Never Say Never
Probabilistic & Temporal Failure Detectors

Dacfey Dzung
ABB Corporate Research
Baden, Switzerland
dacfey.dzung@ch.abb.com

Rachid Guerraoui
IC, EPFL
Lausanne, Switzerland
rachid.guerraoui@epfl.ch

David Kozhaya
IC, EPFL
Lausanne, Switzerland
david.kozhaya@epfl.ch

Yvonne-Anne Pignolet
ABB Corporate Research
Baden, Switzerland
yvonne-anne.pignolet@ch.abb.com

***Abstract*—The failure detector approach for solving distributed computing problems has been celebrated for its modularity. This approach allows the construction of algorithms using abstract failure detection mechanisms, defined by axiomatic properties, as building blocks. The minimal synchrony assumptions on communication, which enable to implement the failure detection mechanism, are studied separately. Such synchrony assumptions are typically expressed as eventual guarantees that need to hold, after some point in time, *forever* and *deterministically*. But in practice, they never do. Synchrony assumptions may hold only probabilistically and temporarily.**

In this paper, we study failure detectors in a realistic distributed system $\mathcal{N}$, with asynchrony inflicted by probabilistic synchronous communication. We address the following paradox about the weakest failure detector to solve the consensus problem (and many equivalent problems), i.e., $\diamond\mathcal{S}$: an implementation of "consensus with probability 1" is possible in $\mathcal{N}$ without using randomness in the algorithm itself, while an implementation of "$\diamond\mathcal{S}$ with probability 1" is impossible to achieve in $\mathcal{N}$. We circumvent this paradox by introducing a new failure detector $\diamond\mathcal{S}^*$, a variant of $\diamond\mathcal{S}$ with probabilistic and temporal accuracy. We prove that $\diamond\mathcal{S}^*$ is implementable in $\mathcal{N}$ and we provide an optimal $\diamond\mathcal{S}^*$ implementation. Interestingly, we show that $\diamond\mathcal{S}^*$ can replace $\diamond\mathcal{S}$, in several existing deterministic consensus algorithms using $\diamond\mathcal{S}$, to yield an algorithm that solves "consensus with probability 1". In fact, we show that such result holds for all *decisive* problems (not only consensus) and also for failure detector $\diamond\mathcal{P}$ (not only $\diamond\mathcal{S}$). The resulting algorithms combine the modularity of distributed computing practices with the practicality of networking ones.

***Keywords*-failure detection; probabilistic links; consensus;**

I. Introduction

The failure detector abstraction is an elegant means to solve difficult distributed computing problems, such as the fundamental consensus problem[1], in a modular manner [2]. Roughly speaking, a failure detector is a distributed "oracle" augmented to an asynchronous system. The purpose of this oracle is to provide hints (possibly incorrect) about which processes of the system have crashed [3]. A failure detector is formally defined by high-level axiomatic properties which, in turn, encapsulate synchrony assumptions that allow problems like consensus to be solved. The task of implementing a given failure detector using synchrony assumptions becomes a separate, lower-level task.

A large body of work [4]–[9] has been devoted to determine which synchrony assumptions are sufficient to implement for instance the $\diamond\mathcal{S}$ failure detector, established to be the weakest, in a precise sense [10], to solve consensus-like problems. The underlying synchrony assumptions, typically adopted in the distributed community to implement $\diamond\mathcal{S}$ take for example the form of some links being eventually timely; i.e., after some point in time, these links never "delay" messages. Such assumptions, besides placing the failure detector approach under scrutiny in the distributed computing community itself [11], are questionable from the networking perspective. We elaborate further on this.

Questioning Failure Detectors. As discussed by Dwork et al. in [12] as well as Lamport in [1], in practice, consensus requires the system to be "good" only sufficiently long, while the weakest failure detector to implement consensus requires parts of the system after some point to be "good" forever [4]–[9]. This controversy was raised and discussed by Charron-Bost et al. [11], who even suggested to abandon failure detectors and seek a better computing model.

From a networking perspective, the dependence of failure detectors on a "deterministic forever" synchrony condition, suggests that failure detectors may be practically unfit. This results from the fact that typical synchrony guarantees provided by networks are, at best, probabilistic. In fact, a considerable amount of research on packet transmissions confirms that synchrony guarantees in various kinds of networks, e.g., wired power line networks [13], hybrid wired/wireless networks [14] and wireless networks [15]–[17], are indeed probabilistic[2]. Consequently, failure detectors, like $\diamond\mathcal{S}$, might not be possible to implement in such networks and thus algorithms designed on top of such failure detectors cannot be practically used. We argue that this in fact is not the case!

Motivation and Approach. The motivation of this work is to bridge the gap between the modular distributed computing approach, based on failure detectors to circumvent the

[1]Consensus [1] is an essential building block of most distributed computing problems and applications such as leader election, state machine replication, atomic commit, etc.

[2]In practice, messages can be delayed at any point in time, for example as a result of bad transmission quality of the underlying channel [15], [16] or unpredictable loads on the system.

impossibility of fundamental distributed problems, and the networking view of communication link characteristics. In this view, link characteristics are probabilistic and temporary, rather than deterministic and perpetual.

We consider $\mathcal{N}$, a fully connected distributed system with probabilistic synchronous communication, i.e., links can probabilistically lose messages, but otherwise respect a known bound on the maximum transmission time. Such losses inherently inflict *asynchrony*, as it can take arbitrarily long until a reliable/successful message transmission between any two processes happens. As opposed to the partial synchrony typically considered in distributed computing, $\mathcal{N}$ does not require the delay of a reliable/successful message, after some point in time, to satisfy a specific bound forever. Solutions on the basis of our system $\mathcal{N}$ are thus applicable to a wide range of real networked systems.

Paradoxically, it is possible to have an implementation of "consensus with probability 1" in $\mathcal{N}$ without the use of randomisation in the algorithm itself[3], i.e., by merely relying on the non-determinism in communication, e.g., [18]. However, we prove that an implementation of "$\diamond\mathcal{S}$ with probability 1" is impossible to achieve in system $\mathcal{N}$ (see Theorem 1 for the meaning of "$\diamond\mathcal{S}$ with probability 1"). This suggests two things: (i) $\diamond\mathcal{S}$ is somehow too strong for consensus[4], at least in a probabilistic environment as $\mathcal{N}$ and (ii) deterministic algorithms solving consensus using $\diamond\mathcal{S}$, cannot be practically put in use to solve consensus in systems like $\mathcal{N}$, as $\diamond\mathcal{S}$ itself cannot be guaranteed in $\mathcal{N}$. This "consensus/failure detector" paradox can be viewed as a re-affirmation of the paradox highlighted in [11], this time in a non-deterministic environment. However, in contrast with [11], we follow a different route.

Whereas [11] suggested to abandon the failure detector approach, we propose instead a way to circumvent the paradox by refining the failure detector notion itself, while preserving its usefulness as an algorithmic building block. We define a probabilistic failure detector $\diamond\mathcal{S}^*$. Roughly speaking, $\diamond\mathcal{S}^*$ requires, with probability 1, that the properties of $\diamond\mathcal{S}$ are satisfied for periods that can be arbitrary long. Interestingly, we prove that $\diamond\mathcal{S}^*$ can be implemented in $\mathcal{N}$, which implies, at least from a failure detector perspective, that our system $\mathcal{N}$ is weaker than the systems considered so far to build $\diamond\mathcal{S}$-like failure detectors [4]–[9]. More importantly, we show that the celebrated rotating coordinator algorithm of [3] to solve consensus using $\diamond\mathcal{S}$, actually solves "consensus with probability 1" in $\mathcal{N}$, when $\diamond\mathcal{S}^*$ is used instead of $\diamond\mathcal{S}$. We then generalize this result, in three directions: (i) to hold for all deterministic consensus algorithms satisfying some $\diamond\mathcal{S}^{\mathcal{N}}$-bounded condition (Section 3) (not only the rotating coordinator algorithm), (ii) to hold for all *decisive problems* (Section VI), not only consensus and (iii) to hold for other failure detectors, e.g., $\diamond\mathcal{P}$.

In particular, we show that any deterministic algorithm, which solves a *decisive problem* using $\diamond\mathcal{S}$ ($\diamond\mathcal{P}$) and is $\diamond\mathcal{S}^{\mathcal{N}}$($\diamond\mathcal{P}^{\mathcal{N}}$)-bounded, can be re-used in system $\mathcal{N}$ to solve the same problem, ensuring termination with probability 1: the result is reached by using $\diamond\mathcal{S}^*$ ($\diamond\mathcal{P}^*$) instead.

Contributions. To summarize, our main contributions are: **1.** We propose a way to circumvent the "consensus/failure detector" paradox in systems with probabilistic synchronous communication ($\mathcal{N}$): we define $\diamond\mathcal{S}^*$ a probabilistic failure detector with accuracy ensured for periods that can be arbitrary long. $\diamond\mathcal{S}^*$ can be implemented in systems like $\mathcal{N}$. **2.** We show that $\diamond\mathcal{S}^*$ can be implemented efficiently and we present an optimal implementation of it, which we believe is interesting on its own. The optimal implementation hinges on a logical linear arrangement of the processes. When links behave in a timely manner (i.e., the delay of a reliable message transmission respects some bound), the number of links carrying messages infinitely often converges to using $\mathcal{C}$ (the number of correct processes), possibly $\mathcal{C}-1$. In the best case, i.e., $\mathcal{C}-1$, our implementation of $\diamond\mathcal{S}^*$ achieves, to the best of our knowledge, the lowest communication overhead compared to all known $\diamond\mathcal{S}$ implementations.

3. For all decisive problems P, beyond consensus (see Section VI-B for decisive problems), we enable existing deterministic protocols which use $\diamond\mathcal{S}$ and $\diamond\mathcal{P}$ and which are $\diamond\mathcal{S}^{\mathcal{N}}$($\diamond\mathcal{P}^{\mathcal{N}}$)-bounded, to be reused in $\mathcal{N}$ to solve "P with probability 1": simply using $\diamond\mathcal{S}^*$ and $\diamond\mathcal{P}^*$ instead of $\diamond\mathcal{S}$ and $\diamond\mathcal{P}$ respectively leads to the desired result. In this sense, our approach succeeds in encapsulating the randomization of the probabilistic link behaviour in the very abstraction of failure detection (without affecting the deterministic algorithms built on top), bridging the gap between distributed computing and networking practices.

Roadmap. Section II discusses related work. Section III presents our distributed system $\mathcal{N}$. Section IV proves the impossibility of implementing "$\diamond\mathcal{S}$ with probability 1" and introduces $\diamond\mathcal{S}^*$. Section V derives communication lower bounds for implementing $\diamond\mathcal{S}^*$ and presents an optimal implementation of $\diamond\mathcal{S}^*$. In Section VI, we show that an existing consensus algorithm with $\diamond\mathcal{S}$ can guarantee "consensus with probability 1" in system $\mathcal{N}$ when $\diamond\mathcal{S}^*$ is used instead. We then generalize our result to a larger set of problems. Section VII summarizes our results and concludes the paper. For presentation simplicity, we defer some proofs and discussions to a dedicated Appendix.

[3]Solving "consensus with probability 1" means that it is possible to devise a protocol that guarantees all safety properties of consensus deterministically but guarantees termination with probability 1 (see Section VI-A for all properties of consensus). In turn, this implies that the probability that the algorithm does not terminate decreases with time.

[4]It is important to recall that $\diamond\mathcal{S}$ is the weakest, *amongst all failure detectors*, to solve consensus [10]. This does not mean however that $\diamond\mathcal{S}$ is equivalent to consensus in a computability sense: one cannot implement $\diamond\mathcal{S}$ from consensus.

II. Related Work

Fault-tolerance has been addressed in many domains and at many levels [19]–[21], for example, to predict failures and figure out their sources and patterns [22]–[26], to detect transient process failures [27], etc. We discuss, in more details, closely related work addressing specifically failure detectors and consensus.

Minimal Synchrony. A large body of work studied Ω, a failure detector abstraction equivalent to $\diamond\mathcal{S}$ [28], and its implementation under different synchrony assumptions [4]–[6], [8], [9]. These implementations either posed assumptions on the behaviour of correct and faulty processes or required links (or a subset of links) to be reliable, fair and/or eventually timely. See Table I for a summary of the assumptions adopted by systems implementing Ω or $\diamond\mathcal{S}$.

In contrast, we focus on $\mathcal{N}$, a system which, from the failure detection point of view, is weaker than the systems considered above. In fact, we prove that $\diamond\mathcal{S}$ properties cannot be guaranteed, with probability 1, in $\mathcal{N}$. We investigate properties of failure detectors that are implementable in $\mathcal{N}$.

Minimal Communication. Another track of research addressed the communication overhead of failure detector implementations [5], [29]–[32]. Aguilera et al. [5] showed that an implementation of Ω in a system S (see Table I) requires all processes to periodically send messages and the minimum number of links carrying messages forever can be at least $(n^2-1)/4$. On the other hand, stronger systems S^+ and S^{++} (Table I) allow *efficient* implementations, where only one process broadcasts messages forever on $n-1$ links. For systems with eventually timely correct processes and links (also reliable), Larrea et al. [31], [33] provided algorithms for $\diamond\mathcal{S}$ and $\diamond\mathcal{P}$, where $2n$ links carry messages forever in the worst case, and for Ω where $n-1$ links carry messages forever respectively. Follow-up work [29], [30] defined communication optimality: in systems with at least one faulty process, the number of correct processes $\mathcal{C}$ equals the minimum number of links necessary to implement $\diamond\mathcal{P}$, $\diamond\mathcal{S}$ and Ω. If the correct process with the smallest id has eventually timely output links, communication-optimal implementations of $\diamond\mathcal{S}$ and $\diamond\mathcal{P}$ exist [32].

Our work is different in that it investigates the communication overhead of $\diamond\mathcal{S}^*$, a weaker variant of $\diamond\mathcal{S}$, in system $\mathcal{N}$, where links never become timely forever. We show that when links in $\mathcal{N}$ start behaving in a timely fashion for some interval, the number of links carrying messages in our implementation of $\diamond\mathcal{S}^*$ will converge to $\mathcal{C}$, possibly $\mathcal{C}-1$. Thus in the best case (i.e., $\mathcal{C}-1$) our implementation of $\diamond\mathcal{S}^*$ in $\mathcal{N}$, where $n-1$ processes can crash, circumvents the bound for $\diamond\mathcal{S}$ when at least one process crashes using $\mathcal{C}$ links [30]. Our algorithm is inspired by the ring structure algorithms [30] for communication-optimal implementations of $\diamond\mathcal{P}$. However, our implementation is different as it assumes a linear arrangement of processes and manages suspicions differently to accommodate for properties of $\diamond\mathcal{S}^*$.

Table I: Assumptions of systems discussed in Section II.

System	Properties
$\mathcal{N}$ *(this paper)*	*All links* lose messages probabilistically and fairly *Propagation delay*, in case of no loss, is bounded *Processes* can crash and processing delays are negligible w.r.t. communication delay
S [5]	*Processes* can be arbitrarily slow and can crash, but have a max. execution speed *Links* can be arbitrarily slow and lossy with at least one eventually timely source (i.e., a timely correct process whose *output* links are eventually timely)
S^+ [5]	S with at least one correct process whose *input and output* links are fair
S^{++} [5]	S and such that all links are fair
[29]–[31]	*All links* are reliable and eventually timely
[8]	*All links* are reliable. Some correct process hears within interval δ from f other processes (f is the maximum number of processes that can crash)
[4]	*All links* are fair and there is one correct process with f output links eventually timely
[9]	*Links* can lose or delay messages At least one correct process that can reach all other correct processes through eventually timely links

Omission Faults. Some researchers explored consensus in systems with message losses without relying on eventual guarantees. Santoro and Widmayer [34] showed that consensus is impossible if $n-1$ of the n^2 possible messages sent in a round can be lost. In contrast, Schmid et al. [35] showed that consensus can be solved even in the presence of $O(n^2)$ moving omission and/or arbitrary link failures per round, provided that both the number of affected outgoing and incoming links of every process is bounded and that all processes are correct. Soraluze et al. [36] considered the general omission model, where processes can fail either by permanently crashing or by omitting messages.They defined a failure detector requiring the existence of a majority of *well-connected* processes, which do not crash, and are able to communicate in both directions and without omissions, either directly or indirectly, with a majority of processes.

In contrast, our work does not bound the number of messages lost at any point in time and does not require any process to be connected at all times with any other process, yet we can implement failure detectors and consensus.

Randomized Consensus Algorithms. Randomized algorithms ensuring "consensus with probability 1" have also been explored. Approaches based on *coin-flips* [37]–[39] or *probabilistic schedulers* [18] lead to consensus algorithms with probabilistic factors. In systems with dynamic communication failures, multiple randomized algorithms [40], [41] addressed the *k-consensus* problem, which requires only k processes to eventually decide. Moniz et al. [40] considered a system with correct processes and a bound on the number of faulty transmission. In a wireless setting, where multiple processes share a communication channel, Moniz et al. [41] devise an algorithm tolerating up to f Byzantine processes

and requires a bound on the number of omission faults affecting correct processes.

Our work does not employ randomization in the algorithm, we focus on deterministic algorithms in probabilistic networks. Moreover, instead of designing new consensus algorithms, we re-use existing deterministic algorithms relying on failure detectors to solve "consensus with probability 1" in $\mathcal{N}$.

Failure Detectors with Probablistic Guarantees. A different line of research explored failure detector implementations with probabilistic guarantees. Chen et al. in [42] studied the quality of service (QoS) of failure detectors in systems where message delays and message losses follow probability distributions. They proposed a set of metrics, among them (i) how fast actual failures are detected and (ii) how well false detections are avoided. In [43], Bertier et al. proposed an implementation supporting a short detection time based on estimations of the expected arrival of monitoring messages, and on adapting QoS to the specific application needs. In [44], Gupta et al. quantified the optimal network load of failure detector algorithms as a function of the failure detection time and the probability of falsely suspecting a correct process. Hayashibara et al. [45] proposed φ-failure detectors capable of adapting to the application requirements and network conditions dynamically, by assigning a value to every known process representing the confidence that it is alive.

In contrast, we evaluate the eventual guarantees of (binary) failure detectors implemented in systems with probabilistic message losses. In particular we investigate implications of such guarantees to solving distributed computing problems, namely consensus. We also study the efficiency of implementing these failure detectors from a communication overhead perspective, defining optimality in terms of the number of links that need to carry messages forever rather than evaluating the real-time performance.

Rethinking Failure Detection. Several researchers challenged (directly or indirectly) the failure detection approach. Biely et al. [46] showed that the asynchronous model augmented with Ω is equivalent to several models where the links from at least one process (the source) are timely. In comparison with Biely et al. [46], our work avoids the necessity of eventually timely link(s), for the solvability of problems such as consensus in the presence of asynchrony.

Charron-Bost et al. [11] highlighted a paradox about the failure detection approach which we re-affirm, in a non-deterministic environment, in this paper: Dwork et al. [12] and Lamport [1] showed that sufficiently long finite "good" periods make consensus solvable while Chandra et al. [10] show that consensus cannot be solved without permanent agreement on a leader from some time on. Given this inconsistency in results, Charron-Bost et al. [11] showed that the "discrepancy" is due to the two-layered structure of the failure detector approach itself. Precisely, authors attribute this "artificial difficulty" to the interface between the failure detector layer and the asynchronous system layer to which the failure detector is augmented and the lack of timing control by failure detectors on the asynchronous system. Charron-Bost et al. [11] concluded that it may be better to look at consensus without using failure detectors. Avoiding any dependence on real time, Cornejo et al. [47] proposed *asynchronous failure detectors* (AFDs) and used them to address challenges related to the hierarchy robustness of failure detectors. They also investigated the relationship between the weakest failure detector and partial synchrony. They showed that a large class of problems, termed as *finite problems*, such as consensus, do not encode the same information about process crashes as their weakest failure detectors do (another validation of the observation in [11]).

In this paper, we take an opposite route compared to [11] and [47]. We refine the notion of a failure detector with explicit dependence on real time to address a similar paradox as that of [11]: "consensus with probability 1" can be implemented in system $\mathcal{N}$ (without the need for randomization within the algorithm) while "$\diamond\mathcal{S}$ with probability 1" is impossible to implement in $\mathcal{N}$. We define $\diamond\mathcal{S}^*$, a variant of $\diamond\mathcal{S}$ implementable in $\mathcal{N}$ and show that such a transformation in the failure detector notion allows deterministic consensus algorithms based on failure detectors to solve "consensus with probability 1" in poorly behaved systems. In this sense, we succeed to repress the "artificial difficulty" highlighted in [11] about the failure detector model, showing that "consensus with probability 1" can be solved in poorly behaving systems (as viewed from a networking perspective) using failure detectors, without requiring an eventually forever agreement on a process that will never crash.

III. System Model

We consider a distributed system $\mathcal{N}$ consisting of a finite set Π of $n > 1$ processes, $\Pi = \{p_1, p_2, ..., p_n\}$, which communicate by message passing. We assume, without loss of generality, that processes have access to a global clock with discrete time events denoted by $t : \{1, 2, 3, ...\}$ (in Appendix B we show that a global clock can be substituted by local clocks that do not have to be synchronized).

Processes can send and/or receive a message, at these discrete time events. The time interval between consecutive events in t is assumed to be an upper bound on the propagation delay (t_{pg}) over any link connecting any two processes. Processing delays are assumed to be negligible compared to communication delays.

Communication Links. The links interconnecting processes are assumed to be uni-directional uni-cast links. In particular, every pair of processes (p_i, p_j) is connected by two uni-directional links: l_{ij} and l_{ji}. These links exhibit changes in their transmission quality, as the quality of the underlying channels might depend on various propagation conditions. We thus assume that a link l_{ij} has a probability

$0 < P_{ij}(t) < 1$ of losing the message sent at time t (if any is sent). This captures the very idea that a link is not always reliable and can lose messages for an unbounded but finite period[5]. The value of $P_{ij}(t)$ can change with time; at each time $t : \{1, 2, 3, ...\}$, $P_{ij}(t)$ may have any value in $(0, 1)$. We refer to such links as *probabilistic*. A probabilistic link thus constitutes an instance of the *fair-loss* link [48], where a message sent infinitely often is received infinitely often.

Faulty Processes. Processes can fail by crashing, i.e., by halting prematurely. We consider that process crashes are permanent, i.e., no crashed process can recover. We use $\mathcal{C}$ to denote the number of *correct* processes, i.e., processes that do not fail. We assume that $\mathcal{C} \geq 1$. When a process p_i sends a message m to process p_j (for $i \neq j$) and m is successfully propagated by link l_{ij}, p_j receives/delivers m. However if m is lost by link l_{ij}, p_j receives nothing.

Reliable Message Transmission. Despite probabilistic losses, a reliable message transmission, be it a unicast or a broadcast can still be achieved in $\mathcal{N}$ [18], [49], [50]. Reliable transmissions can be provided via abstractions running on top of the probabilistic links of system $\mathcal{N}$. For example, a reliable link abstraction would guarantee the following with probability 1: a message sent by a correct process p_i to another correct process p_j, will be delivered by p_j at some future point in time[6]. We provide next the complete specifications of a reliable link and a reliable broadcast primitive in $\mathcal{N}$. A reliable link abstraction guarantees:

1) *Reliable delivery:* If a correct process p sends a message m to a correct process q at time t, then with probability 1 the following is guaranteed: there exists some time $t' > t$ where q delivers m. In particular, q can deliver m at $t + T$ with positive probability $\forall\ T\ \in\ \{1, 2, 3, ...\}$.
2) *No duplication:* No message is delivered by a process more than once.
3) *No creation:* If some process q delivers a message m with sender p, then m was previously sent to q by p.

A reliable broadcast primitive can also be defined in $\mathcal{N}$:

1) *Validity:* If a correct process p broadcasts a message m at time t, then with probability 1 the following is guaranteed: at some time $t' > t$ all correct processes will have delivered m. Precisely, a correct processes can deliver m at $t + T$ with positive probability $\forall\ T\ \in\ \{2, 3, ...\}$.
2) *No duplication:* As for a reliable link.
3) *No creation:* If some process q delivers a message m with sender p, then m was previously broadcast by p.
4) *Agreement:* If a message m is delivered by some process at time t, then with probability 1 the following is guaranteed: at some time $t' \geq t$, m will be delivered by every correct process.

The reliable link abstraction can be achieved over a probabilistic link l_{ij}, for example by deploying buffers and message retransmissions [18], [49], [50]. Typically, process p_i keeps retransmitting a message m forever or in practice until some acknowledgement is obtained for m. In this sense, the message transmission delay of a message m is measured by the number of time slots elapsed from p_i's first attempted transmission of m until the time when m is successfully received by p_j. We do not elaborate on implementation details of such an abstraction, as existing work already addresses this problem in systems where links lose messages [49], [50]. Clearly, such a reliable link abstraction does not provide any deterministic bounds on message transmission delays, as message losses may span unbounded duration. However the reliable link abstraction in $\mathcal{N}$ offers instead a probability distribution on the delay of a message.

The reliable broadcast primitive can be built for example using the reliable link abstraction in system $\mathcal{N}$. Algorithms such as those detailed in [51] can be directly applied and will result also in a reliable broadcast where the delay to deliver a broadcast message, despite possibly being arbitrarily long, admits a probability distribution.

IV. Probabilistic Temporal Failure Detection

In a system augmented with a failure detector, each process has access to a local failure detector module [3]. This module monitors other processes in the system and typically maintains a set of those that it currently suspects to have crashed. Chandra and Toueg [3] defined various kinds of failure detectors based on their achievable properties, namely *completeness* and *accuracy*. Roughly speaking, the completeness property describes the failure detector's ability to suspect crashed processes, while the accuracy property defines the failure detector's ability of not suspecting correct processes. For instance, the $\diamond\mathcal{S}$ failure detector guarantees the following two properties: (i) *strong completeness:* eventually every process that crashes is permanently suspected by every correct process and (ii) *eventual weak accuracy:* there is some time instant $t_G \in \{1, 2, 3, ...\}$ after which some correct process is never suspected by any correct process.

A. The Impossibility of Eventual Weak Accuracy

We establish several lemmata ensuring certain guarantees on message delivery and process failures.

Lemma 1. *In system $\mathcal{N}$, for any finite period Δt and at any time instant $t_s \in \{1, 2, 3, ...\}$, all messages that are sent on*

[5]Note that $P_{ij}(t)$ is assumed to be strictly less than 1 and greater than 0 for presentation simplicity. Our theoretical results can be extended to the case where $0 \leq P_{ij}(t) \leq 1$, such that $0 < P_{ij}(t) < 1$ occurs infinitely often and that $P_{ij}(t) = 0$ or $P_{ij}(t) = 1$ occur for some bounded duration.

[6]**CRUCIAL**: The phrase "the following is guaranteed with probability 1: there is a time when event X occurs" does not mean that there is a point in time where event X occurs with probability 1 but rather that over the infinite course of time event X will certainly occur. For example, if a fair coin is flipped an infinite number of times, then with probability 1, there is a time when the coin lands on head, however there is no point in time where flipping the coin lands on head with probability 1.

link l_{ij} during the interval $t_s + \Delta t$, can be lost with positive probability.

Proof: Recall that $P_{ij}(t)$ is the probability with which the link l_{ij} loses a message at time t. Let $P_{ij}(t \cap t')$ be the probability that l_{ij} loses the messages (if any is sent) at time t and time t'. Since $0 < P_{ij}(t) < 1 \ \forall t$, then

$$0 < P_{ij}(t) = \frac{P_{ij}(t \cap t')}{P_{ij}(t'|t)} < 1 \ \forall \ t', \ t. \tag{1}$$

By (1), $P_{ij}(t'|t) > 0$ (and $0 < P_{ij}(t \cap t') < 1$). By induction, we have $P_{ij}(t'|t, t+1, ..., t'-1) > 0 \ \forall \ t' > \ t$. Denote by $\mathsf{B}(t)$ the event that l_{ij} losses all messages (if any is sent) for the interval $t + \Delta t$, for any finite period Δt. Then the probability of $\mathsf{B(t)}$ happening is:

$$\begin{aligned} Pr(\mathsf{B(t)}) &> P_{ij}(t \cap t+1 \cap t+2 \cap ... \cap t+\Delta t) \\ &= P_{ij}(t) \times P_{ij}(t+1|t) \times ... \times P_{ij}(t+\Delta t|t, t+1, ..., t+\Delta t-1) > 0. \end{aligned}$$

Given $P_{ij}(t) < 1$, then we have $0 < Pr(\mathsf{B(t)}) < 1$. ■

Lemma 2. *Consider any finite period Δt, any time instant $t_s \in \{1, 2, 3, ...\}$ and any subset of processes ΔP. In system $\mathcal{N}$, all processes $\notin \Delta P$ can lose, with positive probability, all messages sent in the interval $t_s + \Delta t$ from and to processes in ΔP.*

Proof: The probability to have any subset of processes losing all messages exchanged with all remaining processes for any finite period Δt, depends on the individual probabilities of the relative individual links losing all sent messages during the interval Δt. Following from Lemma 1, any link in the system can drop all messages (if any were sent) for a finite but unbounded time, with a positive probability. Denote by $\mathsf{B_{ij}}(t)$ the event that l_{ij} losses all messages (if any were sent) in the interval $t + \Delta t$, for any finite period Δt. Then by Lemma 1, $0 < Pr(B_{ij}(t)) < 1$. Following the arguments as in proof of Lemma 1, we have:

$$0 < Pr(\bigcap_{i,j \in [1,n]} B_{ij}(t)) < 1 \ \forall \ i, \ j \in [1, n],$$

where n is the total number of processes in the system. This concludes the proof. ■

Lemma 3. *In system $\mathcal{N}$, no correct process can determine with probability 1, at any point in time, that some other process in the system has crashed, i.e., for any finite period a correct process can be (with probability> 0) indistinguishable from a crashed one.*

Proof: Following from Lemma 2, a single process p can lose all messages exchanged with the whole network (i.e., from and to p) with positive probability for any finite time. Consider an execution $e1$ where p crashes at time t and another execution $e2$ where p loses all communication at time t for any finite period. Then executions $e1$ and $e2$ are indistinguishable to all processes (except p) for the whole time in which communication is lost, i.e., any finite time by Lemma 2. ■

With these lemmata, we can prove that $\diamond\mathcal{S}$ cannot be implemented in system $\mathcal{N}$.

Theorem 1. *It is impossible to implement "$\diamond\mathcal{S}$ with probability 1" in $\mathcal{N}$ even if at most one process can crash. That is, in the presence of processes crashes, it is impossible to have an algorithm in $\mathcal{N}$ that guarantees strong completeness deterministically and which ensures, with probability 1, the following: there is a time after which some correct process is never suspected by all correct processes.*

Proof: We proceed by contradiction. Without loss of generality, assume a system $\mathcal{N}$ of $n = 2$ processes, p_1 and p_2. Suppose that there exists an algorithm $\mathcal{A}$, that guarantees both strong completeness and eventual weak accuracy. Consider three executions: (i) $e1$: an execution where p_1 fails at some time instant t_s in $\{1, 2, 3, ...\}$, (ii) $e2$: an execution where p_2 fails at t_s and (iii) $e3$: an execution where p_1 and p_2 are both correct but all messages exchanged between p_1 and p_2 are lost during the interval $t_s + \Delta t$. By the strong completeness property of $\mathcal{A}$, in executions $e1$ and $e2$ there is a finite period, say $\Delta t'$, after which p_2 suspects p_1 and p_1 suspects p_2 respectively. By Lemma 2, execution $e3$ is a valid execution in $\mathcal{A}$ and Δt can be arbitrarily long, specifically $\Delta t \geq \Delta t'$. Thus, by the strong completeness of $\mathcal{A}$, p_1 suspects p_2 and p_2 suspects p_1 in $e3$. By Lemma 2, execution e_3, such that $\Delta t \geq \Delta t'$, can occur with positive probability at any time instant in $\{1, 2, 3, ...\}$. This implies that $\mathcal{A}$ cannot guarantee with probability 1 the following: there exists some time after which some correct process is *never* suspected (i.e., remains trusted forever) by any correct process. This violates the eventual weak accuracy of $\mathcal{A}$. ■

As a consequence of Theorem 1, we study how to vary the properties of $\diamond\mathcal{S}$, defining a variant $\diamond\mathcal{S}^*$, which is implementable in our system $\mathcal{N}$.

B. Probabilistic & Temporal Failure Detectors

We define a new probabilistic weak accuracy property that holds temporarily for periods that can be arbitrary long. Combined with strong bounded completeness, these two properties define a new failure detector $\diamond\mathcal{S}^*$.

Definition 1. *Failure detector $\diamond\mathcal{S}^*$ guarantees (i) strong bounded completeness: every process that crashes is permanently suspected by every correct process after a maximum of T_D time slots of the actual crash and (ii) probabilistic & temporal weak accuracy: Consider any finite duration Δt. With probability 1 the following occurs: there exists infinitely many time instants t_G, such that a unique correct process is not suspected by any correct process for the interval $t_G + \Delta t$.*

Theorem 2. *It is possible to implement $\diamond\mathcal{S}^*$, in system $\mathcal{N}$, assuming $n - 1$ processes can crash.*

Proof: Let t_s be any time instant in $\{1, 2, 3, ...\}$ and Δt be any finite duration.

Lemma 4. *There is a positive probability that all messages sent by a correct process to all other correct processes, in the interval $t_s + \Delta t$, are not lost (i.e., successfully received).*

Proof: Let $\mathsf{E_{ij}}(t_s)$ be the following predicate: All messages sent by a correct process p_i to a correct process p_j, in the interval $t_s + \Delta t$, are not lost, i.e., successfully received by p_j. The probability that predicate $\mathsf{E_{ij}(t_s)}$ occurs is: $0 < Pr(\mathsf{E_{ij}(t_s)}) < 1$. This can be easily deduced from the proof of Lemma 1, given that the probability of a message (sent from p_i to p_j) being not lost at any time instant $t_s \in \{1, 2, 3, ...\}$ is $0 < 1 - P_{ij}(t_s) < 1$. Let $\mathsf{E}(t_s)$ be the following predicate: All messages sent by a correct process p_i to every correct process $p_j \in \mathcal{C}$, in the interval $t_s + \Delta t$ are not lost. Since $0 < Pr(\mathsf{E_{ij}}(t_s)) < 1$, the probability of predicate $\mathsf{E}(t_s)$ happening, as in the proof of Lemma 2, is $Pr(\mathsf{E}(t_s)) = Pr(\bigcap_{j:\{p_j \in \mathcal{C}\}} E_{ij}(t_s)) > 0$. ∎

Assume algorithm $\mathcal{A}$ executing the following: (i) all processes periodically, at every time event $t = \{1, 2, ..., \infty\}$, broadcast messages (i.e., they send messages to all other processes in the system) and (ii) initially all processes trust (do not suspect) each other. At every time instant in $\{2, 3, 4, ...\}$, process p_i suspects another process p_j only if p_i receives no new message from p_j, otherwise p_i trusts p_j.

The strong bounded completeness of $\diamond\mathcal{S}^*$ is ensured by $\mathcal{A}$. A process that crashes at time instant $t_{crash} \in \{1, 2, 3, ...\}$ stops sending messages and thus by (ii) will be suspected at all times $> t_{crash} + 1$ by all correct processes forever (that is with $T_D = 1$). Let's denote by $\mathsf{E_p(t_s)}$ the following predicate: during the interval $t_s + \Delta t$, all messages sent by a correct process p are successfully received by all correct processes. By (ii) of algorithm $\mathcal{A}$, $\mathsf{E_p(t_s)}$ implies that process p is not suspected by any correct process during the interval $t_s + \Delta t$. Following from Lemma 4, the probability of observing $\mathsf{E_p(t_s)}$ is greater than zero. Note that in $\mathcal{A}$ any correct process can be selected as the unique correct process.

Since in $\mathcal{A}$ processes keep sending messages to all other processes forever (infinitely) and since for any time instant $t_s \in \{1, 2, 3, ...\}$ $P(\mathsf{E_p(t_s)}) > 0$, then with probability 1 the following is satisfied: there exists infinitely many time instants t_G when predicate $\mathsf{E_p(t_G)}$ happens. $\mathcal{A}$ thus guarantees the accuracy of $\diamond\mathcal{S}^*$, concluding the proof. ∎

In Appendix A, we discuss the implementability of other types of probabilistic failure detectors; namely $\mathcal{P}^*$, a probabilistic variant of the perfect failure detector $\mathcal{P}$, and $\diamond\mathcal{P}^*$, a probabilistic variant of $\diamond\mathcal{P}$. In this main part of the paper and for space limitations, we solely focus on $\diamond\mathcal{S}$ and $\diamond\mathcal{S}^*$.

V. $\diamond\mathcal{S}^*$ Bounds and Implementations

We study in this section the communication overhead of $\diamond\mathcal{S}^*$ and present an optimal implementation of it.

A. Lower Bounds

First, we identify the bounds on the number of processes and links required to respectively send and carry messages forever for any algorithm implementing $\diamond\mathcal{S}^*$.

Theorem 3. *Consider any algorithm $\mathcal{A}$ that implements $\diamond\mathcal{S}^*$ in system $\mathcal{N}$ of $n \geq 2$ processes, where $n-1$ processes can crash. Then, $\mathcal{C}-1$ distinct processes send messages infinitely often in $\mathcal{A}$ with probability > 0.*

Proof: Assume that t_s is any time instant in $\{1, 2, 3, ...\}$ and Δt is any finite duration. If no correct process sends messages infinitely often, i.e., all correct processes stop sending messages at some point in time, say t, then $\diamond\mathcal{S}^*$ cannot be implemented. This holds, since after time t every correct process becomes indistinguishable from a crashed process (w.r.t. to all other processes in $\mathcal{N}$). By the strong bounded completeness of $\diamond\mathcal{S}^*$, every correct process suspects all processes in $\mathcal{N}$ after some bounded duration. This violates the probabilistic eventual weak accuracy property of $\diamond\mathcal{S}^*$.

Thus to implement $\diamond\mathcal{S}^*$ in $\mathcal{N}$ some correct process(es) should send messages infinitely often. We now prove Theorem 3 by showing that in system $\mathcal{N}$ with $n \geq 2$ processes, where $n-1$ processes can crash, it is impossible to have with probability 1 an implementation of $\diamond\mathcal{S}^*$ where eventually, only $c : \{0 < c < \mathcal{C} - 1\}$ correct processes send messages infinitely often.

Consider $\bar{c}$ to be the subset of correct process that stop sending messages after time instant t_s and consider the following two executions: (i) $e1$: all processes in c crash at time instant $t_{crash} > t_s$ and (ii) $e2$: all messages exchanged between processes in c and processes in $\bar{c}$ in the interval $t_{crash} + \Delta t$ are lost. By Lemma 3, execution $e2$ is valid, as it has a positive probability of happening. For processes in $\bar{c}$ executions $e1$ and $e2$ cannot be distinguishable in any finite amount of time (since Δt is any finite duration). Therefore, after some time (T_D) processes in $\bar{c}$ suspect all processes in c. If no process in $\bar{c}$ starts to send a message afterwards then if all processes in c did crash no correct process in the system will send messages (a violation). Thus some process(es) in $\bar{c}$ should send messages, which in the case of $e2$, i.e., if processes in c are still alive, results in more than c process sending messages. Since execution $e2$ occurs with a positive probability, then it is impossible to guarantee with probability 1 that only $c : \{0 < c < \mathcal{C} - 1\}$ correct processes send messages infinitely often. This concludes the proof. ∎

N.B. Theorem 3 does not mean that each process sending messages infinitely often, needs to do so by broadcasting (i.e., by sending the message to all other processes in the system). A process may send messages to any subset of the processes in the system. We show now that Theorem 3 can be circumvented, in the sense that $\diamond\mathcal{S}^*$ algorithms can be

implemented such that, with probability 1, less than $\mathcal{C}-1$ processes send messages infinitely often. It can be done by limiting the maximum number of processes that can crash.

Theorem 4. *Given an algorithm $\mathcal{A}$ that implements $\diamond\mathcal{S}^*$ in $\mathcal{N}$ with $n \geq 2$ processes of which at most $f < \frac{n}{2} - 1$ processes may crash, then the number of processes sending messages infinitely often in $\mathcal{A}$ can be less than $\mathcal{C}-1$.*

Proof: Consider an algorithm $\mathcal{A}$ which deterministically selects any $f+1$ processes to keep sending messages infinitely often after some point in time to all processes in $\mathcal{N}$, while all other processes stop sending messages completely. Since the maximum number of processes that may fail is f, then $\mathcal{A}$ guarantees that at least one correct process will send messages infinitely often and at maximum $f+1$ will send messages infinitely often. By the proof of Theorem 2, it is clear that $\diamond\mathcal{S}^*$ can be implemented in $\mathcal{N}$ even if only one correct process sends messages, to all other processes in $\mathcal{N}$, infinitely often. This proves that $\mathcal{A}$ implements $\diamond\mathcal{S}^*$ such that at most $f+1$ processes send messages infinitely often. $f+1 < \frac{n}{2} < \mathcal{C}-1$ (since $\mathcal{C} \geq n-f$). ■

We now determine the number of links that need to carry messages infinitely often in algorithms implementing $\diamond\mathcal{S}^*$. Despite the asynchrony caused by probabilistic message loss, system $\mathcal{N}$ can, with positive probability, reach a point in time where links can be timely (i.e., ensure that the delay of a reliable message transmission respects some bound) for any finite duration. We define next what it means for algorithms to be optimal in $\mathcal{N}$. Let L_{min} be the minimum number of links required to carry messages forever to implement failure detector $\mathcal{X}$ in a synchronous system[7]. Let $\mathcal{A}$ be an algorithm that implements failure detector $\mathcal{X}$, then:

Definition 2. *$\mathcal{A}$ is optimal, if L, the number of links carrying messages infinitely often in $\mathcal{A}$, satisfies: $\lim_{\Delta t \to \infty} L = L_{min}$, where Δt is an interval in which links are timely.*

Theorem 5. *The minimum number of links which need to send messages forever to implement $\diamond\mathcal{S}^*$ in a synchronous system where $n-1$ processes may crash is $\mathcal{C}$ (possibly $\mathcal{C}-1$ depending on what processes crash).*

Proof: First we prove that it is impossible to implement $\diamond\mathcal{S}^*$ if $\mathcal{C}-2$ links send messages infinitely often. The proof is by contradiction. Assume an implementation $\mathcal{A}$ of $\diamond\mathcal{S}^*$ in which only $\mathcal{C}-2$ links carry messages forever. Then there is in $\mathcal{A}$ at least one correct process p which eventually (i.e., at some point t in time) does not exchange messages with any other correct process.

Assume an execution $e1$ of $\mathcal{A}$ with $\mathcal{C} > 1$ correct processes (including p) and another execution $e2$ of $\mathcal{A}$ similar to $e1$ however where p crashes after time t (the time when p eventually stops exchanging messages). $e1$ and $e2$ are indistinguishable to all processes (other than p) and thus processes in $e2$ will keep using the same number of links. However in $e2$ since the number of correct processes is less, then $\mathcal{C}-3$ links should be used which contradicts that $e1$ and $e2$ are indistinguishable.

Now assume an implementation $\mathcal{A}'$ of $\diamond\mathcal{S}^*$ in which only $\mathcal{C}-1$ links carry messages forever. Since there is $\mathcal{C}$ correct processes, such an implementation is only possible if correct processes are arranged in a tree topology (of which a star and a linear list are a special case). In such an arrangement the root of the tree sends heartbeat messages, indirectly, to the rest of the correct processes. Consider an execution $e1$ of $\mathcal{A}'$ in which $\mathcal{C}$ processes are correct and let t be the point in time after which only $\mathcal{C}-1$ links carry messages forever. Consider now $e2$, an execution identical to $e1$ up to t, but where a leaf process p (assumed correct in $e1$) crashes at time t (a leaf process has no successor processes). $e1$ and $e2$ are indistinguishable, to all processes above p in the tree (in this case all processes since p is a leaf node). Hence the process sending messages to p will not stop sending messages to p in $e2$, although the number of correct processes in $e2$ is one less than in $e1$, resulting in $\mathcal{C}$ links being used forever. However, if the process p (which crashes in $e2$) is not a leaf node, then p can be suspected by processes lower in the tree (or following it in a linear list) and initiate a procedure to eliminate communication with p and restore the fact that $\mathcal{C}-1$ links are used, concluding our proof.

■

We present next an optimal $\diamond\mathcal{S}^*$ implementation in $\mathcal{N}$.

B. An Optimal $\diamond\mathcal{S}^*$ Implementation

We now present an optimal algorithm (Algorithm 1) implementing $\diamond\mathcal{S}^*$.

We assume that processes are arranged in a logical linear list, where p_1 is at the head and p_n is at the tail. An intermediate process p_i is preceded by process p_{i-1} and followed by process p_{i+1}. When links in the system are timely for some finite interval, the number of links carrying messages infinitely often converges to $\mathcal{C}$ if at least one process crashes (possibly to $\mathcal{C}-1$ if process p_n, at the tail of the logical linear list, does not crash) and to $\mathcal{C}-1$ when no crashes occur. Recall that a timely link ensures that the delay of a reliable/successful message respects some bound; in this case we assume it to be the specified *time-out*.

The basic idea underlying Algorithm 1 is that a process at location x in the list always suspects all processes succeeding it, i.e., processes at locations $[x+1, ..., n]$. The goal of Algorithm 1 is to achieve two things:

1) Every correct process permanently suspects all crashed processes preceding it after T_D time slots of the crash.
2) When links are timely, no correct process suspects the first correct process in the logical linear list.

[7]In a synchronous system processing and message delays are bounded. This means that messages can be lost as long as they can be retransmitted successfully ensuring that total transmission time satisfies the delay bound.

Every process p_i maintains a set of suspected processes $L(p_i)$ and two variables $pred(p_i)$ and $succ(p_i)$ to respectively refer to the current predecessor process which is monitored by p_i and the current successor process to which p_i periodically (e.g., every t) sends heartbeat messages $< \mathsf{heartbeat}, \mathsf{L(p_i)}, \mathsf{p_i} >$. Note that process p_i at the head of the list has $pred(p_i) = \mathsf{null}$ whereas process p_j at the tail of the list has $succ(p_i) = \mathsf{null}$. We assume that processes have unique identifiers (names) and that they know their position in the list. Process p_i at all times suspects all processes down the list including the tail, i.e., $p_j \in L(p_i) \; \forall \, p_j : \{i < j \leq n\}$.

A process p_i suspects $pred(p_i)$ which it is monitoring, when a *time-out* expires (possibly some multiple of the sending period). In case of suspicion, p_i sends through a reliable link abstraction (as discussed in Section III), a message $< \mathsf{suspicion}, \mathsf{p_i} >$ to $pred(p_i)$ and sets its $pred(p_i)$ to the process before $pred(p_i)$ in the list (regardless if that process is in $L(p_i)$ or not) and updates its set of suspected processes $L(p_i)$ accordingly. Upon its receipt of a $< \mathsf{suspicion}, \mathsf{p_j} >$ message, a process p_i which is alive updates its successor to $succ(p_i) = p_j$ and will start sending $< \mathsf{heartbeat}, \mathsf{L(p_i)}, \mathsf{p_i} >$ to p_j. In addition, p_i also sends a message $< \mathsf{Alive?}, \mathsf{p_i} >$ to all the processes $p_k : \{i < k < j\}$, as p_i knows that p_j suspected all these process (p_k).

When a process p_i receives a message $< \mathsf{Alive?}, \mathsf{p_j} >$, p_i replies by sending to p_j, through a reliable link abstraction, $< \mathsf{heartbeat}, \mathsf{L(p_i)}, \mathsf{p_i} >$.

When a process p_i receives $< \mathsf{heartbeat}, \mathsf{L(p_j)}, \mathsf{p_j} >$, p_i checks if p_j precedes or succeeds it in the list. If p_j precedes p_i in the logical linear list (i.e., $j < i$) and succeeds (or is) the current predecessor of p_i, then $pred(p_i)$ and the set of suspected processes $L(p_i)$ are updated accordingly. Similarly, if p_j succeeds p_i in the logical linear list and precedes the current successor of p_i, then $succ(p_i)$ is updated.

Proof of Correctness of Algorithm 1

We first prove that Algorithm 1 implements $\diamond\mathcal{S}^*$, then we prove it is optimal. From the description of the algorithm, strong completeness is guaranteed if a crashed process p_i is suspected by all correct processes that follow it in the list within T_D timeslots, i.e., by all $p_j : \{j > i\}$.

Lemma 5. *The first correct process p_j, succeeding a crashed process p_i, eventually suspects p_i permanently.*

Proof: Let us denote by t the time at which p_i crashes. By lines (11-14) of Algorithm 1 guarantees that p_j will eventually set p_i as its predecessor and will monitor it. If p_j suspects p_i before time t then if p_j hears no messages from p_i it will suspect it forever. However if p_j hears a message from p_i, then by lines (26-33) of Algorithm 1 p_j will monitor p_i again and will eventually suspect p_i by (11-14) some time after t. Since after time t, p_i will no longer send any messages, then p_i will be suspected forever by p_j. ■

Algorithm 1 An Optimal $\diamond\mathcal{S}^*$ Algorithm.

```
1:  Initialize:
2:  set pred(p_i) = p_{i-1}        //set to null if i=1
3:  set succ(p_i) = p_{i+1}        //set to null if i=n
4:  set L(p_i) = {p_{i+1}, ..., p_n}
5:
6:  Repeat periodically:
7:  if succ(p_i) ≠ null then
8:      send < heartbeat, L(p_i), p_i > to succ(p_i)
9:  end if
10:
11: upon event Timeout on pred(p_i) do        //pred(p_i) not null
12:     L(p_i) = L(p_i) ∪ {pred(p_i)}
13:     send < suspicion, p_i > to pred(p_i)
14:     set pred(p_i) = process directly above pred(p_i) in the list.
15:
16: upon event receive < suspicion, p_j > do
17:     send < heartbeat, L(p_i), p_i > to p_j
18:     for i < k < j do
19:         send < Alive?, p_i > to p_k
20:     end for
21:     set succ(p_i) = p_j
22:
23: upon event receive < Alive?, p_j > do
24:     send < heartbeat, L(p_i), p_i > to p_j
25:
26: upon event receive < heartbeat, L(p_j), p_j > do
27:     if j < i ∧ index(pred(p_i)) ≤ j then        // index(p_i) = i
28:         set pred(p_i) = p_j
29:         update_list(L(p_j))
30:     end if
31:     if j > i ∧ index(succ(p_i)) > j then
32:         set succ(p_i) = p_j
33:     end if
34:
35: Function update_list(L(p_j)):
36: L(p_i) = L(p_j)
37: remove p_i from L(p_i)
```

Lemma 6. *The successor of a correct process p_i will eventually be (when links behave timely) the first correct process following p_i.*

Proof: Let us denote by p_j the first correct process that follows p_i. By lines (11-14) p_j will stop monitoring p_i and will monitor other processes only if p_j suspects p_i. However, p_j sends a suspicion messages through reliable link abstraction. Since both p_i and p_j are correct the suspicion will eventually reach p_i which by lines (16-21) will send p_j a heartbeat message through a reliable link abstraction and will set p_j as its successor. Again by the fact that the two processes are correct p_j will receive this heartbeat message and by lines (26-33) will monitor p_i again. Thus if links are timely p_j will not "time-out" on p_i. ■

By Lemma 6 and lines (26-33), the suspected list of a correct process p_i is propagated to all correct processes following p_i in the logical linear arrangement. By lines (35-37) all process following a crashed process will eventually suspect that crashed process permanently ensuring strong completeness. Now we prove the strong bounded completeness property of Algorithm 1, i.e., a failed process is permanently suspected by all correct processes after some

bounded duration of having failed. The longest delay of suspecting a crashed process would be when the tail of the logical list has to detect the crash of the head of the list. It is important to note the following: if process p_i is monitoring process p_j, then p_i can detect the failure of p_j after "*timeout*" time slots of not hearing from p_j. For presentation simplicity, we consider a network of three process p_1 being the head and p_3 being the tail. We accordingly show that p_3 detects the failure of p_1 within a bounded duration which we compute. By induction and transitivity this could be extended to a general network of n processes.

Assume that p_1 fails at time t. Recall also that every process sends a heartbeat message at each time slot to its successor. In that case, p_2, the process monitoring p_1, permanently suspects p_1 after "timeout" timeslots of not hearing from p_1. Given that a successful message transmission (not lost) between a pair of processes takes one timeslot. This means that p_2 suspects p_1 in the interval $[t+timeout+1, \infty]$ and thus within a bounded delay of "$timeout + 1$". p_3 can detect the crash of p_1 in two cases: (i) via p_2 (by seeing that p_1 is in the suspected list of p_2) or (ii) directly from p_1. The time taken for p_3 to detect p_1 in case (i) would be $timeout + 1 + T_h$, where $T_h < timeout + 1$. While in case (ii) p_3 has to suspect p_2 first, after which it monitors p_1 and suspects it. In that case p_3 would suspect p_1 in $2(timeout+1)$ time slots of not hearing from p_2. The worst-case delay for p_3 to permanently suspect p_1 would thus that p_3 keeps hearing from p_2 until time instant "$t+timeout+1$" and then does not hear from p_2 for a duration longer than $2(timeout + 1)$. This results in p_3 permanently suspecting p_1 in the interval $[t+3(timeout+1), \infty]$. As a consequence, a failed process would be permanently suspected by all correct processes within a maximum of $3(timeout + 1)$ timeslots after having failed. This proves the strong bounded completeness of Algorithm 1, given three processes.

Lemma 7. *When links are timely, all correct processes will trust the correct process at the head of the logical linear arrangement list.*

Proof: By Lemma 5, when links are timely every correct process is monitored by the first correct process following it in the list. Since the first correct process (at the head of list) does not get suspected by the processes monitoring it (as a consequence of links being timely), by Lemma 6 all correct process will eventually adapt its list of suspected processes which it is not included in by lines (35-37). ∎

The probabilistic accuracy can be insured by Lemma 7. At any point in time Lemma 7 has a positive probability of happening. Since we consider infinite time instants, then with probability 1 the following happens: there are infinitely many time instants t_G such that after each instant links in the network behave timely for the interval $[t_G, t_G + \Delta t]$, where Δt is any finite duration.

Now we show that Algorithm 1 is optimal. Let T be the time after which all faulty processes have crashed. Then after T and by Lemma 6, whenever the links become timely for any finite time the set of links sending heartbeat messages will be either $\mathcal{C} - 1$ if process p_n (at the tail of the list) is correct or $\mathcal{C}$ if p_n is faulty.

VI. Decisive Problems

In this section, we discuss what happens to deterministic algorithms using $\diamond\mathcal{S}$ to solve *decisive problems*, e.g., consensus (we give examples of decisive problems beyond consensus in Section VI-B), when put in $\mathcal{N}$ which provides $\diamond\mathcal{S}^*$ guarantees instead.

Definition 3. *A decisive problem is a problem which can be solved when a single irrevocable global decision is reached. Any decisive problem P requires that both of the following two properties are satisfied: (i) Termination: there is a point in time after which every correct process will have decided and (ii) Integrity: No process can decide more than once.*

Clearly consensus is one such problem, as the consensus abstraction guarantees: (i) *Validity:* A value decided is a value proposed, (ii) *Integrity:* No process decides more than once, (iii) *Agreement:* No two processes decide differently and (iv) *Termination:* there is a point in time after which all correct processes would have decided. For illustration, we first focus on consensus, then we discuss decisive problems.

A. Consensus with $\diamond\mathcal{S}^*$

We first show, for an exemplary existing consensus algorithm, that $\diamond\mathcal{S}^*$ can replace $\diamond\mathcal{S}$: the result would be solving "consensus with probability 1", in system $\mathcal{N}$. Then we present a general form of this result.

A Rotating Coordinator Algorithm. The basic idea behind the seminal rotating coordinator algorithm of [3] is that processes alternate in a role of "leader" until one of them succeeds in imposing a decision. The algorithm assumes a correct majority and uses two abstractions: (i) reliable links and (ii) reliable broadcast. Both reliable links and reliable broadcast can be implemented in our system $\mathcal{N}$ (in the sense specified in Section III).

The algorithm is round-based, i.e., the processes move incrementally from one round to the other. Process p_i is the "leader" of every round $k : k \mod n = i$. In such a round, process p_i does the following: (i) p_i selects among a majority the latest adopted value (latest w.r.t. round), (ii) p_i sends that value to all processes and waits for the acknowledgement of the majority and (iii) once p_i succeeds in imposing that value on a majority, p_i uses reliable broadcast to send its decision to all and decides. It is important to note that p_i succeeds if it is not suspected by the majority (processes that suspect p_i inform p_i and move to the next round, including p_i).

Theorem 6. *The algorithm of [3] implements "consensus with probability 1" in $\mathcal{N}$ using $\diamond\mathcal{S}^*$ (instead of $\diamond\mathcal{S}$).*

Proof: It is easy to see that $\diamond\mathcal{S}^*$ guarantees the strong completeness of $\diamond\mathcal{S}$ and that the reliable links and reliable broadcast in system $\mathcal{N}$ (see Section III) guarantee respectively the properties of the reliable links and reliable broadcast depicted in [3]. Thus the proof of correctness provided in [3] remains true, except for the parts relying on the accuracy of $\diamond\mathcal{S}$, namely termination. Thus it is sufficient to prove that the accuracy of $\diamond\mathcal{S}^*$ guarantees that all processes decide.

Consider t_{rand} to be any point in time after all faulty processes have crashed. Since the algorithm of [3] operates in asynchronous rounds, then at time t_{rand} processes might be at different rounds. We denote by r the largest round among all processes at time t_{rand} and by $\Delta r_{t_{rand}}$ the maximum difference between the rounds of the processes at time t_{rand}. Note that from [3], $\Delta r_{t_{rand}} \leq n$, n being the total number of processes in the system.

In the algorithm of [3], after time t_{rand}, a process, be it a leader or not, completes a round when a bounded number of messages (unicast ot broadcast messages) is sent/received or when it suspects the leader of that round. Let M be the maximum number of messages for a process to complete a round. Let $T_M \geq T_D$ be the amount time for exchanging the M messages, such that the probability of exchanging M messages in T_M time slots is positive (the properties of reliable links and reliable broadcast primitive defined in Section III guarantee the existence of such a T_M). Recall that the accuracy of $\diamond\mathcal{S}^*$ guarantees that with probability 1 the following holds: for any finite duration Δt, there exists infinitely many time instants $t_G \in \{1, ..., \infty\}$ such that some correct process, say q, is not suspected by all correct processes for the interval $[t_G, t_G + \Delta t]$.

Consider now $r' \geq r$ to be the round in which q becomes leader. Thus with positive probability, all processes can reach round r' after $T_M \cdot (\Delta r_{t_{rand}} + (r' - r))$ timeslots from t_{rand}. If q is not suspected by any of the processes, then with positive probability every process decides after T_M time slots from reaching round r'. In other words, if process q is not suspected by any correct process in the interval $[t_{rand}, t_{rand} + T_M \cdot (\Delta r_{t_{rand}} + (r' - r) + 1)]$, then there is a positive probability that all processes decide.

Since t_{rand} is any point in time, after all processes have crashed, then we can assume $t_{rand} = t_G$, such that $\Delta t = T_M \cdot (\Delta r_{t_{rand}} + (r' - r) + 1)$. By the accuracy of $\diamond\mathcal{S}^*$, with probability 1: there exists infinitely many t_G time instants (after all faulty processes have crashed) such that process q is not suspected by all correct processes for the interval $[t_G, T_G + T_M \cdot (\Delta r_{t_{rand}} + (r' - r) + 1)]$. Since there is a positive probability of all processes deciding $T_M \cdot (\Delta r_{t_{rand}} + (r' - r) + 1)$ time slots after t_G and there are infinitely many t_G time instants, then with probability 1 we have the following: there is a point in time after which all correct processes would have decided. ■

Definition 4. *An asynchronous algorithm $\mathcal{A}$ that solves a decisive problem P is said to be $\diamond\mathcal{S}^{\mathcal{N}}$-bounded if $\mathcal{A}$ satisfies the following properties:*

1) *$\mathcal{A}$ uses as external blocks only the failure detector $\diamond\mathcal{S}$ and communication primitives implementable in $\mathcal{N}$, such as reliable links and reliable broadcast (see specification in Section III).*
2) *Consider that there exists a point in time, t_G, after which some correct process is never suspected by all correct processes. Then $\mathcal{A}$ needs a bounded number of messages to be sent after t_G and until P is solved (i.e., all correct processes decide).*

In fact many of the consensus algorithms using $\diamond\mathcal{S}$ in the literature are $\diamond\mathcal{S}^{\mathcal{N}}$-bounded. This makes our results applicable to wide range of existing algorithms.

Theorem 7. *Any asynchronous algorithm that uses $\diamond\mathcal{S}$ to solve consensus and is $\diamond\mathcal{S}^{\mathcal{N}}$-bounded, solves "consensus with probability 1" in $\mathcal{N}$ when using $\diamond\mathcal{S}^*$ instead.*

Proof can be seen for the more general result of Theorem 8.

B. Decisive Problems with $\diamond\mathcal{S}^*$

Now we generalize the result of Theorem 7 for *decisive problems* in general.

Theorem 8. *Any asynchronous algorithm $\mathcal{A}$ that uses $\diamond\mathcal{S}$ to solve a decisive problem P and is $\diamond\mathcal{S}$-bounded, solves "P with probability 1" in $\mathcal{N}$, when $\diamond\mathcal{S}^*$ is used instead* [8].

Proof: $\diamond\mathcal{S}^*$ guarantees the strong completeness of $\diamond\mathcal{S}$. Thus w.r.t. $\mathcal{A}$, the difference between $\diamond\mathcal{S}^*$ and $\diamond\mathcal{S}$ is in the provided accuracy property. The accuracy of $\diamond\mathcal{S}$ is a property which holds at some unknown point in time. As a result, any algorithm $\mathcal{A}$ that solves a decisive problem P using $\diamond\mathcal{S}$, guarantees all safety properties required by P regardless of the accuracy of $\diamond\mathcal{S}^*$. $\mathcal{A}$ hence uses the accuracy of $\diamond\mathcal{S}$ to guarantee liveness, in particular termination, i.e., there is a time after which all correct processes decide. It thus suffices to prove that with respect to $\mathcal{A}$ and with probability 1 the following is satisfied: The accuracy of $\diamond\mathcal{S}^*$ guarantees that there is a point in time after which all processes decide.

Assume the existence of an external clock (not accessible but merely used as a reference to clarify the proof construction). Let t_{start} denote the time instant at which $\mathcal{A}$ starts executing. Using $\diamond\mathcal{S}$ in $\mathcal{A}$ to solve P implies that after t_{start} there is a time when all processes decide and P is solved (see Definition 3). Precisely, after some correct process is never suspected by all correct processes, all correct processes executing $\mathcal{A}$ should exchange a finite bounded number of messages after which P would be solved. Let M denote the upper bound on the number of messages (be them unicasts or broadcasts) needed by $\mathcal{A}$ from the time some correct

[8] Solving "P with probability 1" means guaranteeing all safety properties of P deterministically and ensuring termination with probability 1.

process is never suspected by all correct processes until P is solved. All events that could occur have a bounded delay (process speeds, crash detection, etc.), except for reliable message transmissions (be them uni-casts or broadcasts). Using communication primitives as the reliable links and reliable broadcast in $\mathcal{N}$, the delay for delivering a single message may be arbitrarily long. However it is possible, with positive probability, that a message gets delivered after a known fixed delay, e.g., in x time slots after being sent (see reliable transmission Section III). Thus and without loss of generality, at any point in time where some correct process in never suspected by all correct processes, it is possible (with positive probability) for the M messages to be exchanged within a known fixed duration, say T_M, after which all processes would have decided.

From Definition 1, the accuracy of $\diamond\mathcal{S}^*$ guarantees with probability 1 that: there exists infinitely many time instants t_G after which a unique correct process is not suspected by any correct process for the interval $t_G + T_M$. Since there are infinitely many such t_G time instants and at each t_G there is a positive probability for the M messages to be exchanged within T_M, then with probability 1 the following happens: there is a time after which all processes would have decided within T_M timeslots and thus P would be solved. ■

Other Decisive Problems. Besides consensus, there can be many other decisive problems, e.g., non-blocking atomic commit (NBAC), $k-$set agreement, fast consensus. Some of these decisive problems, such as NBAC, are solved using $\diamond\mathcal{P}$. In Appendix C we show that it is possible to formulate $\diamond\mathcal{P}^*$, a variant of $\diamond\mathcal{P}$ (in the main part of the paper we concentrate on $\diamond\mathcal{S}$). We also show in Appendix C that Theorem 8 can be extended to the set of decisive problems solvable with $\diamond P$ when replaced by $\diamond P^*$, thus covering a wider set of problems, besides consensus.

VII. Concluding Remarks

We investigated failure detection in systems embodying asynchrony via probabilistic synchronous communication. In contrast to the conventional distributed computing assumptions when building failure detectors, which hinged on link synchrony guarantees that need to hold deterministically forever, we adopted a more realistic link behaviour motivated by networking views on actual packet loss. We show that "$\diamond\mathcal{S}$ with probability 1" cannot be implemented given such link behaviour ($\diamond\mathcal{S}$ being established as the weakest failure detector to implement consensus), despite the fact that "consensus with probability 1" can be implemented without requiring any randomness in the algorithm itself. We accordingly refine the notion of failure detectors defining $\diamond\mathcal{S}^*$ which does not require any "forever" guarantee from the underlying network. We show that $\diamond\mathcal{S}^*$ can be implemented in system $\mathcal{N}$ and even efficiently. In addition, we show that $\diamond\mathcal{S}^*$ can replace $\diamond\mathcal{S}$ in several deterministic consensus algorithm and yields an algorithm that solves "consensus with probability 1". We also generalise this result to encompass a more general set of problems and failure detectors.

Potential future work may investigate the weakest probabilistic system to implement $\diamond\mathcal{S}^*$ or the solvability of problems, besides the decisive set, using our new notion of failure detectors.

References

[1] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, 1998.

[2] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, 1985.

[3] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, 1996.

[4] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Communication-efficient leader election and consensus with limited link synchrony," in *PODC*, 2004.

[5] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "On implementing omega in systems with weak reliability and synchrony assumptions," *LNCS*, vol. 21, 2008.

[6] A. Mostefaoui, M. Raynal, and C. Travers, "Time-free and timer-based assumptions can be combined to obtain eventual leadership," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, 2006.

[7] A. Mostefaoui, E. Mourgaya, and M. Raynal, "Asynchronous implementation of failure detectors," in *DSN*, 2003.

[8] D. Malkhi, F. Oprea, and L. Zhou, "Omega meets paxos: Leader election and stability without eventual timely links," in *DISC*, 2005.

[9] E. Jiménez, S. Arévalo, and A. Fernández, "Implementing unreliable failure detectors with unknown membership," *Inf. Process. Lett.*, vol. 100, 2006.

[10] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," in *PODC*, 1992.

[11] B. Charron-Bost, M. Hutle, and J. Widder, "In search of lost time," *Inf. Process. Lett.*, vol. 110, no. 21, 2010.

[12] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, 1988.

[13] J.-P. Vasseur and A. Dunkels, *Interconnecting smart objects with ip*. Morgan Kaufmann, 2010.

[14] G. Hasslinger and O. Hohlfeld, "The gilbert-elliott model for packet loss in real time services on the internet," in *MMB*, 2008.

[15] Q. Zhang and S. Kassam, "Finite-state markov model for rayleigh fading channels," *IEEE Trans. Commun.*, 1999.

[16] L. Kanal and A. Sastry, "Models for channels with memory and their applications to error control," *Proceedings of the IEEE.*, vol. 66, 1978.

[17] D. Dzung, R. Guerraoui, D. Kozhaya, and Y.-A. Pignolet, "Source routing in time-varying lossy networks," in *NETYS*, 2015.

[18] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *J. ACM*, vol. 32, no. 4, 1985.

[19] M. Kurt, S. Krishnamoorthy, K. Agrawal, and G. Agrawal, "Fault-tolerant dynamic task graph scheduling," in *SC*, Nov 2014.

[20] V. Berten, J. Goossens, and E. Jeannot, "A probabilistic approach for fault tolerant multiprocessor real-time scheduling," in *IPDPS*, 2006.

[21] Y. Jia, G. Bosilca, P. Luszczek, and J. J. Dongarra, "Parallel reduction to hessenberg form with algorithm-based fault tolerance," in *SC*, 2013.

[22] S. Di, C.-L. Wang, and F. Cappello, "Adaptive algorithm for minimizing cloud task length with prediction errors," *IEEE Trans. Cloud Comput*, vol. 2, 2014.

[23] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing algorithms and fault prediction," *J. Parallel Distrib. Comput.*, vol. 74, 2014.

[24] A. Benoit, A. Cavelan, Y. Robert, and H. Sun, "Assessing general-purpose algorithms to cope with fail-stop and silent errors," in *LNCS*, 2015.

[25] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. Debardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *Int. J. High Perform. Comput. Appl.*, vol. 28, no. 2, 2014.

[26] W. Dweik, M. Abdel-Majeed, and M. Annavaram, "Warped-shield: Tolerating hard faults in gpgpus," in *DSN*, 2014.

[27] J. Beauquier, S. Delat, S. Dolev, and S. Tixeuil, "Transient fault detectors," in *Distributed Computing*, ser. LNCS, 1998, vol. 1499.

[28] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," *J. ACM*, vol. 43, 1996.

[29] M. Larrea, A. Lafuente, I. Soraluze, R. Cortias, and J. Wieland, "On the implementation of communication-optimal failure detectors," in *LNCS*, 2007, vol. 4746.

[30] A. Lafuente, M. Larrea, I. Soraluze, and R. Cortias, "Communication-optimal eventually perfect failure detection in partially synchronous systems," *J. Comput. System Sci.*, vol. 81, 2015.

[31] M. Larrea, S. Arevalo, and A. Fernndez, "Efficient algorithms to implement unreliable failure detectors in partially synchronous systems," *LNCS*, vol. vol.1693, 1999.

[32] M. Larrea, A. F. Anta, and S. Arévalo, "Implementing the weakest failure detector for solving the consensus problem," *IJPEDS*, vol. 28, 2013.

[33] M. Larrea, A. Fernandez, and S. Arevalo, "Optimal implementation of the weakest failure detector for solving consensus," in *SRDS*, 2000.

[34] N. Santoro and P. Widmayer, "Time is not a healer," in *STACS*, 1989.

[35] U. Schmid, B. Weiss, and I. Keidar, "Impossibility results and lower bounds for consensus under link failures," *SIAM J. Comput.*, vol. 38, no. 5, 2009.

[36] I. Soraluze, R. Cortiñas, A. Lafuente, M. Larrea, and F. Freiling, "Communication-efficient failure detection and consensus in omission environments," *Inf. Process. Lett.*, no. 6, 2011.

[37] J. Aspnes, H. Attiya, and K. Censor, "Combining shared-coin algorithms," *J. Parallel Distrib. Comput.*, vol. 70, no. 3, 2010.

[38] D. Alistarh, J. Aspnes, V. King, and J. Saia, "Communication-efficient randomized consensus," in *LNCS*, 2014, vol. 8784.

[39] P. Fraigniaud, M. Gs, A. Korman, M. Parter, and D. Peleg, "Randomized distributed decision," *LNCS*, vol. 27, 2014.

[40] H. Moniz, N. Neves, M. Correia, and P. Verssimo, "Randomization can be a healer: Consensus with dynamic omission failures," in *LNCS*, 2009, vol. 5805.

[41] H. Moniz, N. Neves, and M. Correia, "Turquois: Byzantine consensus in wireless ad hoc networks," in *DSN*, 2010.

[42] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," *IEEE Trans. Comput.*, vol. 51, no. 5, 2002.

[43] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *DSN*, 2002.

[44] I. Gupta, T. D. Chandra, and G. S. Goldszmidt, "On scalable and efficient distributed failure detectors," in *PODC*, 2001.

[45] N. Hayashibara, X. Defago, and T. Katayama, "Two-ways adaptive failure detection with the phi-failure detector," in *ICAC*, 2003.

[46] M. Biely, M. Hutle, L. Penso, and J. Widder, "Relating stabilizing timing assumptions to stabilizing failure detectors regarding solvability and efficiency," in *SSS*, 2007, vol. 4838.

[47] A. Conrejo, N. Lynch, and S. Sastry, "Asynchronous failure detectors," in *PODC*, 2012.

[48] R. Guerraoui, R. Olivera, and A. Schiper, "Stubborn communication channels," Tech. Rep., 1996.

[49] M. Kawazoe Aguilera, W. Chen, and S. Toueg, "Heartbeat: A timeout-free failure detector for quiescent reliable communication," *LNCS*, vol. 1320, 1997.

[50] D. Dzung, R. Guerraoui, D. Kozhaya, and Y.-A. Pignolet, "To transmit now or not to transmit now," in *SRDS*, 2015.

[51] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*. Springer, 2011.

APPENDIX

A. Other Probabilistic Failure Detectors

Since $\diamond\mathcal{S}^*$ can be implemented in our system, we study the possibility of implementing meaningful probabilistic variants of failure detectors which noticeably simplify the design of distributed algorithms, namely perfect failure detection. A perfect failure detector module, $\mathcal{P}$, guarantees in addition to strong completeness, the strong accuracy property, which says that no process is suspected before it crashes. Distributed algorithms, specifically those solving consensus, using $\mathcal{P}$ are easy to design due to their implicit reliance on the strong accuracy property of $\mathcal{P}$ to guarantee some safety property [51]. The liveness of such algorithms typically relies on the strong completeness. It is important to note that on the contrary consensus algorithms based on unreliable failure detectors [3] usually rely on the eventual accuracy to guarantee liveness of the algorithm. We thus define $\mathcal{P}^*$, a probabilistic variant of $\mathcal{P}$, as a failure detector that guarantees strong accuracy and probabilistic strong completeness, where the latter can be formally defined as: *Probabilistic Strong Completeness:* Eventually every process that crashes can be suspected, with positive probability, by every correct process.

Theorem 9. *It is impossible to implement the failure detector $\mathcal{P}^*$ in $\mathcal{N}$, even if at most one process can fail.*

Proof: Consider a network of $n = 2$ processes, p_1 and p_2, and the following executions:

$e1$. an execution where process p_2 fails at time t.
$e2$. an execution where processes p_1 and p_2 are both correct but get partitioned at time t.

By the probabilistic strong completeness there is a time after which p_1 in execution $e1$ has a positive probability of suspecting p_2. By Lemma 3, executions $e1$ and $e2$ can be indistinguishable to p_1 for any finite duration after t. Accordingly there is a time where p_1 has a positive probability of suspecting p_2 in execution $e2$. By the strong accuracy property of $\mathcal{P}^*$ a correct process is never suspected. Thus the probability of p_1 suspecting p_2 in $e2$ should be 0 at all times, a contradiction concluding the proof . ■

We show now that $\diamond\mathcal{P}^*$, a variant of $\diamond\mathcal{P}$ can be implemented. Failure detector $\diamond\mathcal{P}^*$ guarantees: (i) *strong bounded completeness* and (ii) *probabilistic eventual strong accuracy:* Consider any finite duration Δt. With probability 1 the following occurs: there exists infinitely many time instants t_G, such that after each all correct processes are not suspected by any correct process for the interval $t_G + \Delta t$.

Theorem 10. *It is possible to implement $\diamond\mathcal{P}^*$, in $\mathcal{N}$, even if $n - 1$ processes can crash.*

Proof: The proof is similar to that of Theorem 2. Following from Lemma 4, if all correct processes broadcast messages forever (send an infinite number of messages), then with probability 1, the following will be observed: any finite number of consecutive messages, e.g., Δt messages, is successfully transmitted, i.e., with no losses.

As such, an algorithm which satisfies both characteristics below for example implements $\diamond\mathcal{P}^*$:

1) All processes periodically (say with period Δt_1 chosen arbitrarily) broadcast messages forever.
2) Process p_i suspects another process p_j only if p_i receives no message form p_j for a period strictly greater than Δt_1.

■

B. Substituting Global Clock by Unsynchronized Local Clocks

We briefly discuss in this section how the global clock assumption can be substituted with local clocks which do not need to be synchronized. For this purpose, we redefine the system model accordingly to accommodate new notations.

We consider a distributed system $\mathcal{N}$ consisting of a finite set Π of $n > 1$ processes, $\Pi = \{p_1, p_2, ..., p_n\}$, which communicate by message passing. We assume that all processes have access to local clocks with discrete time events denoted by $t_{p_i} : \{1, 2, 3, ...\}$. A process p_i is assumed to take actions, i.e., either send or receive or both, at the discrete time events of t_{p_i}. The time interval between consecutive events in t_{p_i}, $\forall\ i$ (i.e., for all processes) is assumed to be the same such that it is an upper bound on the propagation delay (t_{pg}) over any link interconnecting any two processes. Processing delays are assumed to be negligible compared to communication delays.

Communication Links. The links interconnecting processes are assumed to be uni-directional uni-cast links. In particular, every pair of processes (p_i, p_j) is connected by two uni-directional links: l_{ij} and l_{ji}. These links exhibit changes in their transmission quality, as the quality of the underlying channels might depend on various propagation conditions. We thus assume that a link l_{ij} has a probability $0 < P_{ij}(t_{p_i}) < 1$ of losing messages at time t_{p_i}. This captures the very idea that a link is not always reliable and can lose messages for an unbounded but finite period. The value of $P_{ij}(t_{p_i})$ can change with time; specifically, at each time $t_{p_i} : \{1, 2, 3, ...\}$, $P_{ij}(t_{p_i})$ may have any value in $(0, 1)$. However, we assume that the value of $P_{ij}(t_{p_i})$ remains constant between consecutive intervals of t_{p_i}. We refer to such links as *probabilistic* links. A probabilistic link thus constitutes an instance of the *fair-loss* link [48], where a message sent by some process p_i infinitely often is received infinitely often.

Faulty Processes. Processes faults are defined as in Section III.

Monitoring Schemes. Given local clocks that are not synchronized (i.e., might be skew), we suggest now, through an example, a small modification to the period at which processes send messages and that at which processes suspect each other, such that $\diamond\mathcal{S}^*$ can be implemented.

Precisely, the algorithm in Theorem 2 can be adapted as follows. First, a process p_i sends messages periodically, by sending messages at every time instant of its local clock t_{p_i}. Initially all processes trust (do not suspect) all other processes. At every odd time event of its local clock such that $t_{p_i} > 1$, process p_i suspects a process p_j if it does not receive a new message since the last odd time event. At every time t_{p_i} if p_i receives a new message from p_j, then p_i trusts p_j. If all messages sent by p_i are not lost for some finite period $t_{p_i} + \Delta t$, then all other processes p_j will not suspect p_i for some period $t_{pj} + \Delta t$. The skew (w.r.t. some global time) between the periods in which p_i is trusted by p_j is $\leq 2t_{pg}\ \forall\ j$ (recall that t_{pg} is the maximum bound on the prorogation delay). In other words, this means that, if $\Delta t > 2t_{pg}$, then is a common duration between all processes during which p_i is not suspected. This duration is at least $\Delta t - 2t_{pg}$.

A similar modification can be applied to Algorithm 1 as well, to have a valid implementation of $\diamond\mathcal{S}^*$.

C. $\diamond\mathcal{P}^*$ Algorithms for Decisive Problems

In this section we extend Theorem 8 for the set of algorithms that solve decisive problems using $\diamond P$. First we recall the definitions of $\diamond P$ and $\diamond P^*$.

Failure detector $\diamond\mathcal{P}$ guarantees: (i) *strong completeness* and (ii) *eventual strong accuracy:* There exists a time after which all correct processes are never suspected by any correct process.

Failure detector $\diamond\mathcal{P}^*$ guarantees: (i) *strong bounded completeness* and (ii) *probabilistic eventual strong accuracy:* Consider any finite duration Δt. With positive probability, a all correct processes are not suspected by any correct process for the interval $t_s + \Delta t$, $\forall t_s \in \{1, 2, 3, ...\}$.

Definition 5. *An asynchronous algorithm $\mathcal{A}$ that solves a decisive problem P is said to be $\diamond\mathcal{P}^{\mathcal{N}}$-bounded if it satisfies the following properties:*

1) *$\mathcal{A}$ uses as external blocks only the failure detector $\diamond\mathcal{P}$ and communication primitives implementable in $\mathcal{N}$, such as reliable links and reliable broadcast (see specification in Section III).*
2) *Assume that there exists a point in time t_G when all correct process are never suspected by any correct process. Then $\mathcal{A}$ needs a bounded number of messages to be sent after t_G and until P is solved (i.e., all correct processes decide).*

Theorem 11. *Any algorithm $\mathcal{A}$ that uses $\diamond\mathcal{P}$ to solve a decisive problem P and is $\diamond\mathcal{P}^{\mathcal{N}}$-bounded, solves P in $\mathcal{N}$ guaranteeing termination (i.e., all processes decide) with probability 1, when $\diamond\mathcal{P}^*$ is used instead.*

Proof: We follow similar steps as those adopted in the proof of Theorem 8.

$\diamond\mathcal{P}^*$ provides (in a stronger form) strong completeness as $\diamond\mathcal{P}$. It thus suffices to prove that with respect to $\mathcal{A}$ and with probability 1 the following is satisfied: $\diamond\mathcal{P}^*$ provides the same accuracy as $\diamond\mathcal{P}$.

Assume the existence of an external clock. This clock is not accessible but merely used as a reference to clarify the proof construction. Let t_{start} denote the time instant at which $\mathcal{A}$ starts executing. Using $\diamond\mathcal{P}$ in $\mathcal{A}$ to solve P implies that after t_{start} there is a time when all processes decide and P is solved (see Definition 3). Precisely, after the time when all correct processes are never suspected by any correct process, all correct processes executing $\mathcal{A}$ should exchange a finite bounded number of messages after which P would be solved. Let M denote the upper bound on the number of messages (be them uni-casts or broadcasts) needed by $\mathcal{A}$ from the time all correct processes are never suspected by any correct process until P is solved. All events that could occur after t_G have a bounded delay (process speeds, crash detection, etc.), except for reliable message transmissions (be them uni-casts or broadcasts). Using communication primitives as the reliable links and reliable broadcast in $\mathcal{N}$, the delay for delivering a single message may be arbitrarily long. However it is possible, with positive probability, that a message gets delivered after a known fixed delay, e.g., x time slots of being sent, at any time instant at which it might be sent (see specifications of reliable transmission Section III). Thus and without loss of generality it is possible (with positive probability) for the M messages to be exchanged within a known fixed duration, say T_M, after which all processes would have decided and thus P would be solved.

Therefore, P can be solved with $\diamond\mathcal{P}^*$ if we prove that $\diamond\mathcal{P}^*$ can with probability 1 provide the following: there is some time instant $t_G \in \{1, 2, 3, ...\}$ after which all correct processes are not suspected by any correct process for the interval $[t_G, T_M]$. From Appendix A, the accuracy of $\diamond\mathcal{P}^*$ guarantees that: with positive probability, all correct processes are not suspected by any correct process for the interval $t_s + T_M$, $\forall t_s \in \{1, 2, 3, ...\}$. Since this holds for every $t_s \in \{1, 2, ..., \infty\}$, then $\diamond\mathcal{P}^*$ can with probability 1 provide that: there is some time instant $t_G \in \{1, 2, 3, ...\}$ after which all correct processes are not suspected by any correct process for the interval $[t_G, T_M]$.

■