Validity in Network-Agnostic Byzantine Agreement

Andrei Constantinescu
ETH Zürich, Switzerland
DFINITY Foundation, Zürich, Switzerland

Marc Dufay
ETH Zürich, Switzerland

Diana Ghinea[1]
Lucerne University of Applied Sciences and Arts, Switzerland

Roger Wattenhofer
ETH Zürich, Switzerland

Abstract

Byzantine Agreement (BA) considers a setting of n parties, out of which up to t can exhibit byzantine (malicious) behavior. Honest parties must decide on a common value (agreement), which must belong to a set determined by the honest inputs (validity). Depending on the use case, this set can grow or shrink, leading to various possible desiderata collectively known as validity conditions. Varying the validity property requirement can affect the regime under which BA is solvable.

Our work investigates how the selected validity property impacts BA solvability in the network-agnostic model, where the network can either be synchronous with up to t_s byzantine parties or asynchronous with up to $t_a \leq t_s$ byzantine parties. We give necessary and sufficient conditions for a validity property to render BA solvable, both for the case with cryptographic setup and for the one without. This traces the precise boundary of solvability in the network-agnostic model for every validity property. Our proof of sufficiency provides a universal protocol, that achieves BA for a given validity property whenever the provided conditions are satisfied.

We note that, for any non-trivial validity property, the condition $2 \cdot t_s + t_a < n$ is necessary for BA to be solvable, even with cryptographic setup. Specializing this claim to $t_a = 0$ gives that $t < n/2$ is required whenever one expects a purely synchronous protocol to also work in an asynchronous network when there are no corruptions. This is especially surprising given that, for some validity properties, $t < n$ is a sufficient condition without the last stipulation.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols

Keywords and phrases byzantine agreement, validity, network-agnostic protocols

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.24

Related Version *Full Version*: `https://arxiv.org/abs/2410.19721` [9]

1 Introduction

Achieving agreement among the parties involved in a distributed system is crucial for maintaining consistent views. This becomes particularly challenging due to the potential for parties' failures, which can range from benign crashes to malicious (byzantine) behavior. Byzantine Agreement (BA) is an extensively studied problem in distributed computing that tackles this challenge. It seeks to establish a common value amongst a set of n parties even when up to t parties exhibit byzantine behavior. A crucial aspect of BA lies in its validity condition, which requires that the value agreed upon reflects the honest parties' proposals rather than being a default or arbitrary value. Standard BA definitions consider the so-called *strong unanimity* (also known as *strong validity*): if all honest parties propose the same value v,

[1] This work was partially carried out while the author was at ETH Zürich.

the agreed-upon output must be v. This is a powerful guarantee in applications concerning binary decisions, as it coincides with *honest-input validity*, which requires agreement on an honest input. However, strong unanimity fails to provide meaningful outputs for larger input spaces. For instance, consider a set of parties running a BA protocol to agree on a room's temperature: minor measurement errors are inherent, and hence strong unanimity allows agreement on a corrupted party's proposal. Similar issues arise in scenarios such as deciding on a location using GPS coordinates [5] or when the map is modeled as a graph [11, 30].

While achieving honest-input validity in scenarios where the input space is large (size $\omega(n)$) is impossible [29], the literature offers a plethora of weaker alternatives that are stronger and more meaningful than strong unanimity. For instance, one may avoid corrupted outputs by enhancing strong unanimity with an additional condition called *intrusion tolerance* [13, 22, 23]: the honest parties either agree on an honest input or on a special symbol $\perp$. In the aforementioned scenarios of deciding on a measurement or a meeting point, a highly suitable alternative is the so-called *convex validity*: the output agreed upon must be in the honest parties' inputs convex hull, i.e., within the range of honest inputs if the input space is a subset of $\mathbb{R}$ [11, 22, 23, 28, 30, 32]. Stronger variants for real values require the outputs to be close to the honest inputs' median [10, 31], or to the k-th lowest honest input [27]. However, the honest-range approach is not a universal solution: this would carry no meaning in a voting problem if we represented candidates with integers. Instead, approaches based on social choice theory (*Pareto validity*) lead to more appropriate validity definitions [26].

The multitude of validity definitions leads to a natural question: what are the necessary and sufficient conditions for achieving BA with a given validity property, i.e., *to solve a given validity property*? This question can be concerned with multiple aspects, such as resilience thresholds t, round complexity, or message complexity. In addition, the conditions may depend on whether a cryptographic setup and randomization are available. The communication model assumed also plays an important role: one extreme is the synchronous model, where all parties have synchronized clocks, and all messages get delivered within a predefined amount of time Δ. This enables elegant protocols that operate in rounds, but that may fail in a real-life network where sporadic issues are possible. The other extreme is the asynchronous model, which only assumes that messages get delivered eventually. The asynchronous model comes with highly robust protocols, but also with important drawbacks: **(a)** lower resilience threshold: e.g., $t < n/3$ as opposed to $t < n/2$ (assuming digital signatures) for strong unanimity, or $t < n/4$ as opposed to $t < n/3$ for convex validity in $\mathbb{R}^2$; **(b)** randomization is a requirement [19]. The middle ground between the two models has also been considered. The partially synchronous model [17] bridges the gap between the two extremes by assuming that the network is initially asynchronous and eventually becomes synchronous. In the synchronous and partially synchronous models, the solvability of validity conditions using deterministic protocols is completely understood [7, 8].

In this work, we are concerned with a different paradigm for bridging the gap between synchrony and asynchrony, namely the *network-agnostic* model [4]: parties are initially unaware of whether the network is synchronous or not: if it is synchronous, up to t_s of the parties involved may be corrupted, and otherwise only up to $t_a \leq t_s$. Network-agnostic protocols are designed to provide guarantees in both cases. We ask the following question:

Under what conditions can a validity property be solved in the network-agnostic model?

1.1 Our Contribution

We provide a complete characterization for achieving network-agnostic BA with a given validity property, establishing tight necessary and sufficient conditions.

We first show that, even if cryptographic setup is available, the condition $n > 2 \cdot t_s + t_a$ is a requirement for any non-trivial validity condition to be solvable (i.e., a condition for which simply outputting a default value does not suffice). When no cryptographic setup is available, we show the stronger requirement of $n > 3 \cdot t_s$. Our proof for the latter, in fact, works in the synchronous model and, therefore, strengthens the characterization provided by [7] for the synchronous model. In particular, [7] only focuses on deterministic protocols, while our proofs rely on different techniques that also apply to randomized protocols.

Afterwards, regardless of whether cryptographic setup is available or not, we add one more necessary condition, which is an adaptation of the *similarity condition* of [8] and the *containment condition* of [7] to the network-agnostic model. Roughly, this requires that *similar* honest inputs' configurations have a valid output in common. The term *similar* captures that some of the proposed inputs may come from corrupted parties, and that, in asynchronous networks, some honest inputs may be missing due to high network delays.

Finally, we show that, together, the aforementioned conditions are also sufficient by providing a universal protocol, that achieves network-agnostic BA for a given validity property whenever these conditions are satisfied. This is a more general variant of the protocol of [11]. In particular, the requirement for solvability is precisely $n > 2 \cdot t_s + t_a$ together with the similarity condition assuming cryptographic setup, and $n > 3 \cdot t_s$ together with the similarity condition assuming no cryptographic setup. Our protocol is randomized, which is a requirement when the network may be asynchronous and $t_a > 0$ [19].

1.2 Related Work

General validity conditions. The foundational investigation into general validity properties was initiated by Civit et al. [8] for the partially synchronous model. Subsequently, Civit et al. [7] embarked on a follow-up study, extending their analysis to the synchronous model. Both works provide a complete characterization, identifying the necessary and sufficient conditions for solving a validity property deterministically. We also note that the contributions of [7, 8] extend beyond this characterization, an important side of these works lying in the exploration of lower bounds on message complexity. This generalizes the well-established Dolev-Reischuk bound on message complexity for BA with strong unanimity [15] to encompass the broader landscape of non-trivial validity properties. Considerations of message complexity are outside our scope. We also note that *probabilistic* validity properties are outside the scope of the analysis of Civit et al, and also outside our scope. A notable example is *qualitative validity*, introduced by Goren et al [25].

Network-Agnostic BA and particular validity conditions. Designing protocols that achieve security guarantees in both synchronous and asynchronous networks has been the subject of an extensive line of work. The network-agnostic paradigm was introduced by Blum, Katz and Loss [4]. The work of [4] shows that, if a public key infrastructure is provided, BA with strong unanimity can be achieved if and only if $n > 2 \cdot t_s + t_a$. Further works on network-agnostic BA with strong unanimity have focused on improving the efficiency guarantees [12, 13].

Due to its broad applicability, convex validity within the network-agnostic communication paradigm has attracted increased attention. Ghinea, Liu-Zhang and Wattenhofer [20, 21] have investigated the feasibility of achieving convex validity for a weaker variant of BA, known as Approximate Agreement [1, 14]. In particular, [20] shows that Approximate Agreement on real numbers is solvable under the same necessary and sufficient condition $n > 2 \cdot t_s + t_a$ assuming a public key infrastructure. Building on the previous, [21] gives sufficient conditions for the multidimensional variant of the problem that match the known requirements in the

pure synchronous and asynchronous models [28, 32]. Returning to the non-approximate version, Constantinescu et al. [11] have provided the tight conditions for network-agnostic BA with convex validity for abstract convex spaces. In this case, the conditions include $n > 2 \cdot t_s + t_a$ or, if no cryptographic setup is available, $n > 3 \cdot t_s$, along with a few additional conditions that depend on the Helly number of the convexity space.

Other takes on network-agnostic BA have been considered, such as *scaling* the validity guarantees with the network conditions: for real-valued inputs, [10] proposes a protocol for BA with median validity guaranteeing that the output is closer to the honest inputs' median when the network is synchronous than when it is asynchronous. This protocol simultaneously matches the optimal closeness guarantees for purely synchronous [27, 31] and purely asynchronous [10] networks. Such validity properties that scale with the network conditions are outside our scope.

Comparison to previous works. As outlined above, the conditions $n > 2 \cdot t_s + t_a$ and, if no cryptographic setup is available, $n > 3 \cdot t_s$, have been proven to be necessary for strong unanimity properties, i.e., stronger than strong unanimity [4]. Our work shows that this is a requirement for weaker validity properties as well, i.e., for any non-trivial property. We find this result surprising especially for *weak validity*: if *all parties are honest* and hold input v, then the output agreed upon must be v. Assuming a public key infrastructure, this property is solvable in the synchronous model for $t_s < n$, as a straightforward application of the Dolev-Strong broadcast protocol [16]. On the other hand, our result implies that if we expect a BA protocol with weak validity to remain secure in the asynchronous model even for no corruptions (i.e., $t_a = 0$), then the synchronous resilience threshold steps down from $t_s < n$ to $t_s < n/2$.

In contrast to the work of [11] regarding network-agnostic BA with convex validity, our results move the difficulty of proving such feasibility results as a whole to only verifying whether a validity condition satisfies our similarity condition. That is, one can show that convex validity satisfies this similarity condition if and only if the necessary Helly number-based conditions of [11] hold. Our impossibility arguments diverge: we investigate these under *any* validity property, while the work of [11] considers a fixed validity property but also shows impossibility under weaker agreement requirements. On the other hand, our protocol matching our lower bounds is a more general variant of the protocol of [11].

Our paper provides a characterization similar to those of [7, 8] for the synchronous and partially synchronous settings, respectively. The key difference is that these two models allow for deterministic protocols, while the network-agnostic model inherently requires randomization for achieving BA when $t_a > 0$ [19]. Consequently, the focus shifts towards randomized protocols, requiring our proofs to employ different techniques. While the arguments behind the *containment* condition of [7] can be easily adapted for randomized protocols, this is not immediate for their proof that $n > 3 \cdot t_s$ is necessary when no cryptographic setup is available. Our proof for this lower bound, in fact, assumes the synchronous setting and, therefore, strengthens the characterization of [7]. Summing up, their necessary conditions now hold even for randomized protocols, and, as shown in their paper, they can be matched by deterministic protocols.

2 Preliminaries

We consider a setting with n parties $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ running a protocol in a fully-connected network, where links model authenticated channels. We will be working in the *network-agnostic* model: the network may be synchronous, or asynchronous, and the parties

are not aware a priori of the type of network. If the network is synchronous, the parties hold perfectly synchronized clocks and each message is delivered within a publicly known amount of time Δ. Otherwise, if the network is asynchronous, messages can get delayed for an arbitrary amount of time, and clocks may not be synchronized.

Adversary. We assume a central adversary that may corrupt up to t_s of the n parties if the network is synchronous, and up to $t_a \leq t_s$ parties if the network is asynchronous. Corrupted parties permanently become byzantine, and hence may deviate arbitrarily (maliciously) from the protocol. The adversary additionally controls the message delivery schedule, subject to the conditions of the network type. Our impossibility results assume a static adversary (i.e., chooses which parties to corrupt at the beginning of the protocol's execution). Our protocol, on the other hand, provides security even against an adaptive adversary (i.e., chooses which parties to corrupt at any point in the protocol's execution).

Cryptographic setup. We will consider both settings with and without cryptographic setup. Protocols assuming a cryptographic setup will make use of a public key infrastructure (PKI) and a secure signature scheme, and only hold against a computationally bounded adversary. For simplicity of presentation, we assume that the signatures are perfectly unforgeable.

Byzantine Agreement and Validity. A Byzantine Agreement (BA) protocol assumes that each (honest) party holds a value $v_{\text{IN}} \in V_{\text{IN}}$ as input, and enables the parties to agree on a common output $v_{\text{OUT}} \in V_{\text{OUT}}$ satisfying a given validity condition. We assume that V_{IN} is at most countably infinite. In the full version of our paper [9, Section 7], we explain why such an assumption is actually necessary to get any relevant result. In the following, we present each property that a BA protocol needs to satisfy. The first property is *termination*, which may be deterministic (for synchronous protocols) or probabilistic while the second is *agreement*. We define them below:

- **(Termination)** Every honest party decides on an output v_{OUT}.
- **(Probabilistic Termination)** As time goes to infinity, the probability that an honest party has not yet decided on an output value v_{OUT} tends to 0.
- **(Agreement)** If two honest parties output v_{OUT} and v'_{OUT}, then $v_{\text{OUT}} = v'_{\text{OUT}}$.

Before describing the validity property, we need to define *input configurations*. Regardless of the nature of the adversary, these are defined by looking at the honest parties' inputs *after* the adversary has decided which parties to corrupt. Hence, in impossibility results, where we consider a static adversary, input configurations are defined before the protocol's execution. An *input configuration* is a set $I \subseteq \mathcal{P} \times V_{\text{IN}}$ consisting of pairs of honest parties with their inputs: if $(v, P) \in I$, then P is an honest party with input v. Naturally, no party occurs twice in I (i.e., honest parties cannot simultaneously have two inputs). We use the notation $\text{PARTIES}(I)$ to refer to the set of (honest) parties in the input configuration I. Note that, if $P \notin \text{PARTIES}(I)$, then P is corrupted in the input configuration I. Let $\mathcal{I} = \{\text{input configurations } I \subseteq \mathcal{P} \times V_{\text{IN}} \text{ such that } |I| \geq n - t_s\}$ denote the set of all possible input configurations. We also note the inclusion relation for input configurations: for $I, J \in \mathcal{I}$, $J \subseteq I$ if and only if $\text{PARTIES}(J) \subseteq \text{PARTIES}(I)$ and the parties in $\text{PARTIES}(J)$ have the same input value in both I and J. We say that an input configuration I is maximal if $\text{PARTIES}(I) = \mathcal{P}$. Moreover, as V_{IN} is at most countably infinite, the size of $\mathcal{I}$ is at most countably infinite. A validity property is then defined by a mapping $\text{VAL} : \mathcal{I} \to 2^{V_{\text{OUT}}}$ from honest parties' inputs to *valid* outputs:

- **(Validity)** If $I \in \mathcal{I}$ is the input configuration defined by the honest parties and their inputs, then no honest party outputs $v_{\text{OUT}} \notin \text{VAL}(I)$.

This validity definition matches the one used in [6, 7]. We say that a validity property VAL is *trivial* if $\bigcap_{I \in \mathcal{I}} \text{VAL}(I) \neq \varnothing$. Note that, if this condition holds, we can achieve validity, agreement, and termination with no communication: parties output a value in $\bigcap_{I \in \mathcal{I}} \text{VAL}(I)$.

A validity property VAL is *solvable* if there is a BA protocol *solving* VAL, as defined below.

▶ **Definition 1.** *A protocol* Π *is a* (t_s, t_a)*-secure BA protocol solving a validity property* VAL *if it achieves probabilistic termination, agreement, and validity for the given property* VAL *even when up to* t_s *parties are corrupted if it runs in a synchronous network, and even when up to* t_a *parties are corrupted if it runs in an asynchronous network.*

▶ **Definition 2.** *A protocol* Π *is a* t_s*-secure BA protocol solving a validity property* VAL *if, when running in a synchronous network, it achieves (probabilistic) termination, agreement, and validity for the given property* VAL *even when up to* t_s *parties are corrupted.*

One might be tempted to believe that a $(t_s, 0)$-secure protocol is simply a t_s-secure protocol, but the difference is rather subtle. Namely, a $(t_s, 0)$-secure *network-agnostic* BA protocol also provides guarantees in an asynchronous network if all parties are honest. On the other hand, a t_s-secure (synchronous) BA protocol is not required to provide any guarantees if the synchrony assumptions fail, even if there are no corruptions. We add that this subtle difference would not apply to a purely asynchronous variant of the definition, which is equivalent to (t_a, t_a)-secure *network-agnostic* BA protocol.

Randomness. Our work covers BA protocols which can run in the asynchronous setting. As a result of FLP [19], the protocol considered must be randomized. We consider the randomness as a black box where, at each instant, a party can ask for one or multiple random bits, each set to 0 or 1 uniformly and independently. So the randomness from a party's point of view can be seen as an infinite bitstring being progressively read. There may also be shared randomness, which gives the same result to each party. Therefore, when running a protocol, we can consider its behavior over a probabilistic space $(\Omega, \mathcal{F}, \mu)$ where $\Omega = \left(\{0,1\}^{\mathbb{N}}\right)^k$ for some $k > 0$ is the set of all possible random bits parties will read. $\mathcal{F}$ is the σ-algebra generated by taking all possible prefixes from each bitstring, and μ is the resulting probability measure from having each bit following independently a Bernoulli random variable $\mathcal{B}(0.5)$. From this point on, when mentioning probabilities, like almost surely properties, we refer to the probabilistic space given above.

Executions. For a protocol Π, we define an *execution* ε to be a particular feasible run of Π. In particular, ε contains the input configuration $I \in \mathcal{I}$ from which the protocol started, the behavior of the byzantine parties, and the scheduler's behavior (including whether the network was synchronous or asynchronous). Because an execution depends on the randomness, it is actually a random variable $\varepsilon(\omega)$. We then say that an execution *decides* if all honest parties in the execution eventually decide an output. We note that, given a protocol satisfying probabilistic termination, including the scheduler and strategy of the adversary, an execution for this protocol decides almost surely. We will also be using the term *canonical* to refer to executions occurring in a synchronous network where all messages are delivered exactly Δ units of time after being sent and where all corrupted parties crash right at the beginning of the protocol (i.e., they do not send any messages). We add that, for any given input configuration, there is a unique canonical execution (which, recall, is a random variable).

For a given protocol Π and randomness $\omega \in \Omega$, we say that two executions $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ *cannot be distinguished* by a party P that is honest in both executions if it has the same initial state in $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ (i.e., input and randomness), and receives in both $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ the same messages at the same times. As a consequence, if P cannot distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$, then P is in exactly the same state at any time T in $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$. Hence, if P decides a value v_{OUT} at time T in one of the two executions, then it also decides v_{OUT} at time T in the other. We also say that an execution $\varepsilon(\omega)$ is deciding if all honest parties eventually decide when running with randomness ω. If a protocol satisfies probabilistic termination, then an execution is almost surely deciding.

Validity in indistinguishable executions. Goren et al. [25] offer a framework to formalize indistinguishability for randomized protocols. We instead decided to take a different approach by looking at deterministic indistinguishability after fixing the randomness $\omega \in \Omega$. This can allow for easier results as we do not have to consider the whole probabilistic space at a time. For example, some of our proofs require us to define multiple executions depending on ω and cannot be achieved with the framework above. The drawback is that all the assumptions made must hold *almost surely* for the proofs to be correct. Indistinguishability is a powerful tool, especially when considering scenarios where byzantine parties follow the protocol correctly, but with inputs of their own choice. This leads us to the following lemma. The proof is included in Appendix A.

▶ **Lemma 3.** *Let* VAL *be a validity property and* Π *be a* (t_s, t_a)*-resilient* BA *protocol solving* VAL*. Consider two input configurations* I, J *such that* $J \subseteq I$*. Then, the value agreed upon in any execution of* Π *which decides and where the input configuration is* I *must be in* $\text{VAL}(J)$*.*

Intuitively, Lemma 3 ensures that honest parties cannot distinguish between a scenario where all parties in $\text{PARTIES}(I)$ are honest, and one where the parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ are, in fact, byzantine. With the lemma in mind, for any validity property VAL, we can define a new validity property VAL′ such that $\text{VAL}'(I) = \bigcap_{J \subseteq I} \text{VAL}(J)$ for all $I \in \mathcal{I}$. Property VAL′ is simultaneously a stronger version of VAL, in that for all $I \in \mathcal{I}$ we have $\text{VAL}'(I) \subseteq \text{VAL}(I)$, but it also has the property of being monotonically *decreasing*, in that for $J \subseteq I$ we have $\text{VAL}'(I) \subseteq \text{VAL}'(J)$. Armed as such, the previous lemma has the following immediate corollaries:

▶ **Corollary 4.** *A solvable validity property* VAL *is trivial if and only if it permits deciding the same value for all maximal input configurations, i.e.,* $\bigcap_{I \in \mathcal{I}, \text{PARTIES}(I) = \mathcal{P}} \text{VAL}(I) \neq \varnothing$*.*

We end by stating a technical lemma that will be of use in the proofs presented in the subsequent sections. The proof of Lemma 5 is included in Appendix A.

▶ **Lemma 5.** *Let* VAL *be a solvable validity property and* Π *a protocol solving* VAL*. Let* $I_1 \in \mathcal{I}$ *be a maximal input configuration. If, for every maximal input configuration* $I_2 \in \mathcal{I}$*, the canonical executions of* Π *for* I_1 *and* I_2 *decide the same value almost surely, then* VAL *is trivial.*

3 Lower Bound on n

In this section, we prove that, for any non-trivial validity property, the condition $n > 2 \cdot t_s + t_a$ is necessary for it to be solvable in the network-agnostic model even if cryptographic setup is available. Our proof will be organized as follows: we start with a preliminary lemma in Section 3.1, which focuses on a simplified setting where $n := 2$. In this setting, at most one

party can crash if the network is synchronous, and both parties are honest if the network is asynchronous. Our preliminary lemma will show a somewhat counter-intuitive result: roughly, the value decided upon in canonical executions is independent of the honest parties' inputs. In Section 3.2, we move from the simplified setting with two parties to a warm-up variant of our main proof. We will be working with n parties, but we will only consider the particular case $t_a := 0$. By reducing this case to our preliminary lemma, we show that the condition $n > 2 \cdot t_s$ is necessary in this setting. Finally, Section 3.3 focuses on the general case, showing that $n > 2 \cdot t_s + t_a$ is necessary by reducing to our preliminary lemma. The main proof will be a more general version of the warm-up proof.

We note that the proofs by Civit et al. [7] rely on reducing any non-trivial validity property to weak validity, enabling lower bounds to focus solely on weak validity. However, their reduction is invalid for randomized protocols and hence cannot be used in our setting.

3.1 Preliminary Lemma

As previously mentioned, our preliminary lemma focuses on a simplified setting with $n := 2$ parties in the network-agnostic model. When the network is synchronous, we allow the adversary to corrupt up to one party ($t_s := 1$). We restrict the adversary's capabilities by only allowing the corrupted party to crash. When the network is asynchronous, both parties are honest ($t_a := 0$). Then, we assume a protocol achieving (probabilistic) termination and agreement in this setting. We show that, almost surely, once randomness is fixed, all canonical executions decide the same value. Note that there may be a set of randomnesses such that some canonical executions do not even decide, but the probability of picking a randomness in this set is zero.

▶ **Lemma 6.** *Assume $n := 2, t_s := 1$, $t_a := 0$, and that corrupted parties are only allowed to crash. Consider a protocol A that achieves probabilistic termination and agreement in this setting. Then, almost surely, there exists a value v such that all canonical executions of A decide v when running.*

Proof. Denote the two parties by P_1 and P_2. Let $\omega \in \Omega$. For every input configuration, we will define a finite number of executions using randomness ω, including the required canonical executions. Under the assumption that all executions defined for ω decide, we show that all canonical executions using randomness ω decide the same value v. From here, our proof proceeds as follows: individually, each defined execution decides almost surely (by construction). The number of executions defined for each input configuration is finite, and the set of input configurations is countable (since V_{IN} is countable). A countable intersection of events holding almost surely holds almost surely. Then, almost surely, all defined executions for ω decide, i.e., our additional assumption holds almost surely, completing the proof.

Hence, from now on, we fix $\omega \in \Omega$ and only consider executions running with this randomness ω. We assume that all executions we consider decide, and we want to show that all canonical executions decide the same value v.

We first consider canonical executions where one of the parties crashes. Let $v_1, v_2 \in V_{\text{IN}}$ be two arbitrary values. First, we consider the canonical execution $\varepsilon_1(\omega)$ of A (in a synchronous network) where P_1 is honest and has input v_1. P_2 is corrupted and crashes at the beginning of the execution. We have assumed that $\varepsilon_1(\omega)$ is deciding, hence P_1 outputs some value v_1' within a finite amount of time $T_1(\omega)$ (note: without receiving any messages from P_2). Second, consider the canonical execution $\varepsilon_2(\omega)$ of A (again, in a synchronous network) where P_2 is honest and has input v_2. P_1 is corrupted and crashes at the beginning of the execution. We

have assumed that $\varepsilon_2(\omega)$ is deciding, hence P_2 outputs some value v_2' within a finite amount of time $T_2(\omega)$. We prove that $v_1' = v_2'$ by defining a third execution $\varepsilon_{1,2}(\omega)$, but, this time, in the asynchronous model. Here, both P_1 and P_2 are honest with inputs v_1 and v_2 respectively. The scheduler delays any message between them until time $\max(T_1(\omega), T_2(\omega))$. This way, P_1 cannot distinguish execution $\varepsilon_{1,2}(\omega)$ from execution $\varepsilon_1(\omega)$, and therefore it outputs v_1' in $\varepsilon_{1,2}(\omega)$ as well. Similarly, P_2 outputs v_2' in $\varepsilon_{1,2}(\omega)$. Recall that A achieves agreement, hence $v_1' = v_2'$. Note that the argument holds for arbitrary $v_1, v_2 \in V_{\text{IN}}$, so we get that *all* canonical executions where one of the parties crashes decide the same value, which we denote by v.

We now consider canonical executions where none of the parties crashes and show that all such executions also decide v. Consider any $v_1, v_2 \in V_{\text{IN}}$ and the canonical execution $\varepsilon_M(\omega)$ of A where both parties are honest and have inputs v_1 and v_2. We also define $\varepsilon_{1,2}(\omega)$ as before. From the previous part, it follows that $\varepsilon_{1,2}(\omega)$ decides v. We will now show that $\varepsilon_M(\omega)$ decides v as well. Based on our assumption, we already know that $\varepsilon_M(\omega)$ decides, so it does so after a finite number of messages ℓ. For the next part of the proof, we first give an intuitive outline. Roughly, we will build a chain of $\ell + 1$ scenarios between execution $\varepsilon_{1,2}(\omega)$ and execution $\varepsilon_M(\omega)$: in each scenario $1 \leq m < \ell$, the network is asynchronous, and the scheduler ensures that the first m messages are received synchronously, while all subsequent messages are delayed sufficiently long. Scenarios 0 and ℓ correspond to executions $\varepsilon_{1,2}(\omega)$ and $\varepsilon_M(\omega)$ respectively, and any two consecutive scenarios will be indistinguishable to the party that sent the last message: this will imply that $\varepsilon_M(\omega)$ also decides v. In the following, we write this idea formally.

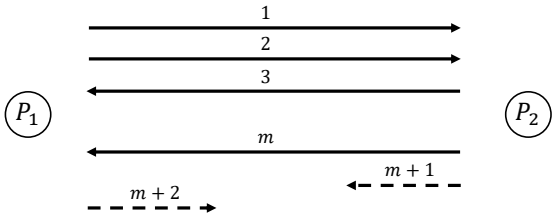

Figure 1 Example of execution $\varepsilon_{M,m}(\omega)$: Any messages sent after the first m messages get delayed until after a value has been decided.

For $0 \leq m \leq \ell$, we consider an execution $\varepsilon_{M,m}(\omega)$ of A in the asynchronous model: both P_1 and P_2 are honest, and they have inputs v_1 and v_2 respectively. In execution $\varepsilon_{M,0}(\omega)$, the scheduler uses the same strategy as that of execution $\varepsilon_{1,2}(\omega)$: the scheduler delays any message in execution $\varepsilon_{M,0}(\omega)$ until time $T = \max(T_1(\omega), T_2(\omega))$. Hence, P_1 and P_2 cannot distinguish between executions $\varepsilon_{M,0}(\omega)$ and $\varepsilon_{1,2}(\omega)$, implying that $\varepsilon_{M,0}(\omega)$ decides v by some time T_0. In execution $\varepsilon_{M,1}(\omega)$, the scheduler allows the first message to be delivered after exactly Δ time. Assume without loss of generality that this message is sent by P_1. All further messages are delayed until time $T_1 > T_0$ (we define the exact time later): hence, P_1 cannot distinguish from this execution and execution $\varepsilon_{M,0}(\omega)$, therefore it outputs v by time T_0, as in execution $\varepsilon_{M,0}(\omega)$. P_2, on the other hand, can distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_{M,0}(\omega)$: however, it cannot distinguish between $\varepsilon_{M,1}(\omega)$ and an execution where the network is, in fact, synchronous, and P_1 has crashed. If P_1 has crashed, P_2 has to output eventually, hence by time T_1'. Hence, the scheduler makes sure to delay all messages until time $T_1 > \max(T_0, T_1')$. Consequently, P_2 has to output v in $\varepsilon_{M,1}(\omega)$ by time T_1. The next executions are defined similarly: in execution $\varepsilon_{M,m}(\omega)$ with $m > 0$, the scheduler allows the

first m messages to be delivered after exactly Δ units of time. The remaining messages are delivered sufficiently late to ensure that the last message's sender is unable to distinguish between $\varepsilon_{M,m}(\omega)$ and execution $\varepsilon_{M,m-1}(\omega)$. Thus, the last message's sender outputs v in execution $\varepsilon_{M,m}(\omega)$, which forces the other honest party to also output v.

We remark that executions $\varepsilon_{M,\ell}(\omega)$ and $\varepsilon_M(\omega)$ are indistinguishable, so $\varepsilon_M(\omega)$ will decide v. Since this holds for arbitrary values v_1, v_2, we have obtained that all canonical executions where no party crashes also output v, hence completing the proof. ◀

3.2 Warm-up: $t_a := 0$

As a warm-up towards the main result, we focus on the particular case $t_a := 0$, and show that the condition $n > 2 \cdot t_s$ is necessary. Intuitively, when $n \leq 2 \cdot t_s$, one cannot distinguish between a scenario where half of the parties crash in a synchronous network, and a scenario where half of the parties are honest but delayed in an asynchronous one. This way, the presence of the asynchronous case, even with no corruptions, allows two disjoint sets of honest parties to only run the protocol within their own set. Since the two sets run the protocol independently, the honest parties agree on a valid value only if the given validity property is trivial. Note that this is the case even for weak validity, which can be solved in the synchronous model for up to $t_s < n$ corruptions. Our result implies that expecting a synchronous protocol to provide guarantees when it runs in a corruption-free asynchronous network impacts the overall resilience.

▶ **Theorem 7.** *Assume $t_s > 0$ and consider a validity property* VAL*. If $n = 2 \cdot t_s$ and there is a $(t_s, 0)$-secure BA protocol solving* VAL*, then* VAL *is trivial.*

Proof. Consider a (possibly randomized) $(t_s, 0)$-secure BA protocol Π that solves VAL when $n = 2 \cdot t_s$. Let I_1 and I_2 be two arbitrary maximal input configurations. We show that the canonical executions of I_1 and I_2 almost surely decide on the same output when run using the same randomness. Then, Lemma 5 ensures that VAL is trivial.

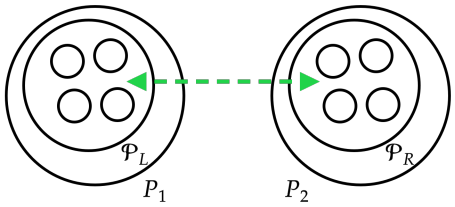

Figure 2 P_1 simulates all parties of $\mathcal{P}_L$ while P_2 simulates all parties of $\mathcal{P}_R$. All the parties of $\mathcal{P}$ communicate between one another not knowing they are being simulated.

We use Π to build a 2-party protocol A that matches the setting described in Lemma 6. For protocol A, we consider the input set $\{0, 1\}$ and output set V_{OUT}, and we denote the two parties running A by P_1 and P_2. Since $n = 2 \cdot t_s$, we may partition the set of n parties $\mathcal{P}$ into two sets $\mathcal{P}_L$ and $\mathcal{P}_R$ of t_s parties each.

Then, as shown in Figure 2, P_1 will simulate the parties in $\mathcal{P}_L$, while P_2 will simulate the parties in $\mathcal{P}_R$. Concretely, in protocol A, P_1 proceeds as follows:

- P_1 simulates all t_s parties of $\mathcal{P}_L$ running Π.
- If P_1 has input 0, then the simulated parties in $\mathcal{P}_L$ use their inputs from I_1. Otherwise, they use their inputs from I_2.

- Messages between the simulated parties in $\mathcal{P}_L$ are received after exactly Δ units of time, and P_1 forwards to P_2 every message sent by a simulated party in $\mathcal{P}_L$ to a party in $\mathcal{P}_R$. Once P_2 receives this message, it immediately forwards it to the simulated receiver in $\mathcal{P}_R$.
- As soon as a party in $\mathcal{P}_L$ decides a value, P_1 decides this value. P_1 continues forwarding messages as described above.

P_2 proceeds identically to P_1, switching $\mathcal{P}_L$ and $\mathcal{P}_R$.

Note that running A in an asynchronous network where both P_1 and P_2 are honest corresponds to running Π in an asynchronous network where all n parties are honest. In addition, running A in a synchronous network where at most one party may crash corresponds to running Π in a synchronous network where at most t_s parties may crash. Then, since Π is a $(t_s, 0)$-secure BA protocol, A achieves probabilistic termination and agreement in the setting described in Lemma 6. Then, applying Lemma 6, we obtain that: almost surely, there is a value v such that all canonical executions of A decide on the same value v. Moreover, the canonical execution of A with input values $(0, 0)$ matches the canonical execution of I_1 for Π. Similarly, the canonical execution of A with input $(1, 1)$ matches the canonical execution of I_2. As a consequence, canonical executions of I_1 and I_2 decide the same value almost surely. Using Lemma 5, we can therefore conclude that VAL is trivial. ◀

3.3 General Result

We are now ready to prove the final statement of this section, presented below.

▶ **Theorem 8.** *Consider a validity property* VAL*, and t_s, t_a such that $t_s > 0$ and $t_s \geq t_a$. If there is a (t_s, t_a)-secure BA protocol solving* VAL *when $n \leq 2 \cdot t_s + t_a$, then* VAL *is trivial.*

Proof. Assume $n = 2 \cdot t_s + t_a$ and that there is a (t_s, t_a)-secure BA protocol Π solving VAL. We consider two input configurations I_1 and I_2 such that $\text{PARTIES}(I_1) = \text{PARTIES}(I_2) = \mathcal{P}$ (i.e., all parties are honest). We show that, almost surely, the canonical executions of I_1 and I_2 decide on the same output. Then, Lemma 5 ensures that VAL is trivial. Similarly to the proof of Theorem 7, we use Π to build a two-party protocol A that matches the setting of Lemma 6. For A, we consider the input space $\{0, 1\}$ and the output space V_{OUT}.

This time, we partition $\mathcal{P}$ into three sets $\mathcal{P}_L$, $\mathcal{P}_M$ and $\mathcal{P}_R$ such that $|\mathcal{P}_L| = |\mathcal{P}_R| = t_s$ and $|\mathcal{P}_M| = t_a$. A crucial difference from the proof of Theorem 7 is that, as shown in Figure 3, each party in $\mathcal{P}$ simulates its own copy of the parties in $\mathcal{P}_M$. Our construction will ensure that, when A runs in the synchronous setting, the two simulated copies of each party in $\mathcal{P}_M$ will be in the exact same state at any point, maintaining the guarantees of Π when at most t_s of the parties are corrupted. Meanwhile, in the asynchronous setting, the copies of the t_a parties in $\mathcal{P}_M$ may be in different states, but Π is able to tolerate t_a byzantine parties in this case. Concretely, in protocol A, party P_1 proceeds as follows:

- P_1 simulates all $t_s + t_a$ parties of $\mathcal{P}_L \cup \mathcal{P}_M$ running Π.
- If P_1 has input 0, then parties in $\mathcal{P}_L \cup \mathcal{P}_M$ take their input from I_1. Otherwise, they take their input from I_2.
- As depicted in Figure 4, the messages sent between the parties simulated by P_1 are exchanged as if all $t_s + t_a$ parties are running independently: each such message is delivered after exactly Δ units of time. Additionally, once a simulated party in $\mathcal{P}_L$ sends a message to a simulated party in $\mathcal{P}_M$, P_1 immediately forwards this message to P_2 as well. Once P_2 receives this message, it immediately forwards it to its own simulated receiver in $\mathcal{P}_M$. The message delay here depends on the type of network that A is running in.
- Messages from a party in $\mathcal{P}_L$ to $\mathcal{P}_R$ are sent from P_1 to P_2. P_2 then forwards each such message to its simulated receiver in $\mathcal{P}_R$.

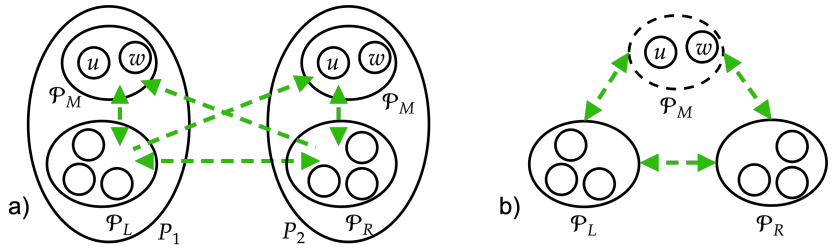

Figure 3 a) P_1 simulates all parties of $\mathcal{P}_L$ and its own copies of the parties in $\mathcal{P}_M$. P_2 simulates all parties of $\mathcal{P}_R$, and its own copy of the parties $\mathcal{P}_M$. The arrows show which simulated group of parties can send messages to which other group.
b) This is how the network looks like from the point of view of a party in $\mathcal{P}_R$ or $\mathcal{P}_M$: they are unaware of the second set $\mathcal{P}_M$ being simulated in parallel.

- P_1 discards any messages sent by the simulated parties in $\mathcal{P}_M$ to parties in $\mathcal{P}_R$.
- As soon as a party in $\mathcal{P}_L$ (but not $\mathcal{P}_M$) decides a value, P_1 decides this value. P_1 continues forwarding messages as described above until all its simulated parties terminate.

The behavior of P_2 is the same as the behavior of P_1, switching $\mathcal{P}_L$ and $\mathcal{P}_R$.

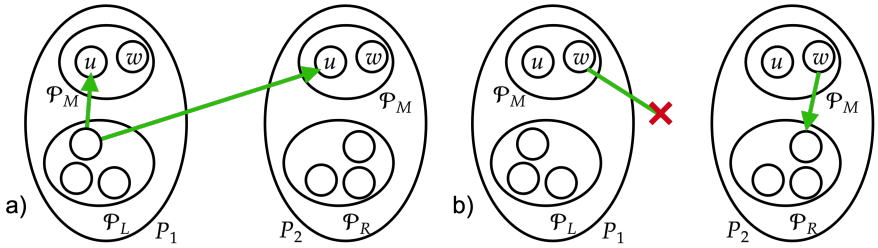

Figure 4 Examples of how different messages are handled.
a) If a simulated party in $\mathcal{P}_L$ wants to send a message to a party in $\mathcal{P}_M$, then two identical messages are actually sent: the first one to the copy of the receiver simulated by P_1, and the second to the copy simulated by in P_2.
b) If a party in $\mathcal{P}_M$ wants to send a message to a party in $\mathcal{P}_R$, if the two parties are simulated by the same entity (here P_2), then the message gets received as expected. Otherwise, the message is discarded and the party in $\mathcal{P}_R$ never receives it.

We now need to analyze A in the setting described by Lemma 6. Running A in an asynchronous network where both parties are honest corresponds to running Π in a network where the communication between parties in $\mathcal{P}_L$ and $\mathcal{P}_R$ is asynchronous. While the simulated copies in $\mathcal{P}_M$ are not necessarily consistent, Π is resilient against $t_a = |\mathcal{P}_M|$ byzantine corruptions, and therefore maintains its guarantees. Hence, A achieves agreement and probabilistic termination in this setting. It remains to show that A also achieves these guarantees in a synchronous network where one of the two parties may crash. Settings where both P_1 and P_2 are honest correspond to running Π in a synchronous network where all n parties are honest. We note that, in this case, for each party in $\mathcal{P}_M$, the copy simulated by P_1 and the copy simulated by P_2 maintain the same state at all times. Settings where at most one of P_1 and P_2 may crash correspond to running Π in a synchronous setting where the t_s parties meant to be simulated only by the party crashing in A are corrupted. As Π tolerates t_s corruptions, it maintains its guarantees. Hence, A achieves agreement and probabilistic termination in this setting as well.

We conclude the proof in an identical manner to the proof of Theorem 7. Applying Lemma 6, we obtain that, almost surely, all canonical executions of A decide on the same value. Moreover, the canonical execution of A with input values $(0, 0)$ matches the canonical execution of I_1 for Π. Similarly, the canonical execution of A with input $(1, 1)$ matches the canonical execution of I_2. As a consequence, canonical executions of I_1 and I_2 decide the same value almost surely. Using Lemma 5, we can therefore conclude that VAL is trivial. ◀

4 Lower Bound on n Without Cryptographic Setup

The previous section has proven that the condition $n > 2 \cdot t_s + t_a$ is necessary regardless of whether a cryptographic setup is available or not. We now focus on settings without cryptographic setup and prove an even stronger condition. We show that, in such settings, the condition $n > 3 \cdot t_s$ is necessary even in the synchronous model. Since a protocol achieving (t_s, t_a)-secure BA in the network-agnostic model also achieves t_s-secure BA in the synchronous model, this bound immediately applies to the network-agnostic model. We add that our result extends the requirement of $n > 3 \cdot t_s$ provided by Civit et al. [7] for synchronous deterministic protocols to cover randomized protocols as well.

4.1 Preliminary Lemma

Similarly to the outline of Section 3, we first consider a setting with three parties, out of which at most one is byzantine. Afterwards, we focus on the general case. Lemma 9 is an improvement over the result of Fischer, Lynch, and Merritt for weak validity [18]: the result of [18] only applies for weak validity and protocols must always decide in a finite amount of time, which is a lot stronger than probabilistic termination. Our proof uses the main core idea but improves it to lift these restrictions. Roughly, we will be running A in a larger ring containing multiple copies of each party, as depicted in Figure 5.

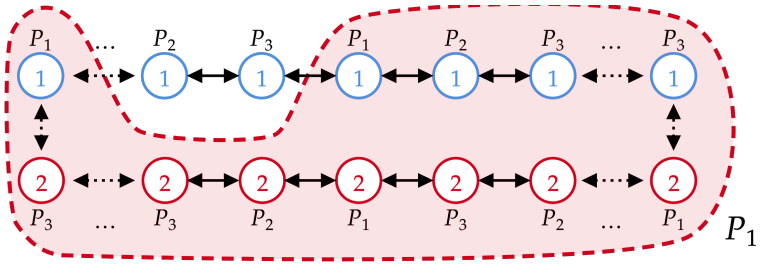

Figure 5 Defining the behavior of a byzantine party (here P_1).

The ring is constructed from two canonical executions with different input configurations. Parties adjacent in this ring cannot distinguish between the ring and the original setting of three parties, as the third party may be byzantine and simulate the rest of the ring. This forces parties adjacent in the ring to output the same value, which implies that the two original executions lead to the same output. We defer the formal proof to Appendix B.1.

▶ **Lemma 9.** *Consider $n := 3$ parties in a synchronous network, and assume a protocol A that achieves (probabilistic) termination and agreement in this setting even when up to one party is byzantine. Then, almost surely, all canonical executions of A decide the same value.*

4.2 General Result

To prove our main result in this setting, we use a strategy similar to the proof of Theorem 7. Concretely, we assume an n-party protocol that achieves VAL whenever $n \geq 3 \cdot t_s$ and use it to construct a three-party protocol that contradicts Lemma 9. We defer the formal proof to Appendix B.2.

▶ **Theorem 10.** *Assume $t_s > 0$ and consider a validity property VAL. If there is a t_s-secure BA protocol solving VAL in the synchronous model when no cryptographic setup is available and $n \leq 3 \cdot t_s$, then VAL is trivial.*

Then, as any lower bound in the synchronous model also holds in the network-agnostic model, Theorem 10 immediately implies the following corollary.

▶ **Corollary 11.** *Consider a validity property VAL, and let t_s, t_a such that $t_s > 0$ and $t_s \geq t_a$. If there is a (t_s, t_a)-secure BA protocol solving VAL when no cryptographic setup is available and $n \leq 3 \cdot t_s$, then VAL is trivial.*

5 Similarity Condition

The lower bounds on n presented in the previous sections are indeed necessary, but not yet sufficient. We may instantiate, for instance, the input space as the (finite) set of vertices of a (publicly known) labeled graph with maximum clique size $\omega \geq 3$. We consider convex validity, under the so-called *monophonic convexity*. For this example, network-agnostic BA requires the stronger lower bound $n > \max(\omega \cdot t_s, \omega \cdot t_a + t_s, 2 \cdot t_s + t_a)$ [11]. Roughly, if any of the conditions $n > \omega \cdot t_s$ and $n > \omega \cdot t_a + t_s$ fails to hold, one can find scenarios where the simple presence of byzantine parties (following the protocol correctly, with inputs of their choice) prevents the honest parties from obtaining a valid output. In this section, we prove the need of one more condition that captures these validity-dependent requirements, and enables the honest parties to find a valid output even if their view over the honest inputs is not accurate. This additional condition matches the *similarity* condition of [8] for the partially synchronous model, and the *containment* condition of [7] in the synchronous model.

We need to establish a few notions. The first is the notion of *neighbors* of an input configuration. The neighbors of I, denoted by $\text{NEIGHBORS}(I)$, are the input configurations J such that the parties in $\text{PARTIES}(I) \cap \text{PARTIES}(J)$ hold the same input values in I and J. Formally, $\text{NEIGHBORS}(I) := \{J \in \mathcal{I} : \ \forall P \in \mathcal{P}, \text{ if } (v_1, P) \in I \text{ and } (v_2, P) \in J \text{ then } v_1 = v_2\}$. The definition of neighbors is symmetric (if $J \in \text{NEIGHBORS}(I)$ then $I \in \text{NEIGHBORS}(J)$).

The second notion is that of *similar configurations* of an input configuration I, denoted by $\text{SIMILAR}(I)$. These are input configurations that may be indistinguishable from I. In the synchronous model, these are configurations $J \in \text{NEIGHBORS}(I)$ such that $J \subseteq I$. Roughly, this models scenarios where some of the parties are corrupted, but follow the protocol correctly with inputs of their own choice. In the asynchronous model, these are configurations $J \in \text{NEIGHBORS}(I)$ containing $n - t_a$ parties: this additionally takes into account that at most t_a honest parties may be isolated due to network delays. Hence, we define $\text{SIMILAR}(I)$ as $\text{SIMILAR}(I) = \{J \in \text{NEIGHBORS}(I) : J \subseteq I\} \bigcup \{J \in \text{NEIGHBORS}(I) : |J| \geq n - t_a\}$.

We may now define the similarity condition, which Lemma 13 proves to be necessary for the network-agnostic model. Lemma 13 adapts Theorem 2 of [8] and Lemma 8 of [7] to the network-agnostic model, and it relies on standard indistinguishability arguments. We defer the proof to Appendix C.

▶ **Definition 12** (Similarity condition). *We say that a validity property* VAL *satisfies the* similarity condition *if there is a Turing-computable function* $\sigma : \mathcal{I} \mapsto V_O$ *such that, for any input configuration* $I \in \mathcal{I}$, $\sigma(I) \in \bigcap_{J \in SIMILAR(I)} \text{VAL}(J)$.

Note that the existence of such a function σ also implies $\forall I \in \mathcal{I}, \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J) \neq \varnothing$.

▶ **Lemma 13.** *If a validity property* VAL *is solvable in the network-agnostic model, then* VAL *satisfies the similarity condition.*

It is worth noting that, in the proof of this lemma, the deterministic Turing function σ is defined based on a randomized protocol that solves VAL. This is justified because, as discussed in the proof, when time complexity is not taken into account, probabilistic Turing machines are as expressive as deterministic ones.

6 Sufficiency and Main Result

We now show that the conditions presented in the previous sections are not only necessary, but also sufficient, hence proving our main result, stated below.

▶ **Theorem 14.** *Assume a non-trivial validity condition* VAL*. Then, there is a* (t_s, t_a)*-secure protocol solving* VAL *if and only if the following conditions hold:*

- VAL *satisfies the similarity condition.*
- $n > 3 \cdot t_s$ *or, if PKI is available,* $n > 2 \cdot t_s + t_a$.

The remainder of the section describes a protocol that matches our necessary conditions, as stated in the lemma below. Theorem 14 then follows from combining Lemma 15 with the requirements discussed previously: the lower bounds on n, described in Theorem 8 and Corollary 11, plus the similarity condition, proven to be necessary in Lemma 13.

▶ **Lemma 15.** *Assume a validity condition* VAL *that satisfies the similarity condition. Then, if* $n > 3 \cdot t_s$*, or, assuming that PKI is available, if* $n > 2 \cdot t_s + t_a$*, there is a* (t_s, t_a)*-secure* BA *protocol solving* VAL*.*

Our construction behind Lemma 15 generalizes the network-agnostic BA protocol of [11] that solves convex validity. The parties distribute their input values using a protocol achieving *Agreement on a Core-Set* (ACS), which provides them with an identical view over the inputs, i.e., with a potential input configuration. This is a (randomized) communication primitive enabling identical views in the pure asynchronous model [2, 3]. We make use of the ACS definition of [11], included below, which differs from the standard definition by providing stronger properties in the synchronous model: it additionally ensures that all honest inputs are included in the common view if the network is synchronous. This property is essential for matching the higher resilience threshold t_s in a synchronous network. This way, the parties can apply a deterministic decision over the view agreed upon in ACS and obtain an output.

▶ **Definition 16.** *Let* Π *be a protocol where every party* P_i *holds an input* v_i *and outputs a set* $I_i \in \mathcal{I}$ *consisting of at least* $n - t_s$ *value-sender pairs. We consider the following properties in addition to those presented in Section 2:*

- ***Integrity****: If* P_i *and* P_j *are both honest and* $P_i \in$ PARTIES(I_j)*, then* $(P_i, v_i) \in I_j$.
- ***Honest Core****: If an honest party outputs* I*, then* PARTIES(I) *contains all honest parties.*

Then, we say that Π *is a* (t_s, t_a)*-secure* ACS *protocol if it achieves the following:*

- *Probabilistic termination, agreement, integrity, and honest core when running in a synchronous network where up to* t_s *parties are corrupted;*

- *Probabilistic termination, agreement, and integrity when running in an asynchronous network where up to t_a parties are corrupted.*

We make use of the ACS construction of [11], described by the result below.

▶ **Theorem 17** ([11]). *Consider n, t_s, t_a such that $t_a \leq t_s$. If $3 \cdot t_s < n$, or, if PKI is available and $2 \cdot t_s + t_a < n$, there is a (t_s, t_a)-secure ACS protocol Π_{ACS}.*

We may now present the proof of Lemma 15, describing our protocol.

Proof of Lemma 15. Our BA protocol solving VAL proceeds as follows: the parties distribute their input values using the protocol Π_{ACS} described in Theorem 17. Π_{ACS} provides the parties with the same output set $I \in \mathcal{I}$ of at least $n - t_s$ value-sender pairs, representing a potential input configuration. Once the parties obtain this set, they output $\sigma(I)$, where σ is the Turing-computable function provided by VAL satisfying the similarity condition.

(t_s, t_a)-secure BA solving VAL

Code for party P_i with input v_i

1: Join Π_{ACS} with input v_i and obtain the output set I.
2: Output $\sigma(I)$ and terminate.

Regardless of whether the network is synchronous or asynchronous, since Π_{ACS} achieves probabilistic termination, our BA protocol achieves probabilistic termination as well. In addition, since Π_{ACS} achieves agreement, the parties compute their output identically, and therefore our BA protocol achieves agreement.

It remains to prove that the honest parties' output is in $\text{VAL}(H)$, where H denotes the (actual) input configuration (containing only the honest parties and their inputs). Since $\sigma(I) \in \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J)$, it will be sufficient to show that $H \in \text{SIMILAR}(I)$. Regardless of the type of network, the integrity property of Π_{ACS} ensures that $H \in \text{NEIGHBORS}(I)$. If the network is asynchronous, $\left|H\right| \geq n - t_a$ and therefore $H \in \text{SIMILAR}(I)$. Otherwise, if the network is synchronous, the honest core property of Π_{ACS} ensures that honest parties and their inputs are included in I. Then, $H \subseteq I$ and therefore $H \in \text{SIMILAR}(I)$ in this case as well. Thus, as $\sigma(I) \in \bigcap_{H \in \text{SIMILAR}(I)} \text{VAL}(H)$, we get that the value agreed upon is in $\text{VAL}(H)$, which proves the validity of the protocol and concludes the proof. ◀

7 Conclusions and Future Work

We investigated the conditions that a validity property needs to satisfy in order to be solvable in the network-agnostic model and established the necessary and sufficient conditions. Our results demonstrate that solving a non-trivial validity property VAL requires **(i)** that VAL satisfies the *similarity condition*, and **(ii)** that $n > 2 \cdot t_s + t_a$ assuming a public key infrastructure, or $n > 3 \cdot t_s$ otherwise. Further, we provided a universal protocol that solves a given validity property whenever these established conditions are met. Our characterization follows the line of works of [7, 8] focusing on the partially synchronous model and on the synchronous model. At the same time, it generalizes prior results on when network-agnostic BA can be achieved [4, 11] from fixed validity properties to arbitrary validity properties.

While our work provides a complete answer for solvability, we leave a number of exciting problems open. Our results and approach do not hold if we allow the protocol to fail with some probability $\varepsilon > 0$. Focusing on this weaker setting would likely allow more efficient protocols. Future works could also extend our characterization to cover settings where both

V_{IN} and V_{OUT} are uncountable sets and consider proving message complexity lower bounds. Other promising directions would aim to improve the efficiency of our universal protocol, or to generalize our results further to network-dependent validity properties (that allow weaker guarantees if the network is asynchronous) [10], or to weaker agreement definitions [11].

References

1 Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *Principles of Distributed Systems*, pages 229–239, Berlin, Heidelberg, 2005. doi:10.1007/11516798_17.

2 Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 52–61, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167109.

3 Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 183–192, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/197917.198088.

4 Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography Conference*, pages 131–150. Springer, 2019. doi:10.1007/978-3-030-36030-6_6.

5 Zohir Bouzid, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal byzantine-resilient convergence in uni-dimensional robot networks. *Theoretical Computer Science*, 411(34):3154–3168, 2010. doi:10.1016/j.tcs.2010.05.006.

6 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Matteo Monti, and Manuel Vidigueira. Every Bit Counts in Consensus. In *37th International Symposium on Distributed Computing (DISC 2023)*, volume 281, pages 13:1–13:26, Dagstuhl, Germany, 2023. doi:10.4230/LIPIcs.DISC.2023.13.

7 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Anton Paramonov, and Manuel Vidigueira. All byzantine agreement problems are expensive. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, PODC '24, pages 157–169, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3662158.3662780.

8 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. On the validity of consensus. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 332–343, 2023. doi:10.1145/3583668.3594567.

9 Andrei Constantinescu, Marc Dufay, Diana Ghinea, and Roger Wattenhofer. Validity in network-agnostic byzantine agreement. Full version of this paper. doi:10.48550/arXiv.2410.19721.

10 Andrei Constantinescu, Diana Ghinea, Lioba Heimbach, Zilin Wang, and Roger Wattenhofer. A Fair and Resilient Decentralized Clock Network for Transaction Ordering. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*, volume 286, pages 8:1–8:20, Dagstuhl, Germany, 2024. doi:10.4230/LIPIcs.OPODIS.2023.8.

11 Andrei Constantinescu, Diana Ghinea, Roger Wattenhofer, and Floris Westermann. Convex Consensus with Asynchronous Fallback. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319, pages 15:1–15:23, Dagstuhl, Germany, 2024. doi:10.4230/LIPIcs.DISC.2024.15.

12 Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In *Theory of Cryptography Conference*, pages 623–653. Springer, 2021. doi:10.1007/978-3-030-90459-3_21.

13 Giovanni Deligios and Mose Mizrahi Erbes. Closing the efficiency gap between synchronous and network-agnostic consensus. In *Advances in Cryptology – EUROCRYPT 2024*, pages 432–461, 2024. doi:10.1007/978-3-031-58740-5_15.

14 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, May 1986. doi:10.1145/5925.5931.

15 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985. doi:10.1145/2455.214112.

16 Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

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

18 Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. In Michael A. Malcolm and H. Raymond Strong, editors, *4th ACM PODC*, pages 59–70. ACM, August 1985. doi:10.1145/323596.323602.

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

20 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, pages 70–80, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538442.

21 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Multidimensional approximate agreement with asynchronous fallback. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pages 141–151, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558481.3591105.

22 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Brief Announcement: Communication-Optimal Convex Agreement. In *The 43rd ACM Symposium on Principles of Distributed Computing (PODC), Nantes, France*, June 2024.

23 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Communication-optimal convex agreement. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '25, pages 39–49, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3732772.3733551.

24 John T. Gill. Computational complexity of probabilistic turing machines. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 91–95, 1974. doi:10.1145/800119.803889.

25 Guy Goren, Yoram Moses, and Alexander Spiegelman. Probabilistic indistinguishability and the quality of validity in byzantine agreement. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, pages 111–125, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558535.3559789.

26 Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Byzantine Preferential Voting. In *14th Conference on Web and Internet Economics (WINE), Oxford, United Kingdom*, December 2018. doi:10.1007/978-3-030-04612-5_22.

27 Darya Melnyk and Roger Wattenhofer. Byzantine agreement with interval validity. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 251–260, 2018. doi:10.1109/SRDS.2018.00036.

28 Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K Garg. Multidimensional agreement in byzantine systems. *Distributed Computing*, 28(6):423–441, 2015. doi:10.1007/s00446-014-0240-5.

29 Gil Neiger. Distributed consensus revisited. *Information Processing Letters*, 49(4):195–201, 1994. doi:10.1016/0020-0190(94)90011-6.

30 Thomas Nowak and Joel Rybicki. Byzantine Approximate Agreement on Graphs. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146, pages 29:1–29:17, Dagstuhl, Germany, 2019. doi:10.4230/LIPIcs.DISC.2019.29.

31 David Stolz and Roger Wattenhofer. Byzantine Agreement with Median Validity. In *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*, volume 46, pages 1–14, Dagstuhl, Germany, 2016. doi:10.4230/LIPIcs.OPODIS.2015.22.
32 Nitin H. Vaidya and Vijay K. Garg. Byzantine vector consensus in complete graphs. In Panagiota Fatourou and Gadi Taubenfeld, editors, *32nd ACM PODC*, pages 65–73. ACM, July 2013. doi:10.1145/2484239.2484256.

Appendix

A Preliminaries: Missing Proofs

▶ **Lemma 3.** *Let* VAL *be a validity property and* Π *be a* (t_s, t_a)*-resilient* BA *protocol solving* VAL*. Consider two input configurations* I, J *such that* $J \subseteq I$*. Then, the value agreed upon in any execution of* Π *which decides and where the input configuration is* I *must be in* $\text{VAL}(J)$*.*

Proof. Consider a deciding execution ε_I of Π for I. Construct the execution ε_J, which is identical to ε_I except that its input configuration is J instead of I. Observe that ε_J is an execution of Π for J. Indeed, parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ (which are byzantine) can act as if they are honest and behave exactly as they do in ε_I. The honest parties in J cannot distinguish between ε_I and ε_J, so the agreed-upon value in both executions must be the same. Since ε_J is a valid execution of Π for J, this value must be in $\text{VAL}(J)$. ◀

▶ **Lemma 5.** *Let* VAL *be a solvable validity property and* Π *a protocol solving* VAL*. Let* $I_1 \in \mathcal{I}$ *be a maximal input configuration. If, for every maximal input configuration* $I_2 \in \mathcal{I}$*, the canonical executions of* Π *for* I_1 *and* I_2 *decide the same value almost surely, then* VAL *is trivial.*

Proof. Note that a countable intersection of events that happen almost surely happens almost surely. We assume the lemma's condition and want to prove that VAL is trivial. Because V_{IN} is at most countably infinite, the number of maximal input configurations $I_2 \in \mathcal{I}$ is at most countably infinite. By taking an intersection of events, one for each maximal $I_2 \in \mathcal{I}$, that happen almost surely by the lemma condition, we get that, almost surely, for all $I_2 \in \mathcal{I}$ the canonical executions of Π for I_1 and I_2 decide the same value. Hence, because it happens with probability 1, it means we can find $\omega \in \Omega$ such that this event happens. That is, for all maximal $I_2 \in \mathcal{I}$, the canonical executions of Π for I_1 and I_2 with randomness ω decide the same value. This implies that, for all maximal $I \in \mathcal{I}$, the canonical execution of Π on I with randomness ω decides the same value $v \in V_{\text{OUT}}$. Therefore, by Corollary 4, VAL is trivial. ◀

B No Cryptographic Setup: Missing Proofs

B.1 Proof of Lemma 9

We describe the proof of Lemma 9, restated below.

▶ **Lemma 9.** *Consider* $n := 3$ *parties in a synchronous network, and assume a protocol* A *that achieves (probabilistic) termination and agreement in this setting even when up to one party is byzantine. Then, almost surely, all canonical executions of* A *decide the same value.*

To prove our statement, we will be running A in a larger ring containing multiple copies of each party, as depicted in Figure 6. The ring is constructed from two canonical executions with different input configurations. We will show that parties adjacent in this ring cannot

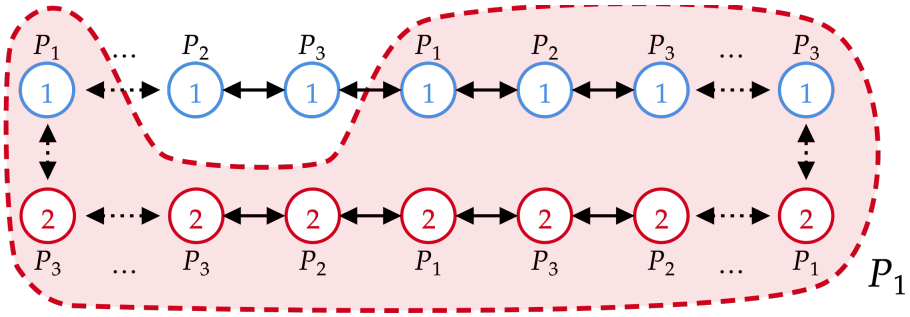

Figure 6 Defining the behavior of a byzantine party (here P_1).

distinguish between the ring and the original setting of three parties, as the third party may be byzantine and simulate the rest of the ring. This will force parties adjacent in the ring to output the same value, which, in turn, guarantees that the two original executions lead to the same output.

Restricting the probabilistic space. In the following proof, we will consider a countable amount of different executions. Since Π satisfies probabilistic termination, it means that each of these executions are deciding almost surely. Because the countable union of almost surely events happens almost surely, this means that the event $E = \{$"all executions considered are deciding"$\}$ happens almost surely. So it is enough to prove that all canonical execution of A in E decide the same value to get the proof.

Constructing the ring. In order to formally describe the construction of the ring, we have to fix two executions with the same randomness. We denote the three parties by $\mathcal{P} = \{P_1, P_2, P_3\}$, and we consider two arbitrary maximal input configurations I_1, I_2. Let $\omega \in E$, and we consider a canonical execution $\varepsilon_1(\omega)$ with input configuration I_1, and a canonical execution $\varepsilon_2(\omega)$ with input configuration I_2. By definition of E, these two executions are deciding.

As $\varepsilon_1(\omega)$ is a deciding execution, there is a number of rounds $r_1(\omega) > 0$ such that, in $\varepsilon_1(\omega)$, all honest parties have decided the same value within $r_1(\omega)$ rounds. Similarly, there is an $r_2(\omega)$ such that, in the canonical execution $\varepsilon_2(\omega)$, all honest parties have decided the same value within $r_2(\omega)$ rounds. Let $r := \max\{r_1(\omega), r_2(\omega)\}$, r implicitly depends on ω.

To construct the ring depicted in Figure 6, we make $4(r+1)$ copies of each party P_i. Out of the $4(r+1)$ copies of party P_i, $2(r+1)$ will be the copies of P_i having its input value from I_1 and $2(r+1)$ will be the copies of P_i with input from I_2. The copies of P_i are then denoted by $P_{k,i,j}$, where $k \in \{1,2\}$, $i \in \{1,2,3\}$, and $j \in \{0,1,\ldots,2r\}$. The copies indexed by $k := 1$ are the ones on the top row of Figure 6, while the copies indexed by $k := 2$ are the ones on the bottom row. We now connect these copies via bidirectional communication channels:

- For $k \in \{1,2\}$ and $j \in \{0,1,\ldots,2r\}$, we add a channel between $P_{k,1,j}$ and $P_{k,2,j}$, and one between $P_{k,2,j}$ and $P_{k,3,j}$.
- Then, to complete the path on the row indicated by index $k := 1$, for every index $j \in \{0,1,\ldots,2r-1\}$, we add a channel between $P_{1,3,j}$ and $P_{1,1,j+1}$.
- Similarly, to complete the path on the row indicated by $k := 2$, for each $j \in \{0,1,\ldots,2r-1\}$, we add a channel between $P_{2,1,j}$ and $P_{2,3,j+1}$.
- We now connect the two rows and hence complete the ring: we add a channel between $P_{1,1,0}$ and $P_{2,3,0}$, and one between $P_{1,3,2r}$ and $P_{2,1,2r}$.

Outputs of adjacent copies. We consider a synchronous execution $\varepsilon(\omega)$ on the ring, where each copy $P_{k,i,j}$ runs A as party P_i, using the input value assigned to it in input configuration I_k. We assume that messages are received exactly Δ units of time after being sent. In the lemma below, we show that adjacent copies in the ring, denoted by Q_1 and Q_2, obtain the same output.

▶ **Lemma 18.** *Consider two parties Q_1 and Q_2 that are adjacent in the ring. Then, in execution $\varepsilon(\omega)$, Q_1 and Q_2 obtain outputs, and they output the same value.*

Proof. Without loss of generality, assume that Q_1 and Q_2 are copies of P_1 and P_2. We consider an execution $\varepsilon'(\omega)$ of A with three parties. In execution $\varepsilon'(\omega)$, P_1 and P_2 have the same input values as Q_1 and Q_2. P_3 is byzantine and simulates the additional parties in the ring, so they have the same behavior as in execution $\varepsilon(\omega)$. All messages are delivered in exactly Δ time, similarly to execution $\varepsilon(\omega)$.

We remark that execution $\varepsilon'(\omega)$ and execution $\varepsilon(\omega)$ are identical. Hence, our execution over the ring is equivalent to running A in a synchronous setting with three parties out of which one is corrupted. Therefore, A maintains its properties, namely probabilistic termination and agreement, on the ring as well. It follows that Q_1 and Q_2 both obtain outputs, and they output the same value. ◀

Outputs on the entire ring. Lemma 18 establishes that adjacent copies in the ring output the same value in execution $\varepsilon'(\omega)$. Using induction, we prove that all parties output the same value v in $\varepsilon(\omega)$. As a consequence, the copies $P_{1,i,r}$ and $P_{2,i,r}$ of each party P_i output v. This will imply that, in the two executions $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ we defined when constructing the ring, all honest parties output the same value.

▶ **Lemma 19.** *In execution $\varepsilon(\omega)$, all parties output, and they output the same value.*

Proof. We prove by induction that, in execution $\varepsilon(\omega)$, after $r' \leq r$ rounds, for $i \in \{1,2,3\}$, parties $P_{1,i,j}$ with $j \in \{r-(r-r'),\ldots,r+(r-r')\}$ are in the same state as party P_i in the canonical execution $\varepsilon_1(\omega)$. The base case $r' := 0$ considers the very beginning of the executions $\varepsilon(\omega)$ and $\varepsilon_1(\omega)$, where the parties have not yet received any messages. Their state then is only defined by their input value and ω. Each party $P_{1,i,0}$ takes its input from I_1, which the case for P_i in execution $\varepsilon_1(\omega)$ as well. We thus obtain that parties $P_{1,i,0}$ in the ring have the same input state as party P_i.

For the induction step, assume that our claim holds for $r' < r$, and we prove that it also holds for $r'+1$. Let $\mathcal{P}_{r'}$ denote the set of parties for which we proved they were in the same state as in $\varepsilon_1(\omega)$ after r' rounds, and let $\mathcal{P}_{r'+1}$ be the set of parties for which we want to prove that the claim holds after $r'+1$ rounds. The set of direct neighbors of $\mathcal{P}_{r'+1}$ (including themselves) in the ring is $\mathcal{P}_{r'}$. Moreover, within one round, parties are only able to receive messages from their direct neighbors. Using the induction hypothesis, we get that parties in $\mathcal{P}_{r'+1}$ receive exactly the same messages at the $(r'+1)$-th round in execution $\varepsilon_1(\omega)$ and $\varepsilon(\omega)$, therefore they stay in the same state after round $r'+1$.

As a consequence, P_i and $P_{1,i,r}$ are in the same state after the r-th round in executions $\varepsilon_1(\omega)$ and $\varepsilon(\omega)$ respectively. However, we have assumed that all parties in $\mathcal{P}$ output by round $r_1 \leq r$ in execution $\varepsilon_1(\omega)$. Therefore, in execution $\varepsilon(\omega)$, $P_{1,i,r}$ outputs the same value v_1 as P_i in $\varepsilon_1(\omega)$. With a symmetric argument, one can show that parties $P_{2,i,r}$ obtain the same output v_2 as P_2 in $\varepsilon_2(\omega)$. This enables us to conclude that $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ decide the same value $v_1 = v_2$ using Lemma 18. ◀

Assumption on deciding execution. Before concluding the proof of Lemma 9 using Lemma 19, we need to discuss the assumptions over deciding executions and make sure that event E happens almost surely. For every possible input configuration of size three, we consider a finite amount of fixed executions. Moreover, we take into account that there is only at most a countably infinite amount of input configurations (because we assumed V_{IN} was countable). Therefore, we consider in total a countable union of a finite amount of executions, which is countable. Therefore, the event that all these executions are deciding E happens almost surely, which concludes the proof of Lemma 9.

B.2 Proof of Theorem 10

We present the proof of Theorem 10, restated below.

▶ **Theorem 10.** *Assume $t_s > 0$ and consider a validity property* VAL*. If there is a t_s-secure* BA *protocol solving* VAL *in the synchronous model when no cryptographic setup is available and $n \leq 3 \cdot t_s$, then* VAL *is trivial.*

Proof. Assume $t_s \geq \lfloor n/3 \rfloor$, and that there is a t_s-secure BA protocol Π solving VAL in the synchronous model.

We consider two arbitrary maximal input configurations I_1 and I_2. We show that all canonical executions of I_1 and I_2 almost surely decide on the same output value. Then, Lemma 5 ensures that VAL is trivial. Similarly to the proof of Theorem 7 and Theorem 8, we use Π to build a protocol A for three parties, denoted by P_1, P_2, P_3, that matches the setting of Lemma 9. For protocol A, we consider the input space $\{0, 1\}$ and the output space V_{OUT}.

To do so, we partition $\mathcal{P}$ into three sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ of size at most t_s each. As shown in Figure 7, in protocol A, P_i simulates all the parties in set $\mathcal{P}_i$. P_i ensures that all messages between the simulated parties in $\mathcal{P}_i$ get delivered within Δ time. In addition, P_i forwards any message sent from a party it simulates to a party in set $\mathcal{P}_j$ $(j \neq i)$ to P_j. When P_j receives this message, it immediately forwards it to the simulated receiver. If P_i has input 0, the simulated parties in $\mathcal{P}_i$ take their input from I_1. Otherwise, they take their input from I_2. When a party in $\mathcal{P}_i$ outputs a value v, P_i outputs v.

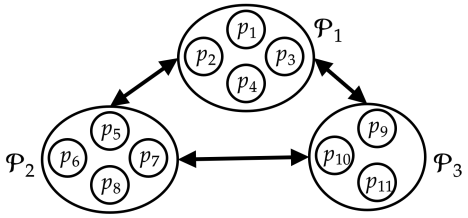

Figure 7 Partitioning $\mathcal{P}$ into 3 sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ with $n = 11$.

Since Π is a t_s-resilient BA protocol when $n \leq 3 \cdot t_s$, we obtain that A achieves probabilistic termination and agreement even when one of the three parties is byzantine. Then, Lemma 9 ensures that, almost surely, all deciding canonical executions of A lead to the same output value.

Moreover, we remark that, by construction, the canonical execution of A with input values $(0, 0, 0)$ matches the canonical execution of I_1 for Π. Similarly, the canonical execution of A with input $(1, 1, 1)$ matches the canonical execution of I_2. As a consequence, canonical executions of I_1 and I_2 decide the same value almost surely. Applying Lemma 5, we conclude that VAL is trivial. ◀

C Similarity Condition: Missing Proof

▶ **Lemma 13.** *If a validity property* VAL *is solvable in the network-agnostic model, then* VAL *satisfies the similarity condition.*

Proof. Assume that Π is a network-agnostic BA protocol solving VAL. Consider an input configuration I.

In the following proof, we will consider a finite amount of different executions. Since Π satisfies probabilistic termination, it means that each of these executions are deciding almost surely. Because the finite union of almost surely events happens almost surely, all of these executions will decide almost surely. So, there exists some $\omega \in \Omega$ which we fix such that all the executions below are deciding when ran with randomness ω.

We first consider the canonical execution $\varepsilon_1(\omega)$ of Π on I. As Π achieves network-agnostic BA, all honest parties output the same value v in execution $\varepsilon_1(\omega)$. We want to prove that $v \in \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J)$, hence we show that $v \in \text{VAL}(J)$ for every $J \in \text{SIMILAR}(I)$. Using the definition of $\text{SIMILAR}(I)$, we split the analysis as follows:

(i) $J \in \text{NEIGHBORS}(I)$ such that $J \subseteq I$. Consider an execution $\varepsilon_2(\omega)$ where the input configuration is J and the network is synchronous. Parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ are byzantine, but follow the protocol correctly using the values assigned to them in I as inputs. Parties in $\mathcal{P} \setminus \text{PARTIES}(I)$ crash at the start of the execution. Parties in $\text{PARTIES}(J)$ cannot distinguish between $\varepsilon_2(\omega)$ and $\varepsilon_1(\omega)$, so the output value v must also satisfy $v \in \text{VAL}(J)$.

(ii) $J \in \text{NEIGHBORS}(I)$ such that $|J| \geq n - t_a$. First, note that $I \cap J \neq \varnothing$: since $|I| \geq n - t_s$ and $|J| \geq n - t_a$, we obtain that $\left|I \cap J\right| \geq n - t_s - t_a$. As VAL is solvable, Theorem 8 ensures that $n > 2 \cdot t_s + t_a$, and allows us to conclude that $\left|I \cap J\right| > 0$.
Let $P \in I \cap J$. We consider an execution $\varepsilon_3(\omega)$ where the input configuration is J and the network is asynchronous. Similarly to execution $\varepsilon_2(\omega)$, parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ are byzantine, but follow the protocol correctly using the values assigned to them in I as inputs, and parties in $\mathcal{P} \setminus (\text{PARTIES}(I) \cup \text{PARTIES}(J))$ crash at the very beginning of the execution. All messages are delivered in a synchronous way, except for the messages sent from parties in $\text{PARTIES}(J) \setminus \text{PARTIES}(I)$. These are delayed until after party P outputs: this is possible as P cannot distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_3(\omega)$ and therefore it has to obtain an output without receiving these messages. In addition, the fact that P cannot distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_3(\omega)$ ensures that the output v agreed upon in $\varepsilon_1(\omega)$ satisfies $v \in \text{VAL}(J)$.

Therefore, the output v in execution $\varepsilon_1(\omega)$ satisfies $v \in \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J)$. So when running $\varepsilon_1(\varepsilon)$, if this execution terminates (which happens almost surely), then this value can be used for $\sigma(I)$.

We now explain how to get a Turing computable function out of protocol Π and this property. We first remark that Π is a randomized protocol, which can be simulated by a probabilistic Turing machine. However, when time complexity is not taken into account, a probabilistic Turing machine is as expressive as a regular deterministic Turing machine (see Theorem 2 from [24]: for a given number of random bits used, we can try all of them in exponential time). Therefore, we can simulate a deciding execution of ε_1 using a deterministic Turing machine. Because there are no byzantine parties, we can also remove the public-key infrastructure assumption which was used as a black box (for example, one can replace the signing function with one that returns the message concatenated with the id of the party). This gives us a Turing function to compute $\sigma(I)$. ◀