $(v_i^1, v_i^2, \ldots)$. The initial view for process p_i, denoted v_i^1, is composed of the set of all processes in the system ($v_i^1 = \Pi$), and each process p_i maintains a local set v_i^k representing its current perception of the group's active membership V at any given logical time. A process is considered *correct* if it never crashes; otherwise, it is *faulty*. When a faulty process p_j crashes, it permanently halts, ceasing to execute steps and send messages. The primary role of the failure detector is to provide p_i with a suspicion predicate $\psi_i(p_j)$, allowing the system to track these crashes and eventually transition to a subsequent new view v_i^{k+1} that safely excludes the faulty members.

2.2. The Eventual Relative-Speed (ERS) Assumption

Building on the view's structural foundation, system progress is defined not through physical intervals but through the relative advancement of processes.

Definition 1 (Eventual Relative-Speed Rate) Let Δ_i and Δ_j be the number of logical steps (e.g., block increments) executed by correct processes $p_i, p_j \in V$ during any given interval. The system satisfies the **Eventual Relative-Speed Assumption** if there exists a finite constant $\Gamma \geq 1$ and an unknown Global Stabilization Time (GST) such that, for any interval after GST:

$$\frac{1}{\Gamma} \leq \frac{\Delta_i}{\Delta_j} \leq \Gamma \tag{1}$$

[1] though the architecture can be easily extended to accommodate multiple process groups

While a constant Γ exists for every run, it is only guaranteed to hold post-GST. Consequently, a failure detection protocol must remain robust regardless of GST, dynamically adapting its threshold (η_i) to accommodate pre-GST instability. This adaptation is essential to mitigate transient false suspicions and ensure that the detector eventually satisfies its accuracy properties once the system stabilizes.

3. Related Work

While the body of research on distributed system models and failure detection is vast, we review here only a few key classical and relevant works to properly contextualize our contribution.

The Eventually Relative-Speed (ERS) model bridges the gap between classical synchrony ($\Gamma = 1$) [Lamport et al. 1982] and the asynchronous extreme ($\Gamma \to \infty$) [Fischer et al. 1985] by parameterizing system behavior through relative logical progress rather than absolute physical time. This approach serves as the logical-time counterpart to the DLS partially synchronous model [Dwork et al. 1988], where bounds are guaranteed only after an unknown Global Stabilization Time (GST). While the original DLS specification restricts absolute real-time delays (Δ and Φ), ERS strictly bounds unitless logical divergence (Γ) post-GST, effectively removing physical time from the equation. Several existing frameworks have explored logical-time modeling or relative-speed progress; a few notable examples follow.

The Θ-Model [Widder and Schmid 2009] enables flexible synchrony by bounding the ratio of message delays, Θ, after an unknown GST. However, while the Θ-Model is rooted in physical-time progress and temporal message delivery, ERS abstracts away physical duration.

The Timetide programming model explicitly models transmission delays as fixed logical ticks rather than physical time [Kenwright et al. 2025]; in this sense, it is similar to the ERS model. However, while Timetide assumes invariant logical delays to maintain determinism, our ERS model is designed to handle transient variations in the relative speeds.

The authors in [Spirakis and Tampakas 1988] used the Archimedean assumption—bounding the ratio of process speeds and message delays—to improve message complexity for problems like mutual exclusion. Unlike this physical-time approach, the ERS model expresses synchrony bounds purely through logical progress.

The Asynchronous Bounded-Cycle (ABC) model [Robinson and Schmid 2011] also avoids physical timeouts, but it derives synchrony exclusively from the topological properties of specific message exchanges ("relevant cycles"). In contrast, ERS directly bounds overall execution rates via Γ, tracking global progress through bounded logical-time divergence.

While some models partition systems into synchronous regions that communicate asynchronously [Veríssimo and Casimiro 2002, de Araújo Macêdo 2025], ERS does not require explicit partitions or local lockstep synchronization. Instead, synchrony in ERS emerges globally from the bounded relative divergence Γ, providing a strictly weaker condition than classical lockstep.

The Heartbeat (HB) failure detector proposed in [Aguilera et al. 1997] avoids

physical timeouts by outputting raw, uninterpreted message counters, a mechanism that relies on the assumption of fair links. In contrast, the ERS-based failure detector introduced in this paper interprets global dependencies through Causal Blocks to generate a definitive suspicion predicate. This process, as we show later, is governed by a logical divergence threshold, η_i, that dynamically adapts to changing system conditions.

Figure 1 illustrates the positioning of the ERS model, occupying the intermediate space where bounds exist but are only guaranteed to hold eventually (after an unknown Global Stabilization Time, GST), specifically targeting logical relative-speed rather than absolute physical time.

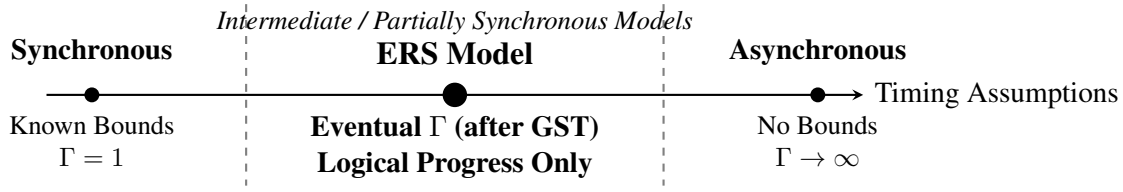

Figure 1. Positioning the ERS model within the synchrony spectrum.

4. The Logical-Time-Based Adaptive Failure Detector

Assuming the ERS model, the proposed failure detector leverages and extends the Causal Blocks communication model. Before detailing the internal dynamics of the failure detector and its adaptive threshold mechanism, we present a brief overview of the Causal Blocks framework [de Araújo Macêdo 1994].

4.1. The Causal Blocks Framework

Each process p_i maintains a logical clock called the **Block Counter** (BC_i), which increases monotonically. Messages are timestamped with these counters to respect potential causality, specifically satisfying two fundamental causal ordering properties:

- **Causal Property** $pr1$: If a process p_i sends message m before sending message m' ($send_i(m) \rightarrow send_i(m')$), then $m.b < m'.b$.
- **Causal Property** $pr2$: If a process p_i delivers message m before sending message m' ($deliver_i(m) \rightarrow send_i(m')$), then $m.b < m'.b$.

Unlike Lamport clocks, BC_i advances specifically on **send** and **delivery** events. This ensures that a message depends only on previously delivered information, directly supporting $pr2$.

Counter Advancement Rules:

- **CA1**: Before $send_i(m)$, increment BC_i and assign $m.b \leftarrow BC_i$.
- **CA2**: Before $deliver_i(m)$, set $BC_i \leftarrow \max(BC_i, m.b)$.

Messages sharing the same block number are grouped into **Causal Blocks**. A block $M[B]$ is represented as a vector of size n, where each entry j indicates whether process p_j contributed a message to block B. By construction, messages within the same Causal Block are guaranteed to be concurrent, meaning they have no causal relationship. These blocks are ordered by increasing numbers, forming the **Block Matrix** M. In the

Causal Blocks representation, the presence of a message in entry $M_i[B][j]$ is indicated by the "+" mark.

Lemma 4.1 (Block Completion)**.** *A Causal Block $M_i[B]$ is complete at process p_i if and only if, for every process p_j in the current view v_i, the following condition holds:*

- ***LC1**: The j-th entry of $M_i[B]$ either (i) contains a message, or (ii) is empty and there exists a block $B' > B$ such that the j-th entry of $M_i[B']$ contains a message.*

Proof. The "only if" direction follows directly from the construction of Causal Blocks and the monotonicity of block numbers. For the "if" direction, condition LC1 ensures that every process p_j has either contributed to block B or has advanced to a subsequent block $B' > B$. Due to the FIFO property of the communication channels and the monotonicity of the Block Counter (BC_j), no process p_j that has already contributed to B' can subsequently send a message belonging to an earlier block B. Therefore, once LC1 is satisfied for all $p_j \in v_i$, all messages that can possibly belong to $M_i[B]$ have been accounted for at p_i. ■

Definition (Last Known Block) At any given physical time t, let $\mathcal{M}_{i,j}(t)$ be the set of all messages received by p_i from p_j. We define the last known block of p_j at p_i, denoted as $last_{i,j}$, as:

$$last_{i,j} = \max\{m.b \mid m \in \mathcal{M}_{i,j}(t)\} \tag{2}$$

Considering a 3-process group, Figure 2 illustrates block completion as perceived by a process p_1. The receipt of p_3's message in B_3 allows p_1 to advance $last_{1,3}$ to 3, satisfying the completion requirement for B_2.

$$M_1 = \begin{bmatrix} + & + & + \\ + & + & \cdot \\ + & + & + \\ \cdot & + & \cdot \end{bmatrix} \begin{array}{l} \leftarrow B_1 : \text{Complete} \\ \leftarrow B_2 : \text{Implicitly Complete (via FIFO from } p_3) \\ \leftarrow B_3 : \text{Complete} \\ \leftarrow B_4 : \text{Active Block} \end{array}$$

Figure 2. Block Matrix M_1

4.2. The Suspicion Predicate

The failure detection logic in the ERS model is grounded in the logical lag between processes. Let $Bmax_i$ be the highest block number currently represented in the local Block Matrix M_i. The suspicion predicate $\psi_i(p_j)$ is formally defined as:

$$\psi_i(p_j) \equiv (Bmax_i - last_{i,j}) > \eta_i \tag{3}$$

where η_i is the *Adaptive Threshold* that process p_i maintains locally for its entire current view v_i. This predicate evaluates to *true* when the logical divergence of any peer p_j exceeds the dynamically adapted threshold, which compensates for transient delays and desynchronization before the Global Stabilization Time (GST).

As illustrated in Figure 2, the logical lag for p_3 is determined by its distance from the highest active block B_4 (i.e., $Bmax_1 = 4$). Since p_3's last recorded contribution is

in block 3 ($last_{1,3} = 3$), its logical lag is exactly 1. Assuming an adaptive threshold of $\eta_1 = 2$, process p_1 will not trigger a suspicion on p_3, as this lag remains well within the permitted limit of two incomplete blocks.

4.3. Periodic Block Creation

In the original Causal Blocks model, block completion was ensured by a time-silence mechanism that guaranteed progress but did not directly support failure detection. We introduce a **Watchdog Periodic Block Creation** mechanism that simultaneously ensures block completion and provides the logical-time basis for failure detection. This mechanism monitors process activity over a timeout period T and manages transitions between application-driven and timer-driven progress. Assume that $SENT_i$ is the block number of the last message sent by p_i. We extend the original model with the following advancement rule: **CA3 (Watchdog Rule):** Upon expiration of the watchdog timer T, if p_i has not contributed to the current logical frontier ($SENT_i < Bmax_i$), it multicasts a null message with $m_{null}.b \leftarrow Bmax_i$. If p_i is already at the frontier, it initiates a new block by setting $m_{null}.b \leftarrow Bmax_i + 1$.

This mechanism ensures that the difference between $Bmax_i$ (the highest block number currently represented in the local Block Matrix M_i) and the last known contribution of a peer p_j ($last_{i,j}$) remains a reliable indicator of relative progress. By forcing periodic logical updates, the watchdog provides the "logical heartbeat" necessary to distinguish a crashed process from one that is merely idle, without requiring physical timeouts to detect failures.

4.4. Protocol for Causal Progress and Failure Detection

The Watchdog-Based Causal Progress Protocol (Algorithm 1) describes how a process p_i manages its logical clock, BC_i, across concurrent communication and monitoring events within the ERS model. It provides a unified solution for maintaining causal progress while dynamically adapting to transient delays and desynchronization before the Global Stabilization Time (GST).

Algorithm 1 operates through three core mechanisms: **(1) Watchdog-Driven Logical Progress:** To ensure the failure detector remains active during application silence, p_i employs a watchdog timer T. If idle, **Rule CA3** (Lines 17–22) forces the advancement of BC_i and the multicast of a null message, providing the "logical heartbeat" peers need to complete blocks without physical timeouts. **(2) Message Reception and Threshold Adaptation:** Arriving messages (Lines 10–16) are recorded in the **Block Matrix** (M_i). Because the relative-speed bound Γ holds only after an unknown Global Stabilization Time (GST), the threshold η_i is adaptive. Receiving a message from a suspected process ($\psi_i(p_j) = \text{true}$) reveals a false suspicion (Line 14). The algorithm retracts the suspicion and increases η_i to accommodate the observed logical gap (Line 15), effectively adapting to pre-GST transient delays. **(3) Decoupled Delivery and Monitoring:** The **DeliveryAndMonitoring** procedure (Lines 23–37) runs in parallel to prevent high-latency reception from blocking local progress. It executes two concurrent tasks: *Causal Delivery* identifies completed blocks (Lemma 4.1) and applies **Rule CA2** (Line 27) to synchronize BC_i strictly at delivery, preserving causal safety ($pr2$); and *Divergence Monitoring* continuously evaluates the gap between $Bmax_i$ and $last_{i,j}$, suspecting p_j if this gap exceeds η_i (Lines 33–35). Through this mechanism, the detector converges toward

Algorithm 1 Watchdog-Based Causal Progress and Failure Detection at p_i

Initial Variables: $BC_i, Bmax_i, SENT_i \leftarrow 0$; $T \leftarrow$ timeout; $\eta_i \leftarrow \eta_0$; $\forall p_j : \psi_i(p_j) \leftarrow$ false

procedure StartTimer() // Refresh timeout tc_i for period T
Cancel tc_i; Schedule new tc_i
end procedure

upon p_i wishes to SEND application message m // Rule CA1
if $BC_i = Bmax_i$ **then** $Bmax_i \leftarrow Bmax_i + 1$; **create block** $M_i[Bmax_i]$
end if
$BC_i \leftarrow BC_i + 1$; $m.b \leftarrow BC_i$; $SENT_i \leftarrow BC_i$
$M_i[BC_i][i] \leftarrow$ "+"; **multicast** m to Π; StartTimer()

upon receive m from p_j with block $m.b$
if $m.b > Bmax_i$ **then** $Bmax_i \leftarrow m.b$; **create block** $M_i[Bmax_i]$
end if
$M_i[m.b][j] \leftarrow$ "+"; store m
if $\psi_i(p_j) =$ true **then** // Adapt to transient delay (before GST)
    $\psi_i(p_j) \leftarrow$ false; $\eta_i \leftarrow \eta_i + 1$
end if

upon timeout tc_i // Rule CA3: Progress Watchdog
if $SENT_i < Bmax_i$ **then** $m_{null}.b \leftarrow Bmax_i$ // Fill existing frontier
else $Bmax_i \leftarrow Bmax_i + 1$; **create block** $M_i[Bmax_i]$; $m_{null}.b \leftarrow Bmax_i$ // Push new frontier
end if
$BC_i \leftarrow \max(BC_i, m_{null}.b)$; $SENT_i \leftarrow m_{null}.b$
$M_i[m_{null}.b][i] \leftarrow$ "+"; **multicast** m_{null} to v_i (current group view); StartTimer()

parallel procedure DeliveryAndMonitoring()
whenever M_i is updated **OR** BC_i increments:
for each incomplete block B in M_i (in increasing order) **do**
    if $\forall p_k \in v_i$: LC1 is satisfied for $M_i[B]$ **then** // Lemma 4.1
        $BC_i \leftarrow \max(BC_i, B)$ // Rule CA2
        deliver messages in $M_i[B]$ to application; mark block B as completed
    end if
end for
for each $p_j \in v_i \setminus \{p_i\}$ **do** // ERS Monitoring
    $last_{i,j} \leftarrow \max\{b \mid M_i[b][j] =$ "+"$\} \cup \{0\}$
    if $Bmax_i - last_{i,j} > \eta_i$ **then**
        $\psi_i(p_j) \leftarrow$ true
    end if
end for
end procedure

Eventual Strong Accuracy as η_i stops growing and stabilizes post-GST, encompassing the maximum logical drift allowed by Γ.

4.5. Correctness Proofs

We now demonstrate that the integrated ERS protocol (Algorithm 1) satisfies the properties of an Eventually Perfect ($\diamond\mathcal{P}$) failure detector while maintaining causal consistency ($pr2$), even when the relative-speed parameter Γ is only guaranteed to hold after an unknown Global Stabilization Time (GST).

4.5.1. Liveness: Strong Completeness

Theorem 4.2 (Strong Completeness)**.** *If process p_j crashes, then eventually every correct process p_i permanently suspects p_j.*

Proof. Suppose p_j crashes at physical time t. After t, p_j ceases all communication. Consequently, the highest block index for p_j in p_i's matrix, $last_{i,j} = \max\{b \mid M_i[b][j] =$ "+"$\}$, remains fixed at some finite value B_{fail}.

Since p_i is correct, it continues to execute the protocol. If the application layer is active, BC_i increments via Rule CA1 (Line 8). If the application is idle, the watchdog timer tc_i expires every T time units (Line 17). Under Rule CA3, even if $SENT_i$ is already at the frontier, the watchdog forces the creation of a new block $Bmax_i + 1$ (Line 19) and updates BC_i accordingly (Line 21).

Thus, BC_i grows monotonically and without bound ($BC_i \to \infty$). Because $Bmax_i \geq BC_i$ by definition, $Bmax_i$ inherently grows without bound as well ($Bmax_i \to \infty$). In the ERS model, the adaptive threshold η_i only increases when p_i receives a delayed message from a previously suspected process. Since p_j has crashed, it sends no further messages and cannot cause η_i to increase. For any remaining correct processes, once the Global Stabilization Time (GST) is reached, their relative logical progress stabilizes strictly within the finite Γ bound, preventing any further false suspicions. Consequently, η_i will eventually stop growing and stabilize at some finite value η^*.

Because $Bmax_i$ grows without bound while both $last_{i,j}$ and the stabilized threshold η_i remain finite, the logical gap $(Bmax_i - last_{i,j})$ will eventually exceed η_i. At this point, the monitoring condition (Line 33) is satisfied, and p_i sets $\psi_i(p_j) \leftarrow$ true. Since $last_{i,j}$ never increases again, the suspicion becomes permanent. ■

4.5.2. Safety: Eventual Strong Accuracy

Theorem 4.3 (Eventual Strong Accuracy)**.** *There exists a time after which no correct process is suspected by any other correct process.*

Proof. By the ERS model assumption, after an unknown Global Stabilization Time (GST), there exists a finite relative-speed constant Γ that strictly bounds the logical divergence between any two correct processes p_i and p_j. Let η^* be the maximum logical lag observed under this Γ after GST.

Suppose p_i falsely suspects a correct process p_j due to transient delays before GST, which caused the logical gap to exceed the current threshold η_i. Since p_j is correct, it will eventually send an application message or a watchdog null-message. Upon reception of

this delayed message m at p_i (Line 10), p_i detects the false suspicion (Line 14), retracts it ($\psi_i(p_j) \leftarrow$ false), and adapts to the observed transient delay.

At Line 15, the threshold is simply incremented ($\eta_i \leftarrow \eta_i + 1$). Because relative speeds stabilize after GST, the logical divergence between correct processes becomes strictly bounded by the finite constant η^*. Consequently, each adaptation strictly increments η_i until it converges to a value $\geq \eta^*$. After this convergence, the logical gap $(Bmax_i - last_{i,j})$ will not exceed η_i for the correct process p_j, satisfying Eventual Strong Accuracy. ■

4.5.3. Structural Safety: Causal Consistency

Theorem 4.4 (Causal Properties $pr1$ and $pr2$)**.** *The integrated ERS protocol ensures that for any two messages m, m', the following causal order properties hold:*

- $pr1$*: If* $send_i(m) \rightarrow send_i(m')$*, then* $m.b < m'.b$*.*
- $pr2$*: If* $deliver_i(m) \rightarrow send_i(m')$*, then* $m.b < m'.b$*.*

Proof. We prove each property based on the Counter Advancement Rules and their implementation in the `DeliveryAndMonitoring` parallel procedure.

Proof of $pr1$: Suppose a process p_i sends an application message m and subsequently sends another message m' (i.e., $send_i(m) \rightarrow send_i(m')$). When p_i sends m, Rule CA1 (Line 5) is triggered, incrementing the local logical clock $BC_i \leftarrow BC_i + 1$ and assigning $m.b \leftarrow BC_i$ (Line 8). Because BC_i is monotonically increasing and only increments further for subsequent sends, when p_i later prepares to send m', Rule CA1 is triggered again. This guarantees BC_i increments at least once more, making the new $m'.b$ strictly greater than $m.b$. Thus, $m.b < m'.b$.

Proof of $pr2$: Property $pr2$ requires that any message sent by p_i must have a block number strictly greater than any message previously delivered by p_i. Suppose p_i delivers a message m belonging to block B (meaning $m.b = B$). This delivery occurs within the `DeliveryAndMonitoring` procedure (Line 28). Immediately upon satisfying the completion condition for block B (Line 26), p_i executes Rule CA2: $BC_i \leftarrow \max(BC_i, B)$ (Line 27). This ensures the local logical clock BC_i is at least equal to the block index of the last delivered message. Any subsequent application message m' sent by p_i triggers Rule CA1 (Line 8). Even if BC_i was already at the frontier ($Bmax_i$), the algorithm ensures BC_i is incremented ($BC_i \leftarrow BC_i + 1$) at Line 8. Consequently, $m'.b$ is assigned a value of at least $B + 1$. Therefore, $m.b < m'.b$, preserving the causal order regardless of any transient delays or eventual relative speeds of the distributed system. ■

5. Implementation Discussion and Performance Trade-offs

While the preceding sections provide a formal characterization of the ERS model and its associated algorithms, it is instructive to consider the practical behavior of the parameters T and η_i in a physical environment. It must be emphasized, however, that a detailed empirical evaluation and specific system optimizations are **outside the scope of this article**. As a theoretical work, our primary aim is to introduce the logical-time model and establish the fundamental correctness of the protocols designed to operate within it.

5.1. Practical Constraints on the Adaptive Threshold

While the theoretical ERS model allows the adaptive threshold η_i to grow indefinitely to accommodate arbitrary transient delays before the Global Stabilization Time (GST) and encompass any eventual finite relative-speed constant Γ, practical implementations necessitate a superior limit, denoted as η_{max}. This upper bound is motivated by two primary factors: (1) **System Abstraction:** Allowing η_i to increase indefinitely would eventually cause the system to operate essentially as a purely asynchronous model. In such an unbounded state, failure detection becomes impossible due to the absence of any progress guarantees, violating the conditions required for achieving eventual synchrony; and (2) **Resource Management:** The Block Matrix M_i requires physical memory to maintain metadata and process contributions for each active block. As η_i increases, the number of concurrent, incomplete blocks that must be buffered grows proportionally. To prevent memory exhaustion, flow control mechanisms such as those proposed in [de Araújo Macêdo 1994,de Araújo Macêdo et al. 1995] are required to prevent senders from initiating new logical blocks that would exceed the available buffering capacity.

Furthermore, specific application requirements may dictate a maximum admissible divergence to ensure timely consistency or bounded latency among peers. Therefore, we assume that η_{max} is a predefined parameter provided by the application environment, and the adaptation process must strictly satisfy the constraint:

$$0 < \eta_i \leq \eta_{max} \tag{4}$$

If the environment's transient delays before GST, or its eventual relative-speed Γ after stabilization, exceed this practical limit η_{max}, the system may experience persistent false suspicions. This state indicates that network conditions have exceeded the operational capacity defined for the application, requiring external intervention or a reassessment of the view v_i. Algorithm 1 can be easily adjusted to comply with η_{max}.

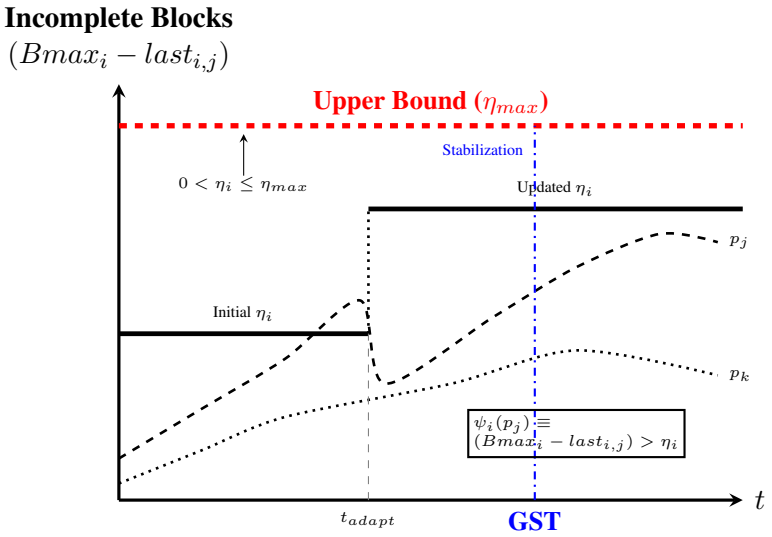

Figure 3. Progression of logical lags showing the adaptation of η_i to transient delays before GST, maintained within the safety boundary η_{max}.

The Cartesian Progression (Figure 3) illustrates how logical lags—defined as the distance in logical time between $Bmax_i$ and $last_{i,j}$—evolve relative to physical time.

The y-axis tracks fluctuations in the processes p_j and p_k as perceived by p_i. Initially, the system employs a conservative threshold η_i. Before the Global Stabilization Time (GST), transient delays may cause a process's lag to exceed this value (as seen with p_j at t_{adapt}), momentarily triggering a suspicion. However, the subsequent arrival of a delayed message proves p_j is functional, activating the Adaptive Threshold Mechanism. This causes η_i to increment ($\eta_i \leftarrow \eta_i + 1$) to accommodate the observed transient desynchronization. This adaptation is strictly bounded by η_{max}, a limit derived from physical memory constraints and application-specific consistency requirements. Once the system reaches GST, the relative logical speeds stabilize within the eventual Γ bound. As long as this stabilized logical lag remains below the η_{max} ceiling, the ERS model maintains both safety and liveness without requiring manual calibration.

5.2. The Gearing between T and η_i

In practical deployments, the watchdog timeout T defines the system's "logical clock rate". Because the ERS model's relative-speed bound Γ holds only after an unknown Global Stabilization Time (GST), the interplay between T and the adaptive threshold η_i dictates detection efficiency: (1) **Aggressive Watchdog (Small T):** Enables faster crash detection and keeps η_i small, but increases network overhead and the risk of false suspicions during pre-GST transient delays; and (2) **Conservative Watchdog (Large T):** Reduces bandwidth consumption, but causes η_i to converge to a higher value to accommodate larger logical "gaps", thereby increasing detection latency.

5.3. Convergence and the Cost of Adaptation

Prior to GST, the ERS detector undergoes an active adaptation phase characterized by three dynamics: (1) **False Suspicions:** Transient pre-GST message delays can cause logical gaps to exceed η_i, triggering false suspicions; (2) **Threshold Inflation:** Retracting these suspicions triggers an adaptation (Algorithm 1, Line 15), monotonically incrementing η_i ($\eta_i \leftarrow \eta_i + 1$) to accommodate the observed transient delays; and (3) **Stabilization:** Post-GST, relative logical speeds stabilize strictly within the finite Γ bound. Consequently, η_i stops growing, allowing the detector to achieve Eventual Strong Accuracy.

5.4. Future Practical Extensions

For future implementations, one might consider integrating this detector with a **Group Membership Service**. A group consensus protocol could act as a secondary validator; for instance, a single "too fast" process p_i would be unable to force a view change if the majority of the group provides evidence that the suspected process p_j is still making relative logical progress. Such mechanisms would further enhance the robustness of the ERS model in highly volatile or heterogeneous distributed environments.

6. Conclusions

In this work, we have formalized the **Eventual Relative-Speed (ERS)** model. By defining system progress in terms of relative logical steps rather than physical units, the ERS model offers a resilient framework for distributed environments where processors and communication channels may fluctuate in speed.

Our **Watchdog-Based Causal Progress Protocol** and **Adaptive Failure Detector** allow processes to maintain a "logical pulse" during idle periods and dynamically

adapt to transient delays and desynchronization before the Global Stabilization Time (GST) through a threshold-inflation mechanism. We demonstrated, through formal proofs, that this protocol preserves the fundamental causal properties $pr1$ and $pr2$ and eventually eliminates false suspicions among correct processes, achieving the $\diamond\mathcal{P}$ properties once the relative speeds stabilize within the Γ bound.

While this article focuses on the theoretical foundations and algorithmic correctness of the ERS model, it opens several avenues for further research. Future work will evaluate the quality of service (QoS) associated with the proposed detector across varied execution scenarios and configurations, specifically regarding the watchdog period T and the adaptive threshold η_i. Furthermore, we intend to investigate the integration of these logical-time detectors with group membership services to provide a complete, robust suite of agreement protocols.

7. AI Assistance Disclosure

The author used Gemini 1.5 for linguistic refinement, technical formatting of figures and text, and bibliographic organization. The author maintains sole responsibility for the conceptual development of the ERS model, the related algorithms, and correctness proofs; the AI served exclusively as a support tool for stylistic consistency and document structure.

References

Aguilera, M. K., Chen, W., and Toueg, S. (1997). Heartbeat: A timeout-free failure detector for quiescent reliable communication. In *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG '97)*, pages 126–140. Springer-Verlag.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

Chen, W., Toueg, S., and Aguilera, M. K. (2002). On the quality of service of failure detectors. *IEEE Trans. Comput.*, 51(1):13–32.

de Araújo Macêdo, R. J. (1994). *Fault-tolerant group communication protocols for asynchronous systems*. Phd thesis, University of Newcastle upon Tyne.

de Araújo Macêdo, R. J. and e Lima, F. R. L. (2004). Improving the quality of service of failure detectors with snmp and artificial neural networks. In *Anais do XXII Simpósio Brasileiro de Redes de Computadores (SBRC 2004)*, Fortaleza, CE, Brazil. SBC.

de Araújo Macêdo, R. J., Ezhilchelvan, P. D., and Shrivastava, S. K. (1995). Flow control schemes for a fault-tolerant multicast protocol. In *Proceedings of the 1995 Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS '95)*, pages 15–21, Newport Beach, California, USA. IEEE Computer Society.

de Araújo Macêdo, R. J. (2000). Failure detection in asynchronous distributed systems. In *Anais do II Workshop de Testes e Tolerância a Falhas*, pages 76–81. SBC.

de Araújo Macêdo, R. J. (2025). Synchronous partitioning and its effects on fault tolerance. In *Anais do XXVI Workshop de Testes e Tolerância a Falhas*, pages 43–56, Porto Alegre, RS, Brasil. SBC.

de Araújo Macêdo, R. J. and Freitas, A. E. (2009). A generic group communication approach for hybrid distributed systems. In *9th IFIP International Conference on Distributed Applications and Interoperable Systems (LNCS, Springer)*.

Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323.

Ezhilchelvan, P. D., Macêdo, R. A., and Shrivastava, S. K. (1995). Newtop: a fault-tolerant group communication protocol. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, ICDCS '95, USA. IEEE Computer Society.

Falai, L. and Bondavalli, A. (2005). Experimental evaluation of the qos of failure detectors on wide area network. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 624–633.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

Freiling, F. C., Guerraoui, R., and Kuznetsov, P. (2011). The failure detector abstraction. *ACM Comput. Surv.*, 43(2).

Kenwright, L., Roop, P., Allen, N., Cascaval, C., and Malik, A. (2025). Timetide: A programming model for logically synchronous distributed systems. *ACM Trans. Embed. Comput. Syst.*, 24(5s).

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401.

Macêdo, R. A., Ezhilchelvan, P., and Shrivastava, S. K. (1993). Modeling group communication using causal blocks. In *Proceedings of the 5th European Workshop on Dependable Computing*, Lisbon, Portugal.

Raynal, M. (2005). A short introduction to failure detectors for asynchronous distributed systems. *SIGACT News*, 36(1):53–70.

Robinson, P. and Schmid, U. (2011). The asynchronous bounded-cycle model. *Theoretical Computer Science*, 412(40):5580–5601.

Sá, A. S. and de Araújo Macêdo, R. J. (2010). Qos self-configuring failure detectors for distributed systems. In *Distributed Applications and Interoperable Systems (DAIS 2010)*, volume 6115 of *LNCC*, pages 126–140, Amsterdam, Netherlands. Springer.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

Spirakis, P. and Tampakas, B. (1988). Efficient distributed algorithms by using the archimedean time assumption. In Cori, R. and Wirsing, M., editors, *STACS 88*, pages 248–263, Berlin, Heidelberg. Springer Berlin Heidelberg.

Veríssimo, P. and Casimiro, A. (2002). The timely computing base model and architecture. *IEEE Trans. Comput.*, 51(8):916–930.

Widder, J. and Schmid, U. (2009). The Theta-model: achieving synchrony without clocks. *Distributed Computing*, 22(1):29–47.

Zhou, Y., Peng, S., Lyu, H., Tong, F., Huang, C., and Niu, J. (2025). Klotski: Towards consensus enabled collaborative vehicles in intelligent transportation. *IEEE Transactions on Vehicular Technology*, 74(12):18620–18634.