Optimal Synchronous Approximate Agreement with Asynchronous Fallback

DIANA GHINEA, Department of Electrical Engineering, ETH Zurich, Switzerland
CHEN-DA LIU-ZHANG, Department of Computer Science, Carnegie Mellon University, Pennsylvania
ROGER WATTENHOFER, Department of Electrical Engineering, ETH Zurich, Switzerland

Approximate Agreement (AA) allows a set of n parties that start with real-valued inputs to obtain values that are at most within a parameter $\varepsilon > 0$ from each other and within the range of their inputs. Existing AA protocols, both for the synchronous network model (where any message is delivered within a known delay Δ time) and the asynchronous network model, are secure when up to $t < n/3$ of the parties are corrupted and require no initial setup (such as a public-key infrastructure (PKI) for signatures).

We consider AA protocols where a PKI is available, and show the first AA protocol that achieves simultaneously security against t_s corruptions when the network is synchronous and t_a corruptions when the network is asynchronous, for any $0 \leq t_a < n/3 \leq t_s < n/2$ such that $t_a + 2 \cdot t_s < n$. We further show that our protocol is optimal by proving that achieving AA for $t_a + 2 \cdot t_s \geq n$ is impossible (even with setup). Remarkably, this is also the first AA protocol that tolerates more than $n/3$ corruptions in the synchronous network model.

1 INTRODUCTION

Byzantine Agreement (BA) is an extensively studied problem of distributed computing: it allows a set of n parties to achieve agreement on a common value even when up to t of the parties may be corrupted and arbitrarily deviate from the protocol.

Despite its importance, it is well known that the problem of BA incurs certain limitations, especially when it comes to deterministic solutions. In particular, the seminal result of Fischer, Lynch and Paterson [13] shows that BA cannot be achieved deterministically in an asynchronous network even under the presence of a single crash failure. Even when the network is synchronous, deterministic solutions require $t + 1$ rounds [10], where t is the corruption threshold. These limitations led to multiple variants of BA. One of the most natural and relevant variants is that of *Approximate Agreement* (AA), introduced by Dolev et al. [9]. In AA, the requirement of exact agreement is replaced by so-called ε-agreement: each party holds a real value as input, and the honest parties are required to output values that differ by at most a parameter ε. Moreover, one also requires that the outputs of honest parties lie within the range of inputs from honest parties. The problem of AA is of great interest in many practical applications, where outputting the exact same value is not needed and it is enough that the output values are sufficiently close. This is the case, for example, in applications for reading sensors or in protocols for clock synchronization [14, 16, 22].

The problem of AA has been widely studied in both the synchronous network model (where parties have access to synchronized clocks and messages sent by honest parties are guaranteed to be delivered within a known delay Δ), and the asynchronous network model (where the message delay can be arbitrary). In the synchronous model, Dolev et al. [9] showed that AA can be achieved deterministically up to $t < n/3$ corruptions, which is optimal when there is no initial setup. Moreover, current solutions [2, 9, 11] require less than $\log_2(\delta/\varepsilon)$ synchronous communication rounds to achieve ε-agreement, where δ denotes the initial size of the inputs range; that is, the inputs get exponentially close with the number of rounds. This is to be compared to deterministic BA protocols, which require at least $t + 1$ rounds. In the asynchronous model, Abraham et al. [1] showed that AA can be achieved deterministically against up to $t < n/3$ corruptions with no setup. It is not hard to see that this bound is optimal even for protocols assuming standard setup assumptions (such as a common random string, PKI, etc).

In this work, we consider the problem of AA with the standard setup of a PKI for signatures. We would like to achieve the highest level of corruption tolerance, while requiring as weaker network assumptions as possible. Concretely, we ask the following question:

Is there an Approximate Agreement protocol (assuming setup) that tolerates up to $t_s < n/2$ corruptions when the network is synchronous? If so, can it also tolerate $t_a < n/3$ corruptions when the network is asynchronous?

To the best of our knowledge, the problem of AA with setup has not been studied before, and all known synchronous protocols tolerate up to $t < n/3$ corruptions. Further note that current solutions for synchronous protocols heavily rely on a very stable network where every message is delivered within Δ time, and completely break down even when a single message is slightly delayed.

It is therefore desirable to construct a protocol that simultaneously achieves *best-of-both-worlds* guarantees: up to a high number t_s of corruptions when the network is synchronous, and a (possibly) lower number $t_a \leq t_s$ of corruptions even when the network is asynchronous. Indeed, this has been the subject of study of a recent line of works [3, 5, 8, 19] that studied the primitives of BA and multi-party computation.

Our Contributions. We completely characterize the feasibility of AA in the best-of-both-worlds setting, by presenting a protocol and a matching impossibility result.

- Feasibility result: Let $0 \leq t_a < n/3 \leq t_s < n/2$. We present an Approximate Agreement protocol that is secure against t_s corruptions in a synchronous network and t_a corruptions in an asynchronous network, as long as $2 \cdot t_s + t_a < n$, assuming a setup for digital signatures. By setting $t_a = 0$, this is also the first AA protocol that achieves up to $t_s < n/2$ corruptions in the purely synchronous model.
- Impossibility result: If $2 \cdot t_s + t_a \geq n$, there is no Approximate Agreement protocol secure against t_s corruptions in a synchronous network and t_a corruptions in an asynchronous network, even with setup.

1.1 Related Work

Best-of-both-worlds Protocols. Designing hybrid protocols that achieve security guarantees in both synchronous and asynchronous networks has been a subject that attracted increased attention in the recent years. Blum, Katz and Loss [3] showed a protocol which achieves BA resilient against $t_s < n/2$ corruptions in the synchronous model, and also resilient against $t_a < n/3$ corruptions in the asynchronous model when $t_a \leq t_s$ and $2 \cdot t_s + t_a < n$, which they prove to be optimal. In a follow-up work, the same authors extended the results to the setting of State-Machine Replication [4]. In a subsequent work, Blum, Liu-Zhang and Loss [5] showed a protocol for Multi-Party Computation (MPC), under the same provably optimal corruption thresholds.

The hybrid protocols of [3] and [5] require a high synchronous round complexity in comparison to the state-of-the-art purely synchronous protocols. In a recent work, Deligios, Hirt and Liu-Zhang [8] improved the round complexity (when the network is synchronous) for BA with the use of randomization techniques. And they also showed a protocol for MPC, whose round complexity is independent of the circuit to evaluate. In another recent work, Momose and Ren [19] introduced a refinement of hybrid protocols, where they also split the security guarantees (safety and liveness) for different thresholds, in line with the work of Kastrati, Hirt and Liu-Zhang [15].

Approximate Agreement. Approximate Agreement was introduced by Dolev et al. in [9]. The authors introduced a synchronous protocol resilient against $t < n/3$ corruptions with no setup, which is optimal when considering protocols with no setup, and an asynchronous variant resilient against $t < n/5$ corruptions. The asynchronous variant was later improved by Abraham et al. [1] to the optimal corruption threshold $t < n/3$.

The works of Fekete [11, 12] focused on obtaining asymptotically optimal convergence speed for AA protocols in multiple failure scenarios, in both synchronous and asynchronous settings. Ben-Or et al. [2] improve the convergence rate of [9] for $t < n/3$ using a mechanism of identifying corrupted parties based on Gradecast.

There are further works that extend the definition of AA. Independently, Mendes and Herlihy [18], and Vaidya and Garg [21] generalize the problem to multi-dimensional real inputs. The work by Nowak and Rybicki [20] defines the Abstract AA problem on a convex space and focuses on discrete settings where the input space is described as a graph.

1.2 Comparison to Previous Works

Comparison to Approximate Agreement Protocols. To the best of our knowledge, the problem of Approximate Agreement with setup has not been studied so far. All previous AA protocols operate in the plain model with no setup and are only secure up to $t < n/3$ corruptions. Our AA protocol assumes a public-key infrastructure and achieves a higher corruption threshold t_s when the network is synchronous, while at the same time tolerating t_a corruptions when the network is asynchronous, for thresholds satisfying $2 \cdot t_s + t_a < n$. In particular, our protocol achieves Approximate Agreement up to $t_s < n/2$ in the purely synchronous model.

In the following, we sketch the main challenges that appear in designing the hybrid protocol and our solution.
Tolerating $t_s < n/2$ corruptions when the network is synchronous. A key technique used in many previous synchronous protocols to ensure that the outputs of honest parties lie within the range of honest inputs is to let parties proceed in iterations where parties distribute their current value to all other parties. Each party obtains n values (possibly by adding some default value to replace any value that was not received) and then it discards the lowest and highest t_s values obtained. When $t_s < n/3$, it is clear that the $n - 2 \cdot t_s$ remaining values lie within the range of honest inputs. One can then take some sort of average of these values (which again lies in the honest range) and use it for the next iteration. One can show that this procedure ensures that the values from honest parties get closer at every iteration, given that the multisets of $n - 2 \cdot t_s$ values considered by different honest parties overlap. This technique does not work in the $t_s < n/2$ regime since $n - 2 \cdot t_s$ values can be as low as a single value, which may be a different value for every honest party. As a consequence, the values from honest parties may not converge.
Our Solution. In order to tolerate $t_s < n/2$ corruptions under a synchronous network, we first use the fact that discarding the highest and lowest t_s values is not always necessary. If $n - t_s + k$ values are received, it is guaranteed that at most k of these values are from corrupted parties (since the values not received are from corrupted parties), and therefore discarding the lowest and the highest k values is enough to remain in the range of the honest values. Secondly, to achieve convergence, we ensure that the multisets of values obtained by any two honest parties are **consistent** and have an overlap containing **all** the honest values by introducing a novel primitive, which we call *Overlap All-to-All Broadcast*.

In order to achieve security against t_a corruptions under an asynchronous network, our Overlap All-to-All Broadcast will also ensure that the values obtained by any two honest parties are consistent and have an overlap of at least $n - t_s$ values (possibly containing corrupted values). In this case, t_a of the values received by a party may be corrupted even if

that party only receives $n - t_s$ values. Since $n - t_s > 2 \cdot t_a$, discarding the lowest and highest t_a values is secure, similar to the previous case.

Of course, the parties do not know whether the network is synchronous or asynchronous, and therefore cannot decide whether to crop k or t_a values. However, we show that discarding the lowest and the highest $\max(t_a, k)$ values received is enough to simultaneously achieve security under both networks. This is achieved with the help of a technical lemma, which shows that the values obtained by any two honest parties after discarding the lowest and the highest values have some common range.

Comparison to Hybrid Protocols. Our techniques differ with current works on hybrid BA protocols in two essential aspects. First, the hybrid BA protocols make use of randomization in an essential way. In contrast, our protocols for AA are completely deterministic. Second, all hybrid protocols for BA, SMR and MPC follow the same approach: there is a black-box compiler that takes as input two protocols, one synchronous and one asynchronous with somewhat enhanced security guarantees, and outputs a hybrid protocol. Our protocol differs from this compiler-approach, and we provide a protocol completely from scratch.

2 MODEL AND DEFINITIONS

We consider a distributed system in which n parties $P_1, P_2, \ldots, P_n$ run a protocol over a network where all the parties are connected through pair-wise authenticated channels. The network can be synchronous or asynchronous, and the parties are not aware of the type of the network in which the protocol is running. If the network is synchronous, any message is delivered within a (publicly) known amount of time Δ and the parties have access to synchronized clocks. If the network is asynchronous, these assumptions are removed: parties do not have synchronized clocks, and the messages may be delayed arbitrarily.

We assume that all the parties have access to a public key infrastructure (PKI). That is, parties hold the same vector of public keys $(pk_1, pk_2, \ldots, pk_n)$, and each honest party P_i holds the secret key sk_i corresponding to pk_i.[1] A signature on a value v using secret key sk is computed as $\sigma \leftarrow \text{sign}_{sk}(v)$; a signature is verified relative to public key pk by calling $\text{ver}_{pk}(v, \sigma)$. For simplicity, we assume in our proofs that the signatures are perfectly unforgeable. When replacing the signatures with real-world instantiations, the results hold except with a negligible failure probability.

We consider an adaptive adversary that can corrupt at any point in the protocol's execution at most t_s parties if the network is synchronous, and at most t_a parties if the network is asynchronous, causing the corrupted parties to deviate arbitrarily. The adversary is strongly rushing: it can observe messages sent by the honest parties before choosing its own messages. Moreover, when an honest party sends a message, the adversary can immediately corrupt that party and replace the message with another of its choice. In addition, the adversary may schedule the delivery of the messages, with the condition that every message is delivered at some point and, if the network is synchronous, within Δ time.

2.1 Multisets

The Approximate Agreement protocol presented in this work makes use of multisets. In this section, we describe the notation. Let M and M' be multisets. We define the following:

- $m_M(v)$ denotes the multiplicity of the value v in the multiset M.
- $M \cup M'$ denotes the union of M and M': for any value v in M or M', $m_{M \cup M'}(v) = \max(m_M(v), m_{M'}(v))$.
- $M \cap M'$ denotes the intersection of M and M': for any value v in M or M', $m_{M \cap M'}(v) = \min(m_M(v), m_{M'}(v))$.

[1] Note that keys from corrupted parties can be chosen adversarially.

- $M \subseteq M'$ if for every value v in M or M', $m_M(v) \leq m_{M'}(v)$.
- $\left|M\right| = \sum_v m_M(v)$.
- $range(M) = [\min M, \max M]$.
- $length(M) = \max M - \min M$.
- $\{\{v_1, \ldots, v_k\}\}$ denotes the multiset containing the (possibly non-distinct) values $v_1, \ldots, v_k$.

2.2 Approximate Agreement

We include the definition of Approximate Agreement, as presented in [1, 9].

Definition 2.1. Let Π be a protocol where initially each party P holds a real-valued input and terminates upon generating an output v_P. We consider the following properties:

- t-Validity: The output value v_P of each honest party P is in the range of honest parties' inputs when at most t of the parties involved are corrupted.
- (t, ε)-Agreement: When at most t of the parties involved are corrupted, for every two honest parties P and P', it eventually holds that $\left|v_P - v'_P\right| \leq \varepsilon$.

Then, Π is a t-secure Approximate Agreement protocol if it achieves t-Validity and (t, ε)-Agreement for any $\varepsilon > 0$.

2.3 Consistency Primitives

We present two primitives which will be essential in the design of our hybrid Approximate Agreement protocol.

2.3.1 Overlap All-to-All Broadcast. This primitive can be seen as some variant of all-to-all Reliable Broadcast tailored to the hybrid setting. Each party initially holds an input and wants to distribute it to the other parties. In the end, each party outputs a set of value-sender pairs.

Definition 2.2. Let Π be a protocol where each party P holds an input v_P and every party may output a set of value-sender pairs O_P upon termination. We consider the following properties:

- t-Termination: If there are at most t corrupted parties involved, every honest party eventually terminates.
- t-Synchronized Termination: Assume that the network is synchronous, there are at most t corrupted parties involved, and all the honest parties start executing Π at the same time. Then, all the honest parties terminate and obtain an output at the same time.
- t-Consistency: Assume that there are at most t corrupted parties involved. If two honest parties P and P' terminate, and $(v, P'') \in O_P$ and $(v', P'') \in O_{P'}$, then $v = v'$.
- (T, t)-Overlap: Assume that there are at most $t \leq T$ corrupted parties involved. If two honest parties P and P' terminate, then $\left|O_P \cap O_{P'}\right| \geq n - T$.
- t-Synchronized Overlap: Assume that the network is synchronous, there are at most t corrupted parties involved and that the honest parties start executing Π at the same time. Then, if an honest party P terminates, then $(v_{P'}, P') \in O_P$ for every honest party P'.

We say that Π is a (t_s, t_a)-secure Overlap All-to-All Broadcast protocol if it achieves:

- t_s-Synchronized Termination, t_s-Consistency, and t_s-Synchronized Overlap under a synchronous network;
- t_a-Termination, t_a-Consistency and (t_s, t_a)-Overlap under an asynchronous network.

2.3.2 Reliable Broadcast. We recall the definition of Reliable Broadcast. We make explicit the concrete running time and simultaneous termination properties, since they will be required when the network is synchronous.

Definition 2.3. Let Π be a protocol where a designated party S (called the sender) holds a value v_S, and every party P may output a value v_P upon termination. We consider the following properties:

- t-Validity: If S is honest and at most t parties are corrupted, then every honest party P eventually outputs $v_P = v_S$.
- (t, c)-Synchronized Validity: Assume that S is honest, at most t parties are corrupted, the network is synchronous, and the honest parties start the execution of the protocol at the same time. Then every honest party P outputs $v_P = v_S$ within $c \cdot \Delta$ time.
- t-Consistency: When at most t of the parties involved are corrupted, if there exists an honest party P that outputs a value v_P, then every honest party P' eventually outputs $v_{P'} = v_P$.
- (t, c)-Synchronized Consistency: Assume that at most t of the parties involved are corrupted, the network is synchronous, and the honest parties start the execution of the protocol at the same time. If the first honest party that terminates obtains output v, then every honest party outputs the same value v within $c \cdot \Delta$ time.

We say that Π is a (t_s, t_a, c_1, c_2)-secure Reliable Broadcast protocol if it achieves:

- (t_s, c_1)-Synchronized Validity and (t_s, c_2)-Synchronized Consistency under a synchronous network;
- t_a-Validity and t_a-Consistency under an asynchronous network.

3 MAIN PROTOCOL

Our Approximate Agreement protocol proceeds in iterations consisting of two steps. In the first step of each iteration `it`, each party distributes their current value – in the first iteration, this is their input value, and afterwards, the value obtained in the previous iteration. The current values are distributed via an *Overlap All-to-All Broadcast* protocol Π_{oBC} (see Definition 2.2 and Section 4 for a construction). After executing Π_{oBC}, each party P obtains a multiset of values V, which contains $n - t_s + k$ values for some $0 \leq k \leq t_s$.

In the second step, each party P computes its new value as follows: P first discards the lowest and the highest $\max(t_a, k)$ values from the multiset V and obtains a multiset denoted by T. Then, it obtains its new value by computing the average between the lowest and the highest value in T.

Intuitively, if the network is synchronous, Π_{oBC} guarantees: 1) that all honest parties remain synchronized: they obtain the output simultaneously, which in turn implies that they also start each iteration at the same time; and 2) the multiset V from each honest party contains all the values sent by honest parties (at least $n - t_s$ values), and at most $k \leq t_s$ values from corrupted parties. In this case, discarding the highest and lowest $\max(t_a, k)$ values ensures security. Validity is achieved because V contains $n - t_s + k$ values, so discarding the highest and lowest k values leaves values within the honest range of values. And the same goes for discarding t_a values (note that since $n - t_s > 2 \cdot t_a$, there is always at least a value remaining). Moreover, with the help of a technical lemma (see Section 3.2), we show that the multisets of values T and T' obtained by any two honest parties P and P' after discarding the lowest and the highest values they received have some common range, which we will show is enough for the output values to converge. In fact, we will show that the range of values obtained by the honest parties is halved in every iteration.

If the network is asynchronous, Π_{oBC} guarantees that for any two honest parties P and P', their output multisets V and V' overlap in at least $n - t_s$ values. From this overlap, it is possible that t_a of the values are from corrupted parties. This implies that discarding the highest and lowest t_a values, say, from multiset V, trivially leaves a multiset T

containing values within the range of honest values. Further note that discarding more values, i.e. $k \geq t_a$ values, leaves a multiset T of size $n - t_s - k \geq n - 2 \cdot t_s > t_a$, which is also within the honest range. Convergence of the asynchronous case will follow similarly to the synchronous case, with the help of the technical lemma.

We formally describe our protocol below. In the description, we assume that an upper bound $\delta_{\max}$ on the length of the input space of the honest parties is known. In practice, this assumption is reasonable since $\delta_{\max}$ could be derived from the data types used to store the values.

For simplicity, in the protocol description, we omit the identification numbers attached to the messages sent over the network. The identification ensures that parties can identify which messages correspond to which protocol instances.

Protocol Π_{AA}

Code for party P with input v

$S := \lceil \log_2(\delta_{\max}/\varepsilon) \rceil$

1: **for** $\mathtt{it}$ from 1 to S **do**:
2: Join Π_{oBC} with input $(v, \mathtt{it})$ and let O denote the set obtained
3: $V := \{\{v' \mid ((v', \mathtt{it}), P') \in O\}\}$
4: $k := |V| - (n - t_s)$
5: $T :=$ multiset obtained by discarding the lowest and highest $\max(t_a, k)$ values from V
6: $v := (\min T + \max T)/2$
7: **end for**
8: Output v and terminate

The following theorem is proven in the next subsections:

Theorem 3.1. *Let n, t_s, t_a such that $0 \leq t_a < n/3 \leq t_s < n/2$ and $2 \cdot t_s + t_a < n$. Then, Π_{AA} is an n-party protocol that satisfies the following properties:*

(1) When Π_{AA} is run in a synchronous network, it is a t_s-secure Approximate Agreement protocol.
(2) When Π_{AA} is run in an asynchronous network, it is a t_a-secure Approximate Agreement protocol.

3.1 Analysis

In our analysis, we use $U_{\mathtt{it}}$ to denote the multiset of values v computed by the honest parties in iteration $\mathtt{it} \geq 1$, and U_0 to denote the multiset containing the initial inputs of the honest parties.

3.1.1 Synchronous analysis. We first consider the case that the network is synchronous, and show that Π_{AA} is a t_s-secure Approximate Agreement protocol.

Lemma 3.2. *When run in a synchronous network, Π_{AA} achieves t_s-Validity.*

Proof. We show that, for every iteration $\mathtt{it} \geq 1$, $range(U_{\mathtt{it}}) \subseteq range(U_{\mathtt{it}-1})$: Π_{oBC} guarantees that, for every honest party P, the multiset V obtained in iteration $\mathtt{it}$ contains $n - t_s$ honest values and $k \geq 0$ values that may be sent by corrupted parties and may be outside $range(U_{\mathtt{it}-1})$. The multiset T is constructed by discarding the highest and the lowest $\max(k, t_a)$ values from V. Note that T is not empty, since T has size at least $\min(n - t_s - 2 \cdot t_a, n - 2 \cdot t_s) \geq 1$. Therefore $range(T) \subseteq range(U_{\mathtt{it}-1})$, and hence the value $v = (\min T + \max T)/2$ obtained by P in iteration $\mathtt{it}$ is in $range(U_{\mathtt{it}-1})$. As this result holds for any honest party P, it follows that $range(U_{\mathtt{it}}) \subseteq range(U_{\mathtt{it}-1})$. We obtain that $U_S \subseteq U_0$, and therefore t_s-Validity is achieved. □

Lemma 3.3. *When run in a synchronous network, Π_{AA} achieves (t_s, ε)-Agreement.*

Proof. We show that, in every iteration $\texttt{it} \geq 1$, the range values obtained by the honest parties is halved: $length(U_{\texttt{it}}) \leq length(U_{\texttt{it}-1})/2$.

Let P and P' denote two honest parties and let V and V' denote the multisets they obtain via Π_{oBC} in iteration $\texttt{it}$. Let v and v' denote the values obtained by P and P' at the end of iteration $\texttt{it}$. As Π_{oBC} achieves t_s-Consistency and t_s-Synchronized Overlap, $\left|V \cup V'\right| \leq n$ and $\left|V \cap V'\right| \geq n - t_s$. By applying Lemma 3.6, we obtain that $\left|v - v'\right| \leq length(T \cup T')/2$, where T and T' are the multisets obtained by P and P' by discarding the lowest and the highest values from V and V' respectively. We have shown in the proof of Lemma 3.2 that $range(T), range(T') \subseteq range(U_{\texttt{it}-1})$. Then, we obtain that $range(T \cup T') \subseteq range(U_{\texttt{it}-1})$ and hence $\left|v - v'\right| \leq length(T \cup T)/2 \leq length(U_{\texttt{it}-1})/2$.

Since this result holds for any pair of honest parties P and P', it follows that $length(U_{\texttt{it}}) \leq length(U_{\texttt{it}-1})/2$ for every iteration $\texttt{it} \geq 1$. This implies that $length(U_S) \leq length(U_0)/2^S$ and hence Π_{AA} achieves (t_s, ε)-Agreement within $S = \lceil \log_2 \frac{length(U_0)}{\varepsilon} \rceil$ iterations. □

3.1.2 Asynchronous analysis. We now consider the case that the network is asynchronous, and show that Π_{AA} is a t_a-secure Approximate Agreement protocol.

Lemma 3.4. *When run in an asynchronous network, Π_{AA} achieves t_a-Validity.*

Proof. Similarly to the proof of Lemma 3.2, we show that for every $\texttt{it} \geq 1$, $range(U_{\texttt{it}}) \subseteq range(U_{\texttt{it}-1})$. For any honest party P, Π_{oBC} guarantees that P obtains a multiset V that contains $n - t_s + k$ values with $k \geq 0$. Out of these values, at most t_a are sent by corrupted parties, and hence at most t_a values are outside $range(U_{\texttt{it}-1})$. Then, P constructs the multiset T by discarding at least the lowest and the highest t_a values from V. Since T is non-empty, as T has size at least $\min(n - t_s - 2 \cdot t_a, n - 2 \cdot t_s) \geq 1$, we have that $range(T) \subseteq range(U_{\texttt{it}-1})$. Therefore, the value $v = (\min T + \max T)/2$ obtained by P at the end of iteration $\texttt{it}$ is in $range(U_{\texttt{it}-1})$, and hence $range(U_{\texttt{it}}) \subseteq range(U_{\texttt{it}-1})$. It follows that $U_S \subseteq U_0$, and therefore t_a-Validity is achieved. □

Lemma 3.5. *When run in an asynchronous network, Π_{AA} achieves (t_a, ε)-Agreement.*

Proof. The proof is similar to the proof of Lemma 3.3. We show that in every iteration $\texttt{it} \geq 1$ the range of the honest values is halved: $length(U_{\texttt{it}}) \leq length(U_{\texttt{it}-1})/2$. Let P and P denote two arbitrary honest parties. Let V and V' denote the multisets of values obtained by P and P' in iteration $\texttt{it}$ via Π_{oBC}, and let v and v' denote the values they compute at the end of iteration $\texttt{it}$. Π_{oBC} guarantees that $\left|V \cap V'\right| \geq n - t_s$ and $\left|V \cup V'\right| \leq n$, hence we can apply Lemma 3.6 and we obtain that $\left|v - v'\right| \leq length(T \cup T')/2$, where T and T' denote the multisets obtained by P and P' after discarding the lowest and the highest received values. We have shown in the proof of Lemma 3.4 that $range(T), range(T') \subseteq range(U_{\texttt{it}-1})$. Then, we obtain that $range(T \cup T') \subseteq range(U_{\texttt{it}-1})$ and hence $\left|v - v'\right| \leq length(T \cup T)/2 \leq length(U_{\texttt{it}-1})/2$.

Since this result holds for any pair of honest parties P and P', it follows that $length(U_{\texttt{it}}) \leq length(U_{\texttt{it}-1})/2$ for every iteration $\texttt{it} \geq 1$. This implies that $length(U_S) \leq length(U_0)/2^S$ and hence, Π_{AA} achieves (t_a, ε)-agreement within $S = \lceil \log_2 \frac{length(U_0)}{\varepsilon} \rceil$ iterations. □

3.2 Technical Combinatorial Lemma

We provide a technical result that allows us to show that the values obtained by the honest parties get closer in each iteration.

Lemma 3.6. *Let n, t_a, t_s and such that $0 \leq t_a, c \leq t_s$ and $2 \cdot t_s + t_a < n$. Let V and V' denote two multisets such that:*

- $\left|V\right| = n - t_s + k$ *and* $\left|V'\right| = n - t_s + k'$*, where* $0 \leq k, k' \leq t_s$*;*
- $\left|V \cup V'\right| \leq n$*;*
- $\left|V \cap V'\right| \geq n - t_s$*.*

We construct the multisets T *and* T' *by discarding the lowest and the highest* $\max(k, t_a)$ *values from* V *and respectively by discarding the lowest and the highest* $\max(k', t_a)$ *values from* V'*. Let* $v = \frac{\min T + \max T}{2}$ *and* $v' = \frac{\min T' + \max T'}{2}$*. Then,* $\left|v - v'\right| \leq \frac{1}{2} \cdot length(T \cup T')$*.*

In order to prove this result, we arrange the values in $V \cup V'$, V, and V' in arrays, and we use the lemma below. We use the notation $A[i]$ to refer the i-th value of the array A, where $i \geq 1$, and $range(A)$ to denote the interval $[a, b]$, where a and b are the lowest and respectively highest values in the array A.

Lemma 3.7. *Let* n*,* t_a*,* t_s *and* c *such that* $0 \leq t_a, c \leq t_s$ *and* $2 \cdot t_s + t_a < n$*. Let* A *denote an array of* $n - c$ *(possibly non-distinct) real values ordered increasingly. Let* k_1, k_2 *such that* $0 \leq k_1, k_2 \leq t_s - c$ *and* $(t_s - c - k_1) + (t_s - c - k_2) \leq (t_s - c)$*, and let* A_1 *and* A_2 *denote two arrays of* $n - c$ *values obtained from* A *by replacing* $t_s - c - k_1$ *and respectively* $t_s - c - k_2$ *values with* $\perp$*. We construct two arrays* T_1 *and* T_2 *by discarding the lowest and the highest* $\max(k_1, t_a)$ *non-*$\perp$ *values from* A_1*, respectively the lowest and the highest* $\max(k_2, t_a)$ *non-*$\perp$ *values from* A_2*. Then,* $range(T_1) \cap range(T_2) \neq \emptyset$*.*

The figure below shows an example of the arrays described in Lemma 3.7.

A :	1	2	3	4	5	6	7	8	9	10	11
A_1 :	1	2	3	4	$\perp$	6	7	8	9	10	11
A_2 :	1	2	3	$\perp$	5	$\perp$	$\perp$	$\perp$	9	10	11
T_1 :	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	6	7	$\perp$	$\perp$	$\perp$	$\perp$
T_2 :	$\perp$	2	3	$\perp$	5	$\perp$	$\perp$	$\perp$	9	10	$\perp$

Fig. 1. Let $n = 11$, $t_s = 5$, and $t_a = 0$. The figure shows an example of arrays A, A_1 and A_2 in Lemma 3.7. Here, $k_1 = 4$ and $k_2 = 1$. Then, T_1 is obtained by discarding the lowest and the highest 4 non-$\perp$ values din A_1, and T_2 is obtained by discarding the lowest and the highest non-$\perp$ value din A_2. Note that, while T_1 and T_2 do not have any values in common, $range(T_1) \cap range(T_2) \neq \emptyset$.

Proof. Let $i_{1,\min}$ and $i_{1,\max}$ denote the minimum, resp. maximum, index i such that $T_1[i] = A[i] \neq \perp$. Similarly, let $i_{2,\min}$ and $i_{2,\max}$ denote the minimum, resp. maximum, index i such that $T_2[i] = A[i] \neq \perp$. (In the example shown in Figure 1, we obtain the following values: $i_{1,\min} = 6$, $i_{1,\max} = 7$, $i_{2,\min} = 2$, and $i_{2,\max} = 10$.)

Since the values in A, and hence the values in T_1 and T_2, are ordered increasingly, $range(T_1) \cap range(T_2) = \emptyset$ implies that either $i_{1,\max} < i_{2,\min}$ or $i_{1,\min} > i_{2,\max}$. We only prove that $i_{1,\max} \geq i_{2,\min}$, as showing that $i_{1,\min} \leq i_{2,\max}$ is analogous.

We first obtain a lower bound for $i_{1,\max}$. Since there are $n - t_s + k_1$ non-$\perp$ values in A_1 and the largest $\max(t_a, k_1)$ values are discarded when constructing T_1, we obtain that $i_{1,\max} \geq n - t_s + k_1 - \max(t_a, k_1)$.

We now obtain an upper bound for $i_{2,\min}$. Since the number of $\perp$ values in A_2 is $t_s - c - k_2$ and the lowest $\max(t_a, k_2)$ values are discarded when constructing T_2, we obtain that $i_{2,\min} \leq t_s - c - k_2 + \max(t_a, k_2) + 1$.

To prove that $i_{2,\min} \leq i_{1,\max}$, we compare these bounds and show that:

$$i_{2,\min} \leq t_s - c - k_2 + \max(t_a, k_2) + 1 \leq n - t_s + k_1 - \max(t_a, k_1) \leq i_{1,\max}.$$

It is therefore enough to prove that the following inequality holds for any valid choice of k_1 and k_2:

$$t_s - c - k_2 + \max(t_a, k_2) < n - t_s + k_1 - \max(t_a, k_1).$$

If $k_1, k_2 \leq t_a$, we obtain the following inequality:

$$t_s - c - k_2 + t_a < n - t_s + k_1 - t_a \iff 2 \cdot t_a + (t_s - k_1) + (t_s - c - k_2) < n.$$

In this case, we use the fact that $(t_s - c - k_1) + (t_s - c - k_2) \leq t_s - c$, and then it is enough to show that $2 \cdot t_a + t_s < n$. This follows immediately from the fact that $2 \cdot t_s + t_a < n$ and $t_a \leq t_s$.

In each of the remaining cases, the result follows from the fact that $2 \cdot t_s + t_a < n$:

- if $k_1 \leq t_a$ and $k_2 > t_a$, we obtain the following:

$$t_s - k_2 + k_2 - c = t_s - c < n - t_s + k_1 - t_a \iff 2 \cdot t_s + t_a - k_1 - c < n.$$

- if $k_1 > t_a$ and $k_2 \leq t_a$, we obtain the following:

$$t_s - k_2 + t_a - c < n - t_s + k_1 - k_1 = n - t_s \iff 2 \cdot t_s + t_a - k_2 - c < n.$$

- if $k_1, k_2 > t_a$, we obtain the following:

$$t_s - k_2 + k_2 - c = t_s - c < n - t_s + k_1 - k_1 = n - t_s \iff 2 \cdot t_s - c < n.$$

Therefore, $range(T_1) \cap range(T_2) \neq \emptyset$. □

We now prove Lemma 3.6 using Lemma 3.7.

Proof of Lemma 3.6. We arrange the (possibly non-distinct) values in $V \cup V'$ ordered increasingly in an array A. As $\left|V \cup V'\right| \leq n$ and $\left|V \cap V'\right| \geq n - t_s$, we obtain that A contains $n - c$ values, where $0 \leq c \leq t_s$. Note that $k, k' \leq t_s - c$.

We now build the two arrays A_1 and A_2 representing V and respectively V'. Firstly, we set every value in A_1 to $\perp$. Afterwards, for every unique value v with multiplicity $m_V(v)$ in V, let i_v denote the index of the first occurrence of v in A. We set $A_1[i] := v = A[i]$ for every index i such that $i_v \leq i \leq i_v + m_V(v) - 1$. In order to apply Lemma 3.7, the number of $\perp$ values in A_1 must be $t_s - c - k_1$, where $0 \leq k_1 \leq t_s - c$. As V contains $n - t_s + k$ values from $V \cup V'$, and hence from the array A, the obtained array A_1 contains $n - t_s + k$ non-$\perp$ values and $(n - c) - (n - t_s + k) = t_s - c - k$ values that are $\perp$. Therefore, the condition is satisfied for $k_1 = k$. We construct the array A_2 identically from V'. Using a similar argument, we obtain that the number of $\perp$ values in A_2 is $t_s - c - k_2$, where $k_2 = k' \leq t_s - c$.

The last condition required by Lemma 3.7 is that $(t_s - c - k_1) + (t_s - c - k_2) \leq (t_s - c)$. Because of the way we constructed the arrays A_1 and A_2, and since $\left|V \cap V'\right| \geq n - t_s$, there are at least $n - t_s$ indices i such that $A_1[i] = A_2[i] = A[i]$, hence at most $t_s - c$ indices i in which at least one of $A_1[i] = \perp$ and $A_2[i] = \perp$ holds. In addition, since every v occurs $\max(m_V(v), m'_V(v))$ times in A, there is no index i such that $A_1[i] = A_2[i] = \perp$. This means that the set of $(t_s - c - k_1)$ indices i such that $A_1[i] = \perp$ and the set of $(t_s - c - k_2)$ indices i such that $A_2[i] = \perp$ are disjoint. Therefore, the condition $(t_s - c - k_1) + (t_s - c - k_2) \leq (t_s - c)$ holds.

We can now apply Lemma 3.7. The arrays T_1 and T_2 constructed as described in Lemma 3.7 contain the same real values as the multisets T and T' described in our hypothesis. Therefore, $range(T) \cap range(T') \neq \emptyset$. We assume without loss of generality that $v \geq v'$. Then, $\max T \geq \min T'$, and, since $range(T) \cap range(T') \neq \emptyset$, $\min T \leq \max T'$. We obtain

that:

$$\begin{aligned}\left|v - v'\right| = v - v' &= \frac{1}{2} \cdot (\max T + \min T) - \frac{1}{2} \cdot (\max T' + \min T') \\ &= \frac{1}{2} \cdot (\max T - \min T') - \frac{1}{2} \cdot (\max T' - \min T) \\ &\leq \frac{1}{2} \cdot (\max T - \min T') \leq \frac{1}{2} \cdot \mathit{length}(T \cup T').\end{aligned}$$

□

4 OVERLAP ALL-TO-ALL BROADCAST

In order to achieve Overlap All-to-All Broadcast and to ensure that the honest parties output enough common values so that the values they obtain converge, we employ the witness technique. This technique is introduced in [1] for obtaining asynchronous Approximate Agreement resilient against $t < n/3$ parties. The idea is that the parties share their values using classical Reliable Broadcast and report each value received using channels that maintain the FIFO property – First In, First Out: namely, the messages are delivered in the order in which they are sent. A party decides that it can stop waiting for values when it has received via Reliable Broadcast all the values reported by $n - t$ parties named *witnesses*. Each witness must report at least $n - t$ values. Here, the guarantee that every two honest parties have $n - t$ common values comes from the fact that every two honest parties P and P' share an honest witness P''. This is because every two honest parties have at least $n - 2 \cdot t > t + 1$ common witnesses, so there is an honest witness whose first $n - t$ values reported via the FIFO channels are received by both parties.

Adapting the witness technique to our hybrid network setting poses several challenges.

First, the quorum argument breaks when there are more than $n/3$ corruptions, even if the network is synchronous. Second, an additional subtle point is that in order to make use of this primitive in the AA protocol and overcome the $n/3$ bound, we will need to compose this protocol throughout sequential iterations. We achieve this by guaranteeing that all parties have simultaneous termination when the network is synchronous.[2] More concretely, if the first value reported by an honest party P is sent by a corrupted party, the other honest parties would have to receive this value as well via Reliable Broadcast, or they would never consider P a witness. If the sender of a Reliable Broadcast invocation is corrupted, honest parties may obtain outputs in different communication rounds, which breaks simultaneous termination.

To tolerate $t_s < n/2$ corruptions in the synchronous model, we devise a mechanism that ensures that all honest parties become witnesses of every party, and that the values sent by honest parties are taken into account. This mechanism will also ensure simultaneous termination. This is achieved by observing that if the sender of a Reliable Broadcast invocation is honest, then we can guarantee that the honest parties obtain outputs within a known constant number of communication rounds r. Additionally, if the sender is corrupted and an honest party outputs at time τ, then all the honest parties output after at most Δ time, at time $\tau + \Delta$.

In addition, our protocols will retain security if the network is asynchronous.

4.1 Reliable Broadcast

Our Reliable Broadcast protocol is based on the protocol by Momose and Ren in [19], adapted to the hybrid network setting. In a nutshell, the idea is that at each step of the protocol, the parties wait for at least Δ time. When the network is synchronous, this ensures that 1) for an honest sender, all parties simultaneously obtain output after a fixed number

[2]Composing protocols with probabilistic termination is known to pose several challenges. See [6, 7, 17] for a nice discussion.

of rounds, and 2) for a corrupted sender, the parties output at times that differ in at most Δ time. Moreover, when the network is asynchronous, security is retained.

Initially, each party marks the time when it starts executing the protocol in $\tau_{\texttt{start}}$. At the same time, the sender sends its signed value to all the parties.

If the network is synchronous, any message is delivered within Δ time. Hence, the honest parties wait until time $\tau_{\texttt{start}} + \Delta$ to ensure that, if the sender is honest, every honest party has received the sender's message before taking any further step. Then, at time $\tau_{\texttt{start}} + \Delta$, the honest parties forward this signed value to all the parties. Hence, by time $\tau_{\texttt{start}} + 2 \cdot \Delta$, any inconsistent messages sent by a dishonest sender in the first step are observed. Detecting such consistencies is the key in tolerating a higher number of corruptions.

Then, if the sender is honest, each honest party sends a signed `vote` message at time $\tau_{\texttt{start}} + 2 \cdot \Delta$, meaning that each honest party should always expect $n - t_s$ `vote` messages by time at least $\tau_{\texttt{start}} + 3 \cdot \Delta$ in order to make a decision and output a value. On the other hand, if the sender is corrupted, a party P that receives $n - t_s$ `vote` messages cannot be certain that every honest party has received enough `vote` messages as well, hence P forwards the signed votes to all the honest parties. If this is the case, the signed votes are received after at most Δ time by every honest party. A party can safely terminate as soon as it receives and forwards the $n - t_s$ signed votes, hence, if the sender is honest, at time $\tau_{\texttt{start}} + 3 \cdot \Delta$.

Note that, if the sender is corrupted, the honest parties may output much later than time $\tau_{\texttt{start}} + 3 \cdot \Delta$. In this case, the only guarantee is that once the first honest party outputs, every honest party outputs the same value within Δ time.

If the network is asynchronous, once an honest party outputs a value, every honest party is guaranteed to eventually receive $n - t_s$ `vote` messages and hence eventually outputs. Inconsistent messages from the sender may not be detected, which lowers the resilience threshold in comparison to the synchronous setting.

Protocol Π_{rBC}

Code for sender S on input v

1: Send $(\texttt{propose}, v, \text{sign}_{sk_S}(v))$ to all the parties

Code for party P

1: $\tau_{\texttt{start}} \coloneqq \tau_{\texttt{now}}$
2: Upon receiving the first valid $(\texttt{propose}, v, \text{sign}_{sk_S}(v))$ from S and when $\tau_{\texttt{now}} \geq \tau_{\texttt{start}} + \Delta$:
3: Forward $(\texttt{propose}, v, \text{sign}_{sk_S}(v))$ to all the parties
4: When $\tau_{\texttt{now}} \geq \tau_{\texttt{start}} + 2 \cdot \Delta$, if no valid $(\texttt{propose}, v', \text{sign}_{sk_S}(v'))$ for $v' \neq v$ was received:
5: Send $(\texttt{vote}, v, \text{sign}_{sk_P}(v))$ to all the parties
6: Upon receiving a set $C(v)$ of $n - t_s$ valid `vote` messages for the same value v, and when $\tau_{\texttt{now}} \geq \tau_{\texttt{start}} + 3 \cdot \Delta$:
7: Send $C(v)$ to all the parties, output v and terminate

Our proof for the following result uses a similar analysis to that of Momose and Ren [19] and is included in the appendix.

Theorem 4.1. *Π_{rBC} is an n-party protocol that achieves $(t_s, t_a, 3, 1)$-secure Reliable Broadcast for $t_a \leq t_s$ such that $2 \cdot t_s + t_a < n$.*

4.2 Overlap All-to-All Broadcast protocol

In our Overlap All-to-All Broadcast protocol, the parties distribute their input value via Π_{rBC}. The honest parties wait until they have collected $n - t_s$ values and $3 \cdot \Delta$ time has passed. Until both of these conditions are satisfied, they send a `report` message for every value they receive.

If the network is synchronous, Π_{rBC} guarantees that any value sent by an honest party is received within $3 \cdot \Delta$ time. In addition, if an honest party receives a value from a corrupted party via Π_{rBC} by time $3 \cdot \Delta$ and reports it, then every honest party receives the `report` message and the same corrupted value by time $4 \cdot \Delta$. Since the honest parties do not report any new values after time $3 \cdot \Delta$, it is ensured that every honest party becomes a witness for all the honest parties by time $4 \cdot \Delta$. So every honest party gathers enough witnesses to terminate at time $4 \cdot \Delta$ and obtains a set O containing the value sent by each honest party.

Note that waiting Δ time at each step does not violate the security of the protocol when the network is asynchronous: every honest party terminates and the outputs of any two honest parties contain $n - t_s$ overlapping values.

We can now present the description of our Overlap All-to-All Broadcast protocol. In the description, we make use of FIFO channels. Such channels can be simulated on top of regular channels: a party attaches a sequential number to each message it sends and only considers a message received when the messages with previous sequential numbers from the same party are received as well.

Protocol Π_{oBC}

Code for party P with input v

1: $\tau_{\mathtt{start}} := \tau_{\mathtt{now}}$
2: $O := \emptyset$; $R_{P'} := \emptyset$ for every party P'
3: Invoke Π_{rBC} with input $(\mathtt{value}, v)$ and, for every party P', join the invocation of Π_{rBC} having P' as sender
4: **repeat**
5: Upon obtaining $(\mathtt{value}, v)$ via Π_{rBC} with sender P':
6: FIFO-send $(\mathtt{report}, v, P')$ and add (v, P') to O
7: Upon FIFO-receiving $(\mathtt{report}, v, P')$ from P'': Add (v', P') to $R_{P''}$
8: **until** $\tau_{\mathtt{now}} > \tau_{\mathtt{start}} + 3 \cdot \Delta$ and $|O| \geq n - t_s$
9: **repeat**
10: Upon obtaining $(\mathtt{value}, v)$ via Π_{rBC} with sender P': add (v, P') to O
11: Upon FIFO-receiving $(\mathtt{report}, v, P')$ from P'': Add (v', P') to $R_{P''}$
12: $W := \{P' \mid R_{P'} \subseteq O \text{ and } |R_{P'}| \geq n - t_s\}$
13: **until** $\tau_{\mathtt{now}} > \tau_{\mathtt{start}} + 4 \cdot \Delta$ and $|W| \geq n - t_s$
14: Output O and terminate

The next result follows from the lemmas proven in the subsections below.

Theorem 4.2. *Π_{oBC} is an n-party protocol that achieves (t_s, t_a)-secure Overlap All-to-All Broadcast protocol for any $t_s < n/2$ and $t_a < n/3$ such that $t_a \leq t_s$ and $2 \cdot t_s + t_a < n$.*

4.2.1 Synchronous analysis. We first assume that the network is synchronous, and we show that Π_{oBC} satisfies t_s-Synchronized Termination, t_s-Synchronized Overlap, and t_s-Consistency.

Lemma 4.3. *Π_{oBC} satisfies t_s-Synchronized Termination and t_s-Synchronized Overlap: assume that t_s of the parties involved are corrupted. If the honest parties start executing Π_{oBC} at the same time τ, then they terminate at the same time. In addition, for every honest party P, the output O contains $n - t_s$ values sent by honest parties.*

Proof. Π_{rBC} guarantees that every honest party receives at least all the values sent by honest parties within $3 \cdot \Delta$ time. Therefore, at time $\tau + 3 \cdot \Delta$, the set O obtained by each honest party contains $n - t_s$ values sent by honest parties. Hence, the honest parties exit the first loop and enter the second loop at the same time $\tau + 3 \cdot \Delta$.

In order to show that the honest parties exit the second loop at the same time as well, it is enough to show that every honest party gathers $n - t_s$ witnesses by time $\tau + 4 \cdot \Delta$.

Let P and P' denote two honest parties, and let W' denote the set of witnesses obtained by P' in line 12. We show that, by time $\tau + 4 \cdot \Delta$, it holds that $P \in W'$. P sent `report` messages by time $\tau + 3 \cdot \Delta$ for $n - t_s$ honest values, and possibly for some corrupted values, which P received by time $\tau + 3 \cdot \Delta$. Afterwards, it stopped sending `report` messages. Then, Π_{rBC} guarantees that P' has received each value reported by P by time $\tau + 4 \cdot \Delta$. In addition, P' has received the `report` messages sent by P by time $\tau + 4 \cdot \Delta$, and therefore has added P to its set of witnesses W' by time $\tau + 4 \cdot \Delta$.

As this result holds for any pair of honest parties, it follows that every honest party obtains at least $n - t_s$ witnesses by time $\tau + 4 \cdot \Delta$, and hence exits the second loop and terminates at time $\tau + 4 \cdot \Delta$. □

The next result trivially follows from t_s-Synchronized Consistency of Π_{rBC}.

Lemma 4.4. *Π_{oBC} satisfies t_s-Consistency: assume that t_s of the parties involved are corrupted. Let P and P' denote two arbitrary honest parties, and let O and O' denote their outputs Π_{oBC}. Then, if $(v, P'') \in O$, then, $(v', P'') \notin O'$.*

4.2.2 Asynchronous analysis. We now assume that the network is asynchronous, and we show that Π_{oBC} satisfies t_a-Termination and (t_s, t_a)-Overlap.

Lemma 4.5. *Π_{oBC} satisfies t_a-Termination: if at most $t_a \leq t_s$ of the parties involved are corrupted and $2 \cdot t_s + t_a < n$, every honest party eventually terminates the execution of Π_{oBC}.*

Proof. Let P and P' denote two honest parties and assume that P' has not yet terminated. P eventually sends `report` messages for at least the first $n - t_s$ values it receives through Π_{rBC}. Then, since Π_{rBC} guarantees that every value received by P is eventually received by P' as well, P' eventually adds P to its set W'. As this result holds for every honest party P, the set W' eventually contains $n - t_s$ parties and hence P' eventually terminates the execution of Π_{oBC}. □

Lemma 4.6. *Π_{oBC} satisfies (t_s, t_a)-Overlap: let P and P' denote two arbitrary honest parties, and let O and O' denote their outputs in Π_{oBC}. If at most $t_a \leq t_s$ of the parties involved are corrupted and $2 \cdot t_s + t_a < n$, then it holds that $|O \cap O'| \geq n - t_s$.*

Proof. Let W and W' denote the sets of witnesses obtained by P and P'. Note that each of W and W' contains at least $n - t_s$ parties, hence at least $n - t_s - t_a$ honest parties. As $2 \cdot t_s + t_a < n$, we obtain that $2 \cdot (n - t_s - t_a) > n - t_a$, and hence $W \cap W'$ contains at least one honest party P''. Then, since the `report` messages are sent through channels that have the FIFO property and the values reported are received via Π_{rBC}, both O and O' contain at least the first $n - t_s$ value-sender pairs reported by P'', which concludes the proof. □

The next result trivially follows from t_a-Consistency of Π_{rBC}.

Lemma 4.7. *Π_{oBC} satisfies t_a-Consistency: assume that at most t_a of the parties involved are corrupted. Let P and P' denote two arbitrary honest parties, and let O and O' denote their outputs in Π_{oBC}. If $(v, P'') \in O$ and $(v', P'') \in O'$, then $v = v'$.*

5 IMPOSSIBILITY RESULT

In this section, we show that there is no Approximate Agreement protocol that is t_s-secure under a synchronous network, and t_a-secure under an asynchronous network, for $2 \cdot t_s + t_a \geq n$. This shows that our protocol Π_{AA} achieves the optimal corruption threshold.

Theorem 5.1. *There is no n-party Approximate Agreement protocol that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network when $2 \cdot t_s + t_a \geq n$.*

Proof. Let Π denote any Approximate Agreement n-party protocol and let $n = 2 \cdot t_s + t_a$. Assume that Π achieves t_s-Validity in a synchronous setting and (t_a, ε)-Agreement in an asynchronous setting.

We fix an arbitrary $\varepsilon > 0$ and we partition the n parties into three sets S_a, $S_{-\varepsilon}$, and $S_{+\varepsilon}$ such that $|S_a| = t_a$ and $|S_{-\varepsilon}| = |S_{+\varepsilon}| = t_s$. The parties in $S_{-\varepsilon}$ have input $-\varepsilon$, while the parties in $S_{+\varepsilon}$ have input $+\varepsilon$.

We consider the following scenarios:

Scenario 1: The protocol runs in a synchronous network. The parties in $S_{+\varepsilon}$ and S_a are honest, and the parties in S_a have input $+\varepsilon$. The t_s parties in $S_{-\varepsilon}$ are corrupted and do not participate in the protocol. Then, as Π achieves t_s-Validity in the synchronous setting, the honest parties in $S_{+\varepsilon}$ and S_a output $+\varepsilon$.

Scenario 2: The protocol runs in a synchronous network. The parties in $S_{-\varepsilon}$ and S_a are honest, and the parties in S_a have input $-\varepsilon$. The t_s parties in $S_{+\varepsilon}$ are corrupted and do not participate in the protocol. Then, as Π achieves t_s-Validity in the synchronous setting, the honest parties in $S_{-\varepsilon}$ and S_a output $-\varepsilon$.

Scenario 3: The protocol runs in an asynchronous network. The parties in $S_{-\varepsilon}$ and $S_{+\varepsilon}$ are honest, while the parties in S_a are corrupted. The scheduler blocks the communication between $S_{-\varepsilon}$ and $S_{+\varepsilon}$ (any message sent between the two sets is delayed until all the honest parties obtain outputs). Additionally, the scheduler ensures that any message sent within $S_a \cup S_{-\varepsilon}$ or $S_a \cup S_{+\varepsilon}$ arrives immediately.

We make a virtual copy of each party in S_a and we obtain two virtual sets of corrupted parties: $S_a^{+\varepsilon}$ and $S_a^{-\varepsilon}$. The virtual copies in $S_a^{+\varepsilon}$ run Π correctly with input $+\varepsilon$ towards the parties in $S_{+\varepsilon}$, and the virtual copies in $S_a^{-\varepsilon}$ run Π correctly with input $-\varepsilon$ towards the parties in $S_{-\varepsilon}$.

The view of the honest parties in $S_{+\varepsilon}$ is identical to their view in Scenario 1, therefore they output $+\varepsilon$. Similarly, the view of the honest parties in $S_{-\varepsilon}$ is identical to their view in Scenario 2, and therefore they output $-\varepsilon$. This contradicts that Π achieves (t_a, ε)-Agreement if the network is asynchronous. □

REFERENCES

[1] Ittai Abraham, Yonatan Amit, and Danny Dolev. 2005. Optimal Resilience Asynchronous Approximate Agreement. In *Principles of Distributed Systems*, Teruo Higashino (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 229–239.

[2] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. 2010. Brief Announcement: Simple Gradecast Based Algorithms. In *Distributed Computing*, Nancy A. Lynch and Alexander A. Shvartsman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 194–197.

[3] Erica Blum, Jonathan Katz, and Julian Loss. 2019. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography Conference*. Springer, 131–150.

[4] Erica Blum, Jonathan Katz, and Julian Loss. 2021. Tardigrade: An Atomic Broadcast Protocol for Arbitrary Network Conditions. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 547–572.

[5] Erica Blum, Chen-Da Liu-Zhang, and Julian Loss. 2020. Always Have a Backup Plan: Fully Secure Synchronous MPC with Asynchronous Fallback. In *Advances in Cryptology – CRYPTO 2020*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer International Publishing, Cham, 707–731.

[6] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. 2016. Probabilistic Termination and Composability of Cryptographic Protocols. In *CRYPTO 2016, Part III (LNCS, Vol. 9816)*, Matthew Robshaw and Jonathan Katz (Eds.). Springer, Heidelberg, 240–269. https://doi.org/10.1007/978-3-662-53015-3_9

[7] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. 2017. Round-Preserving Parallel Composition of Probabilistic-Termination Cryptographic Protocols. In *ICALP 2017 (LIPIcs, Vol. 80)*, Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl (Eds.). Schloss Dagstuhl, 37:1–37:15. https://doi.org/10.4230/LIPIcs.ICALP.2017.37

[8] Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. 2021. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In *Theory of Cryptography Conference*. Springer, 623–653.

[9] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. 1986. Reaching Approximate Agreement in the Presence of Faults. *J. ACM* 33, 3 (May 1986), 499–516. https://doi.org/10.1145/5925.5931

[10] Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.

[11] Alan David Fekete. 1986. Asymptotically Optimal Algorithms For Approximale Agreement. In *5th ACM PODC*, Joseph Y. Halpern (Ed.). ACM, 73–87. https://doi.org/10.1145/10590.10597

[12] Alan David Fekete. 1987. Asynchronous Approximate Agreement. In *6th ACM PODC*, Fred B. Schneider (Ed.). ACM, 64–76. https://doi.org/10.1145/41840.41846

[13] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[14] Joseph Y. Halpern, Barbara Simons, H. Raymond Strong, and Danny Dolev. 1984. Fault-Tolerant Clock Synchronization. In *3rd ACM PODC*, Robert L. Probert, Nancy A. Lynch, and Nicola Santoro (Eds.). ACM, 89–102. https://doi.org/10.1145/800222.806739

[15] Martin Hirt, Ard Kastrati, and Chen-Da Liu-Zhang. 2021. Multi-Threshold Asynchronous Reliable Broadcast and Consensus. In *24th International Conference on Principles of Distributed Systems*. 1.

[16] Leslie Lamport and P. M. Melliar-Smith. 1985. Synchronizing Clocks in the Presence of Faults. *J. ACM* 32, 1 (Jan. 1985), 52–78. https://doi.org/10.1145/2455.2457

[17] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. 2006. On the Composition of Authenticated Byzantine Agreement. *J. ACM* 53, 6 (nov 2006), 881–917. https://doi.org/10.1145/1217856.1217857

[18] Hammurabi Mendes and Maurice Herlihy. 2013. Multidimensional approximate agreement in Byzantine asynchronous systems. In *45th ACM STOC*, Dan Boneh, Tim Roughgarden, and Joan Feigenbaum (Eds.). ACM Press, 391–400. https://doi.org/10.1145/2488608.2488657

[19] Atsuki Momose and Ling Ren. 2021. Multi-Threshold Byzantine Fault Tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 1686–1699. https://doi.org/10.1145/3460120.3484554

[20] Thomas Nowak and Joel Rybicki. 2019. Byzantine Approximate Agreement on Graphs. In *33rd International Symposium on Distributed Computing (DISC 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 146)*, Jukka Suomela (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 29:1–29:17. https://doi.org/10.4230/LIPIcs.DISC.2019.29

[21] Nitin H. Vaidya and Vijay K. Garg. 2013. Byzantine vector consensus in complete graphs. In *32nd ACM PODC*, Panagiota Fatourou and Gadi Taubenfeld (Eds.). ACM, 65–73. https://doi.org/10.1145/2484239.2484256

[22] Jennifer Lundelius Welch and Nancy Lynch. 1988. A new fault-tolerant algorithm for clock synchronization. *Information and Computation* 77, 1 (1988), 1–36. https://doi.org/10.1016/0890-5401(88)90043-0

6 APPENDIX

6.1 Analysis of Π_{rBC}

The lemmas in the subsections below prove that, if $t_a \leq t_s$ such that $2 \cdot t_s + t_a < n$, Π_{rBC} achieves $(t_s, 3)$-Synchronized Validity and $(t_s, 1)$-Synchronized Consistency when it runs in a synchronous setting, and t_a-Validity and t_a-Consistency when it runs in an asynchronous setting. The next result follows immediately:

Theorem 4.1. *Π_{rBC} is an n-party protocol that achieves $(t_s, t_a, 3, 1)$-secure Reliable Broadcast for $t_a \leq t_s$ such that $2 \cdot t_s + t_a < n$.*

6.1.1 Synchronous analysis. We first assume that the network is synchronous, and we show that Π_{rBC} satisfies $(t_s, 3)$-Synchronized Validity and $(t_s, 1)$-Synchronized Consistency.

Lemma 6.1. *Π_{rBC} satisfies $(t_s, 3)$-Synchronized Validity: assume that at most t_s of the parties involved are corrupted, the sender is honest and has input v. Then, if the honest parties start executing Π_{rBC} at the same time τ, they terminate with output v at the same time $\tau' = \tau + 3 \cdot \Delta$.*

Proof. Note that using unforgeable signatures guarantees that no party can forge signatures for messages on $v' \neq v$ on behalf of the honest sender. Hence, no honest party can output $v' \neq v$.

As the sender is honest, it sends $(\texttt{propose}, v, \mathsf{sign}_{sk_S}(v))$ at time τ, meaning that every honest party receives this message by time $\tau + \Delta$. Since no honest party can receive a valid `propose` message for a different value, every honest party sends a `vote` message for v at time $\tau + 2 \cdot \Delta$. These messages are received by time $\tau + 3 \cdot \Delta$. Then, at time $\tau + 3 \cdot \Delta$, every honest party has received the set $C(v)$ containing at least $n - t_s$ valid `vote` message for v, sends $C(v)$ to every party, and, since it cannot receive a set $C(v')$ for $v' \neq v$ (as this would require `vote` messages from $n - t_s > t_s$ different parties), it terminates with output v. □

Lemma 6.2. *Assume that at most t_s of the parties involved are corrupted. If the honest parties start executing the protocol and the same time and two certificates $C(v)$ and $C(v')$ are formed during the execution of the protocol, then $v = v'$.*

Proof. We assume that $v \neq v'$. Then, since $n - t_s > t_s$, honest parties have sent `vote` messages for both v and v'. Let P denote the first honest party that sent a `vote` message for v, at time τ. Similarly, let P' denote the first honest party that sent a `vote` message for v' at time τ'. We assume without loss of generality that $\tau \leq \tau'$. Then, as P sent a `vote` message for v at time τ, P has received and forwarded a valid `propose` message from S for v at time $\tau - \Delta$, which P' has received at time $\tau \leq \tau'$. We obtain a contradiction since it follows that P' could not have voted for v'. □

Lemma 6.3. *Π_{rBC} satisfies $(t_s, 1)$-Synchronized Consistency: assume that at most t_s of the parties involved are corrupted and that the honest parties start executing the protocol at the same time. If an honest party outputs v at time τ, then every honest party outputs v by time $\tau + \Delta$.*

Proof. Let P denote the first honest party that obtains output v, at time τ. Lemma 6.2 shows that no honest parties can receive a valid set $C(v')$ for $v' \neq v$ and hence implies that no honest party can output $v' \neq v$. It remains to show that all the other honest parties can output v by time $\tau + \Delta$.

Let P' denote an honest party who has not yet terminated by time τ. P has received a valid set $C(v)$ at time τ and sent it to all the parties. Then, P' has received $C(v)$ at time $\tau + \Delta$ the latest, and hence forwards $C(v)$, outputs v and terminates by time $\tau + \Delta$. □

6.1.2 Asynchronous analysis. We now assume that the network is asynchronous, and we show that Π_{rBC} satisfies t_a-Validity and t_a-Consistency.

Lemma 6.4. *Π_{rBC} achieves t_a-Validity.*

Proof. We assume that the sender is honest and has input v. Since using unforgeable signatures guarantees that no party can forge signatures for messages on $v' \neq v$ on behalf of the honest sender, no honest party can output $v' \neq v$. It remains to show that every honest party can output v. Every party eventually receives $(\texttt{propose}, v, \mathsf{sign}_{sk_S}(v))$ from the sender, and no party can receive a valid `propose` message for a different value. Therefore, at least every honest party sends a `vote` message for v, and hence every party eventually receives a set $C(v)$ and no set $C(v')$ for $v' \neq v$ (as this would require `vote` messages from $n - t_s > t_a$ different parties). It follows that every honest party eventually outputs v and terminates. □

Lemma 6.5. *If at most t_a of the parties involved are corrupted and two certificates $C(v)$ and $C(v')$ are formed during the execution of the protocol, then $v = v'$.*

Proof. Assume that $v \neq v'$. Since $C(v)$ and $C(v')$ contain `vote` messages from $n - t_s$ parties each, there are $2 \cdot (n - t_s) - n > t_a$ parties, hence at least one honest party, that sent a `vote` message for both v and v', which contradicts the steps of the protocol. □

Lemma 6.6. *Π_{rBC} achieves t_a-Consistency.*

Proof. Assume that a party P terminates with output v. Lemma 6.5 shows that no honest party can obtain a valid set $C(v')$ for $v' \neq v$, and hence no honest party can terminate with a different output. In addition, P forwards the valid set $C(v)$ to all the parties. Hence all the honest parties eventually receive $C(v)$ and, since they cannot receive a valid set $C(v')$ for v', they eventually output v and terminate. □