Network agnostic consensus in constant time

Simon Holmgaard Kamp¹, Julian Loss¹, and Jesper Buus Nielsen²

 $^1\,$ CISPA Helmholtz Center for Information Security $^2\,$ Aarhus University

Abstract. Network agnostic protocols (Blum, Katz, Loss TCC '19) are consensus or MPC protocols that strike a balance between purely synchronous and asynchronous protocols. Given thresholds t_a, t_s that satisfy $t_a < n/3 < t_s < n/2$ and $2t_s + t_a < n$, they have the unique property of remaining secure against an adversary that either (1) corrupts up to t_s parties in a synchronous execution where all messages are delivered within a known bound Δ or (2) corrupts up to t_a in an asynchronous execution where messages can be delayed arbitrarily. All existing network agnostic protocols follow a design pattern which first attempts to run a synchronous path, and then switches to an asynchronous path as a fallback option if the synchronous path times out after some time T due to the network being asynchronous path might take an unusually long time even when the network is synchronous. As a result, for various basic tasks including Byzantine Agreement or MPC, no existing network agnostic protocol is able to terminate for all honest parties within constant expected time in *all possible executions*. In this work, we introduce a new paradigm to construct network agnostic consensus that, for the first time, overcome this barrier. Using this new design pattern we first present simple protocols for reliable

time, overcome this barrier. Using this new design pattern we first present simple protocols for reliable broadcast (RB) and binary agreement (BA) that are *responsive* when no more than t_a parties are corrupted and run in expected constant time regardless of the network conditions.³ We then extend our results to asynchronous common subset (ACS) and MPC. Notably, our approach *reverses the order* of the synchronous and asynchronous path by designing protocols that are first and foremost asynchronous and only fall back to the synchronous execution path when more than t_a parties are corrupted.

³ The latter was already possible for RB, which by design requires only constantly many constant rounds, but is not guaranteed to terminate for all parties.

Table of Contents

1	Introduction	
	1.1 Our Contribution: A New Perspective on Network Agnostic Protocols	
	1.2 Technical overview	
	1.3 Related Work	
2	Preliminaries 10	
	2.1 Network, Time and Corruption 10	
	2.2 Problem definitions 13	
3	Reliable Broadcast 18	
4	Binary Agreement 17	
	4.1 Validated Graded Agreement 18	
	4.2 Validated Binary Agreement	
	4.3 Reducing BA to VBA 22	
5	Agreement on a Core Set $\dots 2^{2}$	
	5.1 Network Agnostic ACS	
	5.2 On the Number of Honest Inputs in the Core Set	
А	Constant Time Honest Majority Asynchronous ACS from RB and CC	
	A.1 Causal Cast	
	A.2 Causal Cast Complexity	
	A.3 Justified Gather	
	A.4 Justified Graded Gather	
	A.5 Justified Graded Block Selection	
	A.6 Justified Block Selection	
	A.7 Agreement on a Core Set 40	

1 Introduction

Distributed protocols such as consensus or multi-party computation protocols allow n parties connected over a pair-wise network to compute a common function of their inputs. An important property for such protocols is to remain secure and live even in the presence of some number of malicious faults t < n that may deviate arbitrarily from the protocol. A long line of research has established conditions under which various distributed protocols can be implemented. They can roughly be divided into to classes: (1) The first class of protocols assumes a synchronous network that delivers sent messages within some fixed duration Δ and parties share a common notion of time. In this model, it is possible to tolerate up to t < n/2 corrupted parties for many important distributed tasks such as MPC and consensus (assuming some cryptographic setup). (2) The second class of protocols is designed to run in an asynchronous network that guarantees nothing beyond the eventual delivery of sent messages. As a result, MPC and consensus can be solved in this model if and only if t < n/3. This poses a dilemma for a protocol designer: what type of network should a protocol be designed for to be as secure as possible under *all* possible conditions? To soften the blow of this trade-off Blum et al. [BKL19] (BKL19) introduced what has since been coined the *network agnostic (NA)* setting. In their work, they study the fundamental problem of binary Byzantine agreement, where parties wish to agree on a single bit. They show a protocol Π with two corruption thresholds t_a and t_s satisfying $0 < t_a < n/3 \le t_s < n/2$ and $2t_s + t_a < n$ that remains secure against a malicious adversary that corrupts up to t_a parties if the network is asynchronous and corrupts up to t_s when the network is synchronous. Moreover, they show that $n > t_a + 2t_s$ is a necessary condition in this setting to solve BA.

The BKL19 Paradigm and Its Limitations. To construct a protocol Π with these properties, BKL19 first constructs a synchronous protocol Π_s that is secure against t_s corruptions and is guaranteed to terminate after some fixed duration T if the network remains synchronous. Their second protocol Π_a is asynchronous with a corruption tolerance of t_a , but has a crucial additional property: it remains secure against t_s many corruptions in a scenario where all honest parties supply the same input to Π_a . Leveraging these properties, BKL19 show that one can run Π_s and Π_a back to back, where the input to Π_a is the output of Π_s . If the network is synchronous, Π_s will always terminate with all honest parties in consensus, and thus the overall security of the protocol is t_s . If the network is asynchronous, parties simply let Π_s time out and then run Π_a on their initial input if Π_s did not output. Since the initial work of BKL19, many followup works have shown that NA solutions exist for various tasks such as general MPC [BZL20], distributed key generation [BCLZL23,ACC22], state-machine replication [BKL21,ABKL22], and other variants of consensus [DE24,CGWW24,GLZW22]. While these protocols have also made substantial progress in terms of efficiency, they all follow the above design pattern to some degree. This results in a severe penalty to the round efficiency of all existing NA protocols. Namely, the timeout parameter T of the synchronous component Π_s must always be super-constant such that non-termination of Π_s by time T implies that the network is in the asynchronous mode. This issue was partially addressed in the work of Deligios, Hirt, and Liu-Zhang [DHLZ21] and the recent work of Deligios and Erbes [DE24]. These works construct NA protocols for MPC and consensus, respectively, that achieve optimal corruption parameters and run in O(1) expected many synchronous rounds whenever the network is synchronous. However, there is currently no NA protocol (either for consensus or MPC) that runs in O(1) expected time even when the network is asynchronous. In this work, we present a novel design pattern for NA protocols, which, for the first time, overcomes this barrier. Leveraging our approach, we show optimally resilient protocols for various important consensus tasks such as byzantine agreement, asynchronous common subset, reliable broadcast, and (assuming fully homomorphic encryption) MPC. In addition to always running in O(1) expected time, all of our protocols are optimistically responsive, in the sense that if t_a or fewer parties are corrupted then the running time only depends on the actual network delay δ .

1.1 Our Contribution: A New Perspective on Network Agnostic Protocols

As briefly explained above, network agnostic protocols have so far followed the blueprint laid out by Blum, Katz and Loss in [BKL19] where a synchronous protocol is followed by an asynchronous protocol only after the network is demonstrated to be asynchronous. ⁴ First, the parties try to reach consensus through a synchronous path via a protocol that (when the network is synchronous) has a worst case running time T after which each honest party will either have received an output or be able to conclude that the network was not synchronous. Thus honest parties that reach this point without having received output can conclude that the network must be asynchronous and switch to an alternative asynchronous consensus mechanism. While this design pattern has proven very fruitful, it has the major drawback that the synchronous protocol must run for T time before switching to asynchrony. Otherwise the adversary can break safety when the network is asynchronous.

From Protocols to Rounds via Network Agnostic Quorums. Our solution to overcome this challenge is to move the logical switch between the two paths from the protocol to the round level. In every round of the protocol, parties first attempt to obtain sufficiently many messages to trigger the next step on the asynchronous path. If this fails, it must mean that the asynchronous path did not have enough support from $n-t_a$ honest parties and thus, the execution must be a synchronous one with $t_s > t_a$ corrupted parties. Thus, they instead wait for the round to conclude, at which point the worst-case delivery guarantees of the network will ensure that they receive a sufficient number of messages to move on to the next instruction on the synchronous path. This idea has several interesting implications. First, it inverts the order in which paths are attempted when compared to the BKL19 paradigm. Second, it allows for fluidity between paths, meaning that the protocol is never really locked into one of the paths at any given point of the protocol execution. Lastly, it is has the distinct advantage of our protocols being inherently *responsive*, meaning that they run in time proportional to the at the actual of the network delay δ (as opposed to the worst-case delay Δ) in synchronous executions where the number of corrupted parties is low enough.

To hint at how this works: consider that a party who takes a step down the asynchronous path can wait until unanimous votes from $n - t_a$ parties are received. This can be thought of as a *network agnostic quorum*, in the sense that if $n - t_a$ unanimous votes are received by one party, then the same cannot happen for a conflicting vote for some other party. Since this other party received votes from some other $n - t_a$ parties, the two sets have an intersection of at least $n - 2t_a > t_s$ parties, which includes at least one honest party, regardless of the network condition.

Achieving Liveness on Both Paths. By following a pattern that only relies on such network agnostic quorums, it is fairly straight-forward to construct protocols that are safe and life in asynchrony and remain safe against t_s corruptions in synchrony using standard techniques from asynchronous consensus. However, in synchronous executions we need to make progress against $t_s > t_a$ corruptions and cannot wait for $n - t_a$ votes.

As already hinted above, our design facilitates this via a synchronous path which is used as a fallback option when parties fail to collect enough votes to continue moving along the asynchronous path after waiting for a timeout dependent on Δ . On the synchronous path,

⁴ A few notable exceptions include protocols by Momose and Ren [MR21] and the compiler by Deligios and Erbes [DE24]. In both cases the logic that allows asynchronous output is always executed. But this does not change the fact that T time must pass before giving asynchronous output.

we will generally enforce that parties do not send any votes towards the completion of a step on the synchronous path before they have waited long enough. here, long enough means that if the network is synchronous, they have received all votes that honest parties might have sent towards the completion of the corresponding step on the asynchronous path. The goal is to use the votes from the synchronous path to establish a different type of quorum based on only $n - t_s$ unanimous votes, which allows our protocols to be live in synchrony. In addition, by allowing outputs based on quorums from both synchronous and asynchronous paths they are also live in asynchrony. But with two paths to output, we need to consider some implications for safety.

Achieving Safety on Both Paths. If parties follow the asynchronous path and give output based on the network agnostic quorum, then they received $n - t_a$ unanimous votes which as discussed implies uniqueness even in synchrony. In addition to this, we enforce that an honest party will not send conflicting votes on the two paths. Hence, consistency between the two different quorums types also holds in all network conditions because $n - t_a - t_s > t_s$ implies that the two quorums include a vote from some common honest party. This is enough to establish safety of outputs from the asynchronous path, regardless the network. Thus, it remains to consider outputs from the synchronous path, i.e., based on $n - t_s$ synchronous votes.

For this, we need to instead consider safety in each of the two possible network settings separately. First, safety on the synchronous path can come from the network in fact being asynchronous and thus at most $t_a < t_s$ parties being corrupted. Namely, this implies that any two sets of size (at least) $n-t_s$ have an intersection of at least $n-2t_s > t_a$ parties, one of which is honest. Alternatively, safety of the decisions made using the synchronous path (when the network is synchronous and up to t_s parties can be corrupted), must, as hinted above, be based on messages being delivered within the known delay Δ . But it might not immediately be clear how to enforce that parties can reliably wait to hear all honest contributions towards the synchronous path before contributing to the asynchronous path. Thus, valid concern is that since some parties can make decisions responsively on the asynchronous path, they might drift too far apart from parties who follow the synchronous path.

So, in order to have safety in synchronous executions, we will use a design pattern which enforces that honest parties will never be out of sync (in terms of when they enter the same asynchronous step) by more than δ . Moreover, the synchronous path is only attempted after waiting for 2Δ time after first being able to send the type of message we are waiting for. Together, these imply that when an honest party moves down the synchronous path, then we are in one of two cases:

- either the network is synchronous and all honest messages that are sent as part of the asynchronous path will be seen by all other honest parties before they potentially use the synchronous path,
- or alternatively the network is asynchronous but the number of corruptions is so low that any two sets of messages from $n - t_s$ parties contains a message from a common honest party.

We outline this design pattern below.

Composing Synchronous Protocols without Synchronized Rounds. If we want to achieve an optimistically responsive and network agnostic protocol it is necessary to make use of asynchronous quorums without waiting for timeouts. So, if one is not careful it is possible that one party follows a fast asynchronous path while another waits for the timeout, resulting in desynchronization. To prevent this scenario the protocols we present will never rely on the parties being synchronized in a lockstep fashion, but only on being able to wait until all honest parties have responded to a request.

Concretely they will follow a common pattern where messages, inputs, and outputs come with a *justifier*. A justifier corresponds to a justifier predicate J that allows checking whether a message satisfies certain properties. Usually these properties guarantee that the message is something an honest party could have sent if it saw a certain set of messages. In our case, we additionally want the justifiers to have the property that a party who receives a justified message which was prompted by some activation rule of the protocol, will be "caught up" and itself be able to trigger that same activation rule and send its own justified message (if it did not previously do so). With these properties in place, a party can trust that when it triggers an activation rule, then it will hear the message caused by that activation rule from all honest parties within 2 network delays.

A concrete example of a justifier with this property is used by Cachin et al. [CKPS01] to reduce the problem of solving binary agreement to solving the (in many settings) easier problem of validated binary agreement. While binary BA imposes the validity requirement that the output must be the input of an honest party, validated BA simply requires the output to satisfy some predicate. So to solve binary BA, parties can initially sign and send their binary input and gather identical messages from t+1 parties, which eventually happens when n > 3t. This set of signatures can be used to justify the input as having been the initial input of an honest party to other parties, who in turn can forward the same set to justify proposing the same value.⁵

1.2 Technical overview

To demonstrate the technique we start out by constructing simple consensus primitives in the form of network agnostic (expected) constant time reliable broadcast and binary agreement protocols that are responsive against up to t_a corruptions. Then we show how to build asynchronous expected constant time ACS with n > 2t from black box use of reliable broadcast and a common coin. When this ACS is instantiated with our network agnostic RB, it inherits network agnostic security and optimistic responsiveness.

Reliable Broadcast. Our first contribution is a network agnostic protocol for reliable broadcast as defined by Bracha [Bra87]. Reliable broadcast allows a designated sender to send a message to a set of parties with the following crucial property, usually referred to as *totality*: *if* an honest party outputs the sender's message, then all honest parties eventually output the same message. This primitive was studied in the network agnostic setting by Momose

⁵ These certificates were coined *justifiers* by Cachin et al. [CKPS01]. In line with [DYMM⁺20,KNTT22,LN24] we apply justifiers more generally to protocol messages, inputs and outputs.

and Ren [MR21] and Ghinea et al. [GLZW22], who give constant time solutions following the BKL19 paradigm.⁶ With our technique we also achieve optimistic responsiveness.

In our reliable broadcast, the sender first signs a message and sends it to all parties, who in response sends a signed asynchronous vote for the message to all parties. If $n - t_a$ asynchronous votes are received, they are unique and can simply be forwarded to all parties to propagate the unique output within the actual network delay δ (c.f. Section 2.1). If at most t_a parties are corrupted, then this terminates within 2δ of the honest sender giving input, meaning it is responsive. Finally, it might deadlock in synchrony. But then parties will wait for 2Δ and send a synchronous vote on the same message unless they in the meantime saw a conflicting message from the sender (through an asynchronous vote). A party who receives just $n - t_s$ synchronous votes can trust that the message is unique: in asynchrony due to any two quorums intersecting on at least $t_a + 1$ parties, and in synchrony due to $n - t_s > t_s$ and all honest parties being guaranteed to send the same (if any) synchronous votes. This serves as an alternative certificate of unique output, and uniqueness across the two types of certificates is argued along the lines of the discussion in Section 1.1.

Binary Agreement. We construct BA by reducing it to validated BA (VBA) using input justifiers. As discussed in Section 1.1 when t < n/3 input justifiers can be established simply by collecting *signed* inputs from n - t parties where at least t + 1 have sent the same bit. But in our setting the thresholds $n > 2t_s + t_a$ only a priori allow relying on an input being valid given inputs from $t_s + 1$ parties (since we could be in a synchronous execution). On the other hand, we can only expect to receive that many identical inputs by waiting to hear from $2t_s + 1$ parties, which is more than the assumed number of honest parties in a synchronous execution. If we insist on optimistic responsiveness, then we cannot collect inputs from all honest parties. In Section 4.3 we resolve this apparent deadlock.

To construct VBA the main challenge is to obtain a protocol for graded agreement that can terminate responsively. But having justified inputs helps a great deal. At a very high level, if there is a unique justified input bit, then this corresponds to the situation in reliable broadcast where an honest sender only signs a single message. In that case we can reuse the logic of the reliable broadcast to obtain a justified certificate which excludes a certificate of the same type on any different message. If we are not in this situation, then two honest parties have different justified input bits. Together these serve as a certificate that both inputs are justified. So by simply running the logic of the reliable broadcast, we are guaranteed to obtain one of these two types of certificates. Since the two outputs that are justified by the reliable broadcast logic have mutually exclusive justifiers, these 3 possible outputs can be cast as a validated flavor of *crusader agreement*. We expand on this intuition and how it can be used to obtain graded agreement with more grades in Section 4.1. From this and a common coin, piecing together an expected constant time validated BA is simple. We provide a construction in Section 4.2.

⁶ Since reliable broadcast may never terminate, we define the running time as the maximum of the time it takes between an honest sender giving input and all honest parties receiving output, and the time between the first and last honest parties receiving output, c.f. Remark 1.

Agreement on a Core Set. We finally build a more directly useful primitive in the form of Agreement on a Core Set (ACS), which is a key building block in asynchronous MPC and directly implies most standard definitions of atomic broadcast, state machine replication, public bulletin board, blockchain, total-order broadcast, etc. There are several constructions of network agnostic ACS following the common approach of first establishing local sets with a large common core and then running n parallel BA protocols to reach agreement on a set including at least the common core. However, even if we use our expected constant time BA, the expected time before the last of n parallel instances terminate is $O(\log n)$.

Instead, we base our design on the remarkably simple DAG-Rider protocol by Keidar et al. [KKNS21]. It constructs a round-based DAG where blocks are sent through reliable broadcast and point to n-t blocks from the previous round. Then after a *wave* of 4 rounds it relies on a common core being established and uses it to agree on the block of a leader using a single λ -bit coin instead of n parallel BAs. This allows the construction to be expected constant time.

In order to adapt this to the network agnostic setting, we have to live with the possibility of t_s corruptions, which may exceed the n/3 corruption bound in DAG-Rider. Conveniently, an amended version of the DAG-Rider protocol turns out to be asynchronously secure with up to t < n/2 Byzantine corruptions from black box use of reliable broadcast and common coin. In particular when instantiated with our reliable broadcast, it inherits network agnostic security, optimistic responsiveness, and has expected constant round complexity. This implies MPC with the same properties based on fully homomorphic encryptions and non-interactive proofs of knowledge following [BKLZL20,Coh16].

We finally consider a potential issue that, for some use cases of ACS it might not suffice to only guarantee a core of size $n - t_s$. As we discuss in more detail in Section 1.3 and Section 5.2, Constantinescu et al. [CGWW24] use ACS to solve convex consensus and require that their network agnostic ACS satisfies the stronger property that all honest inputs are included in the core in synchronous executions. We first show that this is incompatible with optimistic responsiveness, but also that it is simple to add a single round gadget to our protocol ensuring this property by initially waiting for all honest inputs and then giving these as input to the original ACS. It is an interesting problem if this property is necessary to solve convex consensus or computing other natural function. Another natural option is to ensure that the majority of the inputs in the core are from honest parties. This can be achieved responsively up to t_a corruptions by adding an optimistic path to the above gadget, which lets parties send an input set before they are done waiting if they manage to collect $n - t_a$ inputs.

1.3 Related Work

Momose and Ren [MR21] describe a reliable broadcast protocol similar which is similar to the one we describe in Fig. 1, but their version always waits for a timeout before outputting, and as such cannot give responsive output. They justify this choice by showing that the protocol cannot always be responsive by generalizing an impossibility on responsiveness to the network agnostic setting, or rather to their more general *multi-threshold* setting where separate thresholds for safety and liveness in both synchrony and asynchrony are studied. Specifically, they show that a protocol cannot be safe against β_s Byzantine faults in synchrony and responsive when there are $\frac{n-\beta_s}{2}$ Byzantine faults. In our setting where we insist in both liveness and safety and assume $n > t_a + 2t_s$ and $t_s > t_a$, their impossibility result implies that you cannot make protocols responsive when there are $\frac{n-t_s}{2} > \frac{t_s+t_a}{2} > t_a$ Byzantine faults. This does not contradict our result as the protocols we present are responsive only when $t \leq t_a$. We leave it as an open question if it is possible to commit responsively when there are between t_a and $\frac{n-t_s}{2}$ Byzantine faults.

There has been a few other works on round efficient consensus which all follow the overall design pattern by Blum et al. of waiting for the worst case bound of the synchronous BA rounds before the asynchronous path is attempted. Notably, Deligios et al. [DHLZ21] study round efficient synchronous BA with asynchronous fallback. However, they only consider round efficiency of synchronous executions. In order to terminate using the asynchronous path, it still needs to run λ rounds of the synchronous protocol. They also give an instantiation for MPC from an ACS procedure which waits for n parallel BAs to terminate with certainty. This is in order to preserve lock step synchrony, as protocol that terminate early have to deal with some parties terminating in different rounds.

A recent work by Deligios and Erbes [DE24] design a compiler that turns any pair of synchronous and asynchronous BA protocols into a network agnostic BA. In order for their compiler to make black box use of the BA protocols, it has to run both the synchronous and the asynchronous BA. In spite of this they still (up to a small additive constant) preserve the concrete expected round complexity of the synchronous protocol by employing a gadget that makes the asynchronous protocol terminate deterministically whenever output of the synchronous BA is unique. As a result of being able to use any synchronous protocol, their approach has better concrete round complexity than [DHLZ21] in synchronous executions, if instantiated with [GGLZ22], which achieves an error probability $(cr)^r$ in r rounds but does not allow early termination. However, the approach still has no hope of achieving constant time regardless of which concrete BA protocols are used.

Another recent work by Constantinescu et al. [CGWW24] uses ACS as a building block to solve convex consensus. Their network agnostic ACS initially follows a pattern similar to ours, as some rounds of their GTHR protocol serve a role in both the synchronous and asynchronous path. But after this phase the output is used to construct ACS by running n parallel BAs using [DHLZ21], which in asynchrony requires at least $O(\lambda)$ rounds and in synchrony has expected $O(\log n)$ round complexity. While the ACS we present is expected constant time and optimistically responsive, Constantinescu et al. point out that it does not satisfy their ACS security definition. Namely, our protocol only guarantees outputting a (fully adversarially chosen) set of $n - t_s$ parties' inputs, which in synchronous executions might only contain a single honest value. On the other hand they require a property named *Honest Core*: that the output of ACS includes all honest inputs, and rely on this property to implement convex consensus based on the output of ACS. We briefly discuss in Section 5.2 how to adapt our protocol to satisfy Honest Core in synchronous executions by adding an initial one round gadget. The resulting construction cannot terminate responsively, which we show is inherent. Finally, we show an optimistically responsive version of the gadget with a slightly weaker guarantee than Honest Core: that the majority of inputs in the core are honest. We suspect this property is sufficient to solve convex consensus.

2 Preliminaries

2.1 Network, Time and Corruption.

We consider a set of n parties \mathcal{P} connected via point to point channels and assume that message delivery is handled by an adversary who may adaptively corrupt up to t_s parties if the network is synchronous and up to t_a when it is asynchronous. We assume that $t_a+2t_s < n$ and a strong flavour of adaptive corruptions where the adversary can corrupt a party and drop messages that it sent while it was honest.

Clocks, Rounds, and Time Complexity. In synchronous executions, the adversary must deliver all messages between two honest parties within a fixed bound Δ which is known to all parties. We assume that parties have access to local clocks which for simplicity are synchronized up to an error bounded by Δ . We follow a common design pattern in partially synchronous protocols, where parties do not enter and exit rounds in lock step synchrony, but instead wait 2Δ between synchronous steps to allow other parties to respond to a message before sending the next. This additionally trivializes composition of synchronous protocols whenever outputs or inputs come with a justifier (see below) that allows a receiver to be caught up to the state of the sender and respond to messages immediately. A similar light use of clocks without lock-step synchrony has previously been used for synchronous protocols [HMW18,AMN⁺20].

In asynchronous executions there are no guarantees on when messages are delivered, and termination guarantees are only stated in terms of the maximal duration in which a message between two honest parties has been en route. In line with Canetti and Rabin [CR93], for any finite execution we use the maximal such duration to measure time complexity. Following the literature on optimistic responsiveness [PS18] we use δ to denote it.

In synchronous executions we measure time complexity in Δ while in asynchronous executions we use δ . The duration of each round is of length at most Δ when the network is synchronous and at most δ when there are t_a or fewer corruptions. This is because when $\Delta > \delta$, the activation rules relying on timeouts never trigger. This accounts for both asynchronous time complexity and optimistic responsiveness. We use time and rounds interchangeably, and note that all the presented protocols have (expected) constant round and time complexity.

Threshold Signatures and PKI. In line with most literature in the area we follow the Dolev-Yao model [DY81] and idealize signatures schemes as ideal objects with λ bit sized signatures and perfect unforgeability. We assume a signature scheme (Gen, Sig, Ver) and a PKI, where in some initial synchronous round all parties $\mathsf{P}_i \in \mathcal{P}$ sample $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^{\lambda})$ and send vk_i to a trusted third party which makes public $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$. The adversary gets to see vk_i for all honest P_i before picking its own keys, and it does not have to pick its own keys at random, it can use any vk_j for a corrupted P_j . We will also make use of threshold

signatures with reconstruction thresholds t_s+1 , t_a+1 , $n-t_s$, and $n-t_a$. For this we define a t threshold signature scheme with unique signatures (Setup_t, Sig_t, Ver_t, ShareVerify_t, Combine_t), where $(vk, sk_1, \ldots, sk_n) \leftarrow Setup_t(1^{\lambda})$ generates a verification key vk and a signing share sk_i for P_i , Sig_{t,sk_i} partially signs m, ShareVerify_t can verify a partial signature, and Combine_t computes $\sigma = \mathsf{Sig}_{t,sk}(m)$ from t verified shares.

Common Coin. We assume a protocol for common coin Π_{CC} . If all honest parties invoke Π_{CC} with an identifier *id*, the functionality gives a unique coin c_{id} to all honest parties. The value of this coin is unpredictable until the first honest party invokes Π_{CC} with *id*. This can be implemented in asynchrony from threshold signatures following [CKS00] under honest majority, which is implied by $2t_s + t_a < n$.

Justified messages, inputs and ouputs. In many cases the inputs, outputs and messages sent in our protocols have a corresponding *justifier*, which is a convenient abstraction allowing parties to be convinced that the information they receive satisfies certain validity properties by buffering the information until it is seen to satisfy some justifier predicate J. Additionally, in order to make it easy to justify information based on other justified information, where our definitions are in line with [KNTT22] (see Definition 2). A justifier predicate is said to be monotone if for any message m that J justifies in the view of a party P at some point in time, m remains justified in P's view. J is said to be propagating if any message that is justified in P's view eventually becomes justified in every party P's view. As an example, consider the previously discussed reduction from BA to VBA where parties send a t + 1 threshold signature to demonstrate that an honest party received the bit as input [CKPS01]. This is clearly a monotone justifier, and if parties additionally forward the signature, then it is also propagating. Similarly all blocks in the DAG-Rider protocol by Keidar et al. [KKNS21] can be cast as justified by being send through reliable broadcast and referencing only justified messages. Again this justifier is monotone and since reliable broadcast satisfies totality, it is also propagating.

Some previous works using similar abstractions for justifiers introduce an explicit predicate for every justifier [DYMM⁺20], but since in our case almost every message is associated with a justifier that would be quite verbose. Instead, we might write: "wait until t + 1 signatures on some bit b are received, and forward b justified by these t + 1 signatures." So, to simplify notation considerably while keeping the formal meaning of each justifier unambiguous, we introduce the following conventions allowing us to define justifiers ad hoc when defining the protocol rules giving rise to the justified information.

We consider a formal model where all messages sent in a protocol implicitly come with a message identifier. This is a typically assumed but often ignored abstraction which allows having more than one instance of the protocols and primitives described with meaningful security properties, without writing an explicit copy for each one. We do however deliberately keep this hidden from the specifications of our security definitions and protocol descriptions. ⁷ This is to not clutter them with unnecessary syntax. Yet, we will still tacitly assume that

⁷ With the exception that we do make explicit use of message identifiers to compose and disambiguate between protocol instances in the causal cast framework presented in Appendix A.1.

each message is associated with this unique string and take advantage of it when defining justifiers.

Definition 1 (Message identifiers - informal). In our protocols all messages will have a message identifier mid specifying which protocol it belongs to, what round of the protocol it comes from, sent by whom and so on. Each message identifier mid specifies a party P^{mid} , which we think of as the party which is to send the message identified by mid. Each mid also specifies a so-called justifier J^{mid} , which is a predicate depending on the message m and the local state of a party. When we write pseudo-code then we write $J^{mid}(m)$ to denote that the party P executing the code computes J^{mid} on m using its current state. In definitions and proofs we write $J^{mid}(m, P, \tau)$ to denote that we apply J^{mid} to m and the local state of P at time τ .

We further characterize justifiers by requiring the following useful properties (adapted from a similar definition in [KNTT22]).

Definition 2 (Justifier). For a message identifier mid we say that J^{mid} is a justifier if the following properties hold.

- **Monotone:** If for an honest P and some time τ it holds that $J^{\text{mid}}(m, \mathsf{P}, \tau) = \top$ then at all $\tau' > \tau$ it holds that $J^{\text{mid}}(m, \mathsf{P}, \tau') = \top$.
- **Propagating:** If for honest P and some point in time τ it holds that $J^{\text{mid}}(m, \mathsf{P}, \tau) = \top$, then eventually the execution will reach a time τ' such that $J^{\text{mid}}(m, \mathsf{P}', \tau') = \top$ for all honest parties P'.

In the protocols we define, many internal protocol messages will come with a justifier, which intuitively is used to force the adversary to behave consistently with honest input output behaviour. Our proofs will often follow a pattern where we show that for any message of a certain type, some predicate P holds for all *possible justified messages*, by which we mean that it holds for honest messages which are sent to all parties and in addition that the adversary cannot even cook up messages which look justified to some honest party but do not have the property P.

Definition 3 (Possible Justified Messages). Let Π be protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. When we say that an ℓ -ary predicate P holds for all possible justified messages we mean that any PPT adversary should win the following game with negligible probability.

- Run an execution of Π under attack by the adversary.
- At any point the adversary may output a sequence of triples $(\mathsf{P}^1,\mathsf{mid}^1,m^1),\ldots,(\mathsf{P}^\ell,\mathsf{mid}^\ell,m^\ell).$
- We say that the adversary wins if the message identifiers $\operatorname{mid}^1, \ldots, \operatorname{mid}^{\ell}$ identify messages of Π , $\mathsf{P}^1, \ldots, \mathsf{P}^{\ell}$ are honest (but not necessarily distinct) parties, for $j = 1, \ldots, \ell$ it holds that $J^{\operatorname{mid}_j}(m^j) = \top$ at P^j , and $P(m^1, \ldots, m^{\ell}) = \bot$. Otherwise the adversary looses the game.

We will mostly use this to talk about predicates that are satisfied by a subset of all possible justified messages of a protocol, specifically ones prompted by a concrete activation rule of the protocol. This is covered by Definition 3 by making the class of messages part of the predicate P.

Note that it is important for the definition to be meaningful that honest parties send their messages through a channel that leak them to the adversary, as otherwise an honest message that does not satisfy the P does not imply the adversary winning the game. For this reason it will be convenient when defining security properties to use all *possible justified outputs* of a justified protocol as shorthand for the outputs of honest parties as well as anything the adversary could convincingly claim to have gotten as output. This motivates Definition 4 which considers a predicate P only on outputs of the protocol Π , but then requires the outputs to be sent to all parties. As honest parties send their outputs, the adversary could easily win the game if an honest party receives output that does not satisfy P.

Definition 4 (Possible Justified Outputs). Let Π be protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ with output justifier J. Let Π' be the protocol Π with only change being that each party on getting output, sends their output to all parties if this was not already done. Consider an ℓ -ary predicate P, and let P' be equivalent to P with the only change being that its domain is restricted by making it evaluate to \bot when any of its inputs are not an output of Π' . When we say that an ℓ -ary predicate P holds for all possible justified outputs of Π we mean that P' holds for all possible justified messages of Π' .

Most of our protocols will be *justified protocols* which puts validity constraints on the inputs and outputs.

Definition 5 (Justified Protocol). A justified protocol Π can have input and output justifiers.

- **Input justifier:** If a protocol has an input justifier J_{IN} it means that a message identifier mid_{IN} is associated with the input, $J_{IN} = J^{\text{mid}_{IN}}$, and it is $\mathsf{P}^{\text{mid}_{IN}}$ which gets the input. Furthermore, if $\mathsf{P}^{\text{mid}_{IN}}$ is honest then it is guaranteed that $J_{IN}(m, \mathsf{P}^{\text{mid}_{IN}}, \tau) = \top$ when m is input to $\mathsf{P}^{\text{mid}_{IN}}$ at time τ .
- **Output justifier:** If a protocol has an output justifier J_{OUT} it means that a message identifier mid_{OUT} is associated with the output, $J_{OUT} = J^{\text{mid}_{OUT}}$, it is $P^{\text{mid}_{OUT}}$ which gives the output, and when it gives the output it may send the output to all parties with message identifier mid_{OUT}. Furthermore, if $P^{\text{mid}_{OUT}}$ is honest and gets output m then it is a security property of the protocol that if P does send its output to all parties at time τ , then $J_{OUT}(m, P, \tau) = \top$.

When necessary, we use the notation ΠJ_{IN} and ΠJ_{OUT} to disambiguate J_{IN} and J_{OUT} from the input and output justifiers of other protocols.

2.2 Problem definitions

We include standard definitions of the main studied primitives, and defer definitions of additional primitive used building blocks to the relevant sections.

Reliable Broadcast. Reliable broadcast allows parties to send messages with strong consistency guarantees, which guarantees that if one honest party receives the message, then all honest parties receives the same message. In contrast to synchronous broadcast, it is not possible to reach agreement on the fact that no message was sent.

Definition 6 (Reliable Broadcast). Let Π be protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where a designated sender S has input $m \in \{0,1\}^*$ and each party P_i may output a message m_i . We say that Π is a secure reliable broadcast protocol if the following properties hold:

- Validity: If S is honest and has input, and honest P_i has output then $m_i = m$.
- **Agreement:** If honest parties P_i and P_j have output then $m_i = m_j$.
- **Eventual Output 1:** If S is honest and has an input, and all honest P_i start running the protocol, then eventually all honest P_i have an output m_i .
- **Eventual Output 2:** If an honest P_j has output m_j , and all honest parties start running the protocol then eventually each honest P_i has output m_i .

Remark 1. Note that RB does not have a unified termination property, but requires termination only given an honest sender, of if an honest party already received output. When we discuss the latency of reliable broadcast and claim that it is constant time or responsive, then we consider the duration between an honest party either sending or receiving a message through reliable broadcast and the last honest party receiving the message. We let the maximal such duration define the running time of the protocol.

Binary Agreement. Binary agreement allows the parties to reach agreement on a bit decision, which is guaranteed to be the input of an honest party.

Definition 7 (BA). Let Π be protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i holds an input $x_i \in \{0, 1\}$ and outputs a value $y_i \in \{0, 1\}$ upon terminating. We say that Π is a secure BA protocol if the following properties hold:

Liveness: If every honest party P_i has input $x_i \in \{0, 1\}$, then eventually every honest party P_j will have output $y_i \in \{0, 1\}$.

Agreement: If honest parties P_i and P_j have output then $y_i = y_j$. **Validity:** If honest party P_i has output, then for some honest party P_j : $y_i = x_j$.

Validated Binary Agreement. Valididated BA relaxes the validity requirement of BA, so that the output just has to satisfy some predicate. On the other hand the outputs is typically assumed to come with a transferable certificate.

Definition 8 (Validated BA). Let $\Pi(J_{IN})$ be a protocol parameterized by an input justifier J_{IN} and executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i receives an input $x_i \in \{0, 1\}$ for which $J_{IN}(x_i, \mathsf{P}_i, \tau) = \top$ at the time of input τ and outputs a value $y_i \in \{0, 1\}$ satisfying an output justifier J_{OUT} upon terminating. We say that $\Pi(J_{IN})$ is a secure VBA protocol if the following properties hold:

Liveness: If every honest party P_i has justified input $x_i \in \{0, 1\}$ (where $J_{IN}(x_i) = \top$), then eventually every honest party P_j will have justified output $y_j \in \{0, 1\}$ (where $J_{OUT}(y_j, \mathsf{P}_j, \tau) = \top$ at the time of output τ).

Justified Agreement: For all possible justified outputs y and y': y = y'. **Justified Validity:** If $J_{OUT}(y) = \top$, then $J_{IN}(y) = \top$.

Agreement on a Core Set. Agreement on a core set (ACS) allow a set of parties to reach agreement on a subset of their inputs.

Definition 9 (Agreement on a Core Set). Let Π be a protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. All honest P_i have an input B_i and may obtain output U_i consisting of a set of inputs.

- **Liveness:** If all honest parties start running the protocol with a J_{IN} -justified input then eventually all honest parties have a J_{OUT} -justified output.
- **Validity:** If an honest party P_i obtains output U_i then for all honest P_j where $(P_j, B_j) \in U_i$ it holds that P_j had input B_j .

Agreement: If a pair of honest parties P_i and P_j obtain outputs U_i and U_j then $U_i = U_j$. t-Large Core: If an honest party P_i obtains output U_i it holds that $|U_i| \ge n - t$.

3 Reliable Broadcast

We present a reliable broadcast protocol and show that the protocol has network agnostic security if the thresholds satisfy $n > t_a + 2t_s$.

The logic of our reliable broadcast somewhat resembles the reliable broadcast protocols by Momose and Ren [MR21] and Ghinea et al. [GLZW22], but we do not require synchronized starts and are able to output responsively. In the protocols of [MR21,GLZW22], parties first report the message they received from the sender by sending it to all parties. Then (after a timeout), they vote for it if no conflicting messages from the sender have been seen after all honest parties have had a chance to report what they saw from the sender. When $n-t_s$ votes are collected, they serve as a certificate of the output's uniqueness and can be relayed to all parties to ensure totality. The vote message in these two protocols plays the same role as our synchronous vote, in the sense that in synchronous executions all honest (synchronous) votes are on the same message. This ensures unique output: in synchrony because $n - t_s$ votes include an honest vote and in asynchrony because any two sets have $n - 2t_s > t_a$ votes in common.

Where our logic differs is that the initial report of the sender's proposal is also seen as a *asynchronous* vote, and $n - t_a$ asynchronous votes form a certificate without waiting for Δ . This larger set of signatures is still a valid certificate in asynchrony, but now also in synchrony because any two asynchronous certificates share $n - 2t_a > t_s$ asynchronous votes. To see that the message is unique across the two different certificate types, observe that they share $n - t_s - t_a > t_s$ votes, and that an honest party will not send an asynchronous and synchronous vote on two different messages.

Reliable Broadcast Π_{RB}

- **Input:** The input of the designated sender S is m. In response to this input S sends $(m, \sigma_s = Sig_{sk_s}(m))$ to all parties. Create initially empty sets SignedAsync and SignedSync.
- Asynchronous Vote: P_i : Upon receiving (m, σ_s) from S, where $\mathsf{Ver}_{\mathsf{vk}_s}(m, \sigma_s) = \top$ and where no such message was received before and there is no $(\mathsf{P}_j, m_j, \sigma_j) \in \mathsf{SignedAsync}$ with $m_j \neq m$, proceed as follows:
 - 1. Let $\sigma_i = \text{Sig}_{sk_i}((\text{ASYNC}, m))$ and send (m, σ_s, σ_i) to all parties.
 - 2. Add $(\mathsf{P}_i, m, \sigma_i)$ to SignedAsync.
 - 3. Record current time τ_i .
- **Collect Asynchronous Votes:** All parties: Upon receiving $(m_j, \sigma_s, \sigma_j)$ from P_j , where $\mathsf{Ver}_{\mathsf{vk}_s}(m_j, \sigma_s) = \top$ and $\mathsf{Ver}_{\mathsf{vk}_j}((\mathsf{ASYNC}, m_j), \sigma_j) = \top$ and no such value was received from P_j before, add $(\mathsf{P}_j, m_j, \sigma_j)$ to SignedAsync.
- Synchronous Vote: P_i : At $\tau_i + 2\Delta$ if there are at least $n t_s$ values $(\mathsf{P}_j, m, \cdot) \in \mathsf{SignedAsync}$ and there does not exist $(\mathsf{P}_k, m', \cdot) \in \mathsf{SignedAsync}$ where $m' \neq m$, then let $\sigma_i = \mathsf{Sig}_{\mathsf{sk}_i}((\mathsf{SYNC}, m))$ and send (m, σ_i) to all parties.
- **Collect Synchronous Votes:** All parties: Upon receiving (m_j, σ_j) from P_j , where $\mathsf{Ver}_{\mathsf{vk}_j}((\mathsf{SYNC}, m_j), \sigma_j) = \top$ and no such value was received from P_j before, add $(\mathsf{P}_j, m_j, \sigma_j)$ to SignedSync.
- Asynchronous Output: All parties: If there exists m such that there are $n t_a$ values $(\mathsf{P}_j, m, \sigma_j) \in \mathsf{SignedAsync}$ then let $\Sigma = \{(\mathsf{P}_j, \sigma_j)\}_{(\mathsf{P}_j, m, \sigma_j) \in \mathsf{SignedAsync}}$, output m, send (m, Σ) to all parties, and terminate.
- Synchronous Output: All parties: If there exists m such that there are $n t_s$ values $(\mathsf{P}_j, m, \sigma_j) \in \mathsf{SignedSync}$ then let $\Sigma = \{(\mathsf{P}_j, \sigma_j)\}_{(\mathsf{P}_j, m, \sigma_j) \in \mathsf{SignedSync}}$, output m, send (m, Σ) to all parties, and terminate.

Output by Relay: Upon receiving (m, Σ) from any party where either

- Σ contains $n t_s$ values $(\mathsf{P}_j, m, \sigma_j)$ for distinct P_j such that $\mathsf{Ver}_{\mathsf{vk}_j}((\mathsf{SYNC}, m), \sigma_j) = \top$ (call such a value synchronous-valid), or
- Σ contains $n t_a$ values $(\mathsf{P}_j, m, \sigma_j)$ for distinct P_j such that $\mathsf{Ver}_{\mathsf{vk}_j}((\mathsf{ASYNC}, m), \sigma_j) = \top$ (call such a value asynchronous-valid),
- output m, send (m, Σ) to all parties, and terminate.

Fig. 1. A protocol for RB with dual thresholds t_s and t_a and designated sender S.

Theorem 1 (Network Agnostic reliable broadcast). Π_{RB} is a reliable broadcast protocol for the network agnostic model where either the network is synchronous and there are at most t_s corruptions or the network is asynchronous and there are at most t_a corruptions. All parties terminate within time $2\delta + 2\Delta$. Furthermore, if $t \leq t_a$ and the sender is honest then all honest parties terminate within time 2δ .

It is not hard to see that the protocol has eventual output and validity. The running time is also straight forward. The main observation is that when corruption is bounded by t_a then within time δ the $n - t_a$ honest parties trigger **Asynchronous Vote** and then within δ all parties have an asynchronous-valid output. We sketch why the protocol has agreement. The pivotal property which the protocol has by design is the following.

Lemma 1 (Synchronous Vote Agreement). If two honest P_i and P_j send (m_i, σ_i) and (m_j, σ_j) in Synchronous Vote then $m_i = m_j$.

Proof. Consider P_i and P_j sending (m_i, σ_i) and (m_j, σ_j) in **Synchronous Vote**. Then they sent some $(m_i, \sigma_s, \sigma'_i)$ and $(m_j, \sigma'_s, \sigma'_j)$ in **Asynchronous Vote**. Assume P_i sent $(m_i, \sigma_s, \sigma'_i)$ first. Then P_j waited until at least $\tau_j + 2\Delta \geq \tau_i + 2\Delta$, i.e. at least 2Δ after $(m_i, \sigma_s, \sigma'_i)$ was

sent before sending its synchronous vote. Assume that the network is synchronous. Then P_j saw $(m_i, \sigma_s, \sigma'_i)$ before $\tau_j + 2\Delta$ which is the earliest point (m_j, σ_j) could have been sent. Therefore $m_j = m_i$, or $(m_i, \sigma_s, \sigma'_i)$ would have blocked the sending of (m_j, σ_j) . Assume then that the network is asynchronous. Then by assumption $t \leq t_a$. Recall that $2t_s + t_a < n$. If P_i sent (m_i, σ_i) then it saw $n - t_s$ values $(\mathsf{P}_k, m_i, \cdot) \in \mathsf{SignedAsync}$. If P_j sent (m_j, σ_j) then it saw $n - t_s$ values $(\mathsf{P}_k, m_i, \cdot) \in \mathsf{SignedAsync}$. If P_j sent (m_j, σ_j) then it saw $n - t_s$ values $(\mathsf{P}_k, m_i, \cdot) \in \mathsf{SignedAsync}$. If P_j sent (m_j, σ_j) then it saw $n - t_s$ values $(\mathsf{P}_k, m_j, \cdot) \in \mathsf{SignedAsync}$. This means they saw values from $n - 2t_s > t_a$ common parties. So they saw $(\mathsf{P}_k, m_i, \cdot)$ and $(\mathsf{P}_k, m_j, \cdot)$ from at least one joint honest P_k . Therefore $m_i = m_j$.

Lemma 2 (Network Agnostic Agreement). If P_i and P_j are honest and output m_i and m_j then $m_i = m_j$.

Proof. If P_i outputs m_i then it saw a valid (m_i, Σ_i) and if P_i outputs m_i then it saw a valid (m_i, Σ_i) . If any of the parties saw a synchronous-valid value, then rename the parties such that P_i saw one. This gives three cases on the validity flavour of (m_i, Σ_i) - (m_i, Σ_i) : synchronous-synchronous, synchronous-asynchronous, and asynchronous-asynchronous. Assume first they both are synchronous-valid. Recall that $2t_s - t_a < n$. Among the $n - t_s$ parties in Σ_i there is at least one honest party as $n - t_s > t$, where t is the actual number of corruptions. Similarly, among the $n - t_s$ parties in Σ_i there is at least one honest party. Agreement then follows from Lemma 1. Assume then that both (m_i, Σ_i) and (m_j, Σ_j) are asynchronous-valid. Then among the $n - t_a$ parties in Σ_i and the $n - t_a$ parties in Σ_j there are at least $n - t_a - t_a > 2t_s - t_a > t_s \ge t$ common parties. Therefore there is at least one common honest party. Honest parties sign at most one message m. Assume then that (m_i, Σ_i) is synchronous-valid and (m_j, Σ_j) is asynchronous-valid. Among the $n - t_s$ parties in Σ_i and the $n - t_a$ parties in Σ_j there are at least $n - t_s - t_a > t_s$ common parties. Since $t_s \geq t$, where t is the actual number of corruptions, it follows that there is at least one honest party in common among Σ_i and Σ_j . Clearly, if an honest party signs both (SYNC, m) and (ASYNC, m') then m' = m. Therefore $m_i = m_j$.

4 Binary Agreement

We construct binary agreement (Definition 7) by first solving the simpler problem of validated binary agreement (Definition 8) and then giving a separate reduction from BA to VBA in Section 4.3 in line with Cachin et al. [CKPS01]. VBA is in turn reduced to validated proxeensus and a common coin.

As discussed in our overview (Section 1.2), obtaining justified inputs for VBA in an optimistically responsive manner is surprisingly challenging. To resolve the apparent deadlock, we let parties first optimistically wait to see $t_s + 1$ identical inputs, as such a set of inputs is guaranteed to eventually arrive in asynchronous executions, and is always a sound certificate. But in a synchronous execution, such a set may never arrive. Notice however, that if all $(> t_s)$ honest parties have the same input, then a certificate arrives within a network delay of them getting input. So if a party waits long enough to receive the inputs of all honest parties in a synchronous execution and did not see such a set, then either (1) both bits have been given as inputs to honest parties or (2) the network is asynchronous. If (1) is true, then both bits are valid and if (2) is true then a certificate of signed identical inputs from only $t_a + 1$ parties is enough to demonstrate validity. Since we want the certificates to be transferable and "having waited long enough" is not a justifiable statement, we first send an *intention* to downgrade the threshold and require the downgraded certificate of $t_a + 1$ signed inputs to include signed intentions from $t_s + 1$ parties, of which at least one was honest and waited long enough to ensure validity.

The main remaining challenge is to construct graded agreement.

4.1 Validated Graded Agreement

We first abstract a high level idea from [Kam25] which defines validated proxeensus (c.f. Definition 10). As a special case: consider a validated flavor of crusader agreement which is to crusader agreement what VBA is to BA. In crusader agreement, parties input bits and receive output in the form of a bit or and undecided value '?', with *validity* requiring that if all honest parties input the same bit then that should be the unique output and *graded agreement* requiring that no honest parties have different bits as output. In validated crusader agreement, validity is replaced with the condition that if only on input can be justified, then it should be the uniquely justified output. Similarly, graded agreement. More precisely an adversary cannot construct an output and a corresponding justifier that convinces an honest party, unless the output has graded agreement with all other such outputs.⁸ From this, validated graded consensus with more grades can be constructed by sequentially applying this protocol.

To construct validated crusader agreement we notice that the logic from the reliable broadcast almost solves the problem. The justified votes on the senders signed message can be seen as solving a weak consensus protocol (sometimes called *reliable consensus* [BKLZL20]) where all honest parties having the same input leads to them obtaining that message as output, and where no conflicting outputs are possible. When inputs are justified bits, then we can make parties send their justified input to all parties and also give it as input to this reliable consensus. Now, either all honest parties had the same input, and the reliable consensus terminates, or at least two different justified inputs will propagate. This solves validated crusader agreement by outputting the bit from reliable consensus, or an undecided value if both possible inputs are seen to be justified.

With this logic in place, the primitive can be used to instantiate validated procensus with $2^G + 1$ grades by invoking the primitive sequentially G times. We now present the more general solution using the definition of validated procensus by Kamp [Kam25].

Definition 10 (Validated Proxeensus [Kam25]). Let $\Pi_{VPROX-G}(J_{IN})$ be a protocol for n parties, parameterized by an input justifier J_{IN} , and outputting $y \in \{0, \ldots, G-1\}$ satisfying an output justifier J_{OUT} . We say that $\Pi_{VPROX-G}(J_{IN})$ solves VProx(G) if the following holds:

⁸ We define *all possible justified* messages and output in Definitions 3 and 4.

- **Liveness** If every honest party P_i have justified input $x \in \{0, 1\}$ where $J_{\text{IN}}(x) = \top$, then eventually every honest party P_j will have justified output $y \in \{0, \ldots, G-1\}$ where $J_{\text{OUT}}(y) = \top$.
- **Justified Graded Agreement** For all possible justified outputs y and $y': |y y'| \leq 1$.

Justified Validity If b is the only possible J_{IN} justified bit, then $y = b \cdot (G - 1)$ for all possible justified outputs y.

Validated Proxcensus $\Pi_{VProx-2s-1}(J_{in})$

We implement $\Pi_{\text{VPROX}-2s-1}$ from $\Pi_{\text{VPROX}-s}$ for $s \geq 2$. As a base case for the recursion, define $\Pi_{\text{VPROX}-2}(J_{\text{IN}})$ to output its input with $J_{\text{OUT}} = J_{\text{IN}}$ (trivially satisfying Definition 10).

- 1. On input x_i where $J_{IN}(x_i) = \top$, P_i runs $\Pi_{VPROX-s}(J_{IN})$ with input x_i . Let z_i denote the output.
- 2. P_i multicasts (ASYNCPROPOSAL, z_i) with a signature, a shares of threshold signature schemes with thresholds $n t_a$ and $n t_s$. It also records the current time τ_i .
- 3. Upon receiving justified (ASYNCPROPOSAL, z) and (ASYNCPROPOSAL, z 1), P_i lets $y_i = 2z 1$, multicasts (ASYNCPAIR, y_i) and terminates with output y_i . The value is justified by the justifiers of z and z 1.
- 4. Upon receiving justified (ASYNCPROPOSAL, z) from $n t_a$ distinct parties, P_i lets $y_i = 2z$, multicasts (ASYNCQUORUM, y_i) and terminates with output y_i . The value is justified by an $n t_a$ threshold signature on z.
- 5. After time $\tau_i + 2\Delta$ and having collected (ASYNCPROPOSAL, z) from at least $n t_s$ distinct parties, P_i multicasts (SYNCPROPOSAL, z) with a signature and a share of a $t_s + 1$ threshold signature. The value is justified by an $n - t_s$ threshold signature on z.
- 6. On having received $t_s + 1$ justified (SYNCPROPOSAL, z) messages from $t_s + 1$ distinct parties, P_i lets $y_i = 2z$, multicasts (SYNCQUORUM, y_i), and terminates with output y_i . The value is justified by a $t_s + 1$ threshold signature on z.

Fig. 2. A recursive description of the validated proxcensus protocol.

Lemma 3. For any two justified values (ASYNCPAIR, y) and (ASYNCPAIR, y') produced in step 3, y = y'.

Proof. By induction hypothesis $\Pi_{VPROX-s}(J_{IN})$ satisfies Justified Graded Agreement, so y and y' are uniquely defined by the two possible justified outputs of the inner protocol.

Lemma 4. For any two justified values (ASYNCQUORUM, y) and (ASYNCQUORUM, y') produced in step 4, y = y'. Additionally, for any justified (SYNCPROPOSAL, z) sent in step 5, y = 2z.

Proof. Any two $n-t_a$ threshold signatures justifying y and y' have shares from $n-2t_a > t_s \ge t$ parties in common. Regardless of the network condition, at most t_s parties are corrupted, so y = y' as no honest party sends more than one AsyncProposal. When comparing y and z, the justifier of y is an $n - t_a$ threshold signature on y/2, while the justifier for z is an $n - t_s$ threshold signature on z. At least $n - t_a - t_s > t_s \ge t$ parties would need to have proposed both, so they must be equal, so y = 2z.

We show that justified output of the asynchronous path satisfy Justified Graded Agreement.

Lemma 5 (Asynchronous Graded Agreement). For any justified output (AsyncPAIR, y) and (AsyncQuorum, y'), $|y - y'| \leq 1$.

Proof. Consider y justified by step 3 and y' justified by step 4. We know by the above two lemmas that they are uniquely defined. We show that |y - y'| = 1. By Justified Graded Agreement of $\Pi_{\text{VPROX}-s}(J_{\text{IN}})$, we know that the two values z and z-1 justifying y are the only possible justified outputs of $\Pi_{\text{VPROX}-s}(J_{\text{IN}})$. Since $y' \in \{2z, 2z-1\}$ we have $y = 2z-1 = y'\pm 1$.

Lemma 6 (Honest Synchronous Proposal Agreement). For any two honest messages (SYNCPROPOSAL, z) and (SYNCPROPOSAL, z) sent through step 5, z = z'.

Proof. If the network is asynchronous, then $t = t_a$ and the two sets of size $n - t_s$ intersect on $n - 2t_s > t_a$ parties, so the values are identical. If instead the network is synchronous, then consider the first honest party P_i that sends a AsyncProposal message and starts the timeout in step 2 at time τ_i . By synchrony all honest parties receive the justified message from P_i not later than $\tau_i + \Delta$, and send their own AsyncProposal which reaches all honest parties no later than $\tau_i + 2\Delta$. Since P was the first honest party to start waiting, no honest party finishes waiting before $\tau_i + 2\Delta$. So any honest party sending a (SyncProposal, z) message does so on the single value z that was included in all honest (SyncProposal, z) messages.

Lemma 7. For any two justified messages (SYNCQUORUM, y) and (SYNCQUORUM, y') sent in step 6, y = y'.

Proof. The messages are justified by a $t_s + 1$ threshold signature from $t_s + 1$, with a a contribution from an honest party who sent, (SYNCPROPOSAL, y/2). By Lemma 6 all honest SYNCPROPOSAL messages are consistent.

Lemma 8 (Justified Graded Agreement). The protocol $\Pi_{VPROX-2s-1}$ is a validated proxcensus protocol as defined in Definition 10.

Proof. Combining the previous lemmas, we have that all outputs are identical, except those produced in step 3, but these are at most off by one. The communication complexity is $O(n^2\lambda)$. The round complexity is at most 2 per recursive call, i.e.logarithmic in the number of grades and the protocol is responsive up to t_a corruptions.

4.2 Validated Binary Agreement

We present our VBA, which roughly follows the approach of Feldman and Micali [FM97] using graded agreement and a common coin to reach consensus in an expected constant number of rounds.

In order to avoid explicit use of session identifiers, we will exploit the fact that each invocation of validated proxcensus uses a unique justifier and (abusing notation slightly) use the justifier to distinguish between different sessions.

Theorem 2. The protocol Π_{VBA} presented in Fig. 3 is a secure validate binary agreement according to Definition 8.

Validated Binary Agreement Π_{VBA} (J_{in}) **Input:** On input x_i justified by J_{IN} , party P_i initiates an iteration counter r = 0 and defines $x_i^0 = x_i$ and $J_0 = J_{\text{IN}}$. P_i then executes the following steps. **Proxeensus:** P_i runs $\Pi_{\mathsf{VPROX}-7}(J_r)$ with input x_i^r . **Coin Flip:** On justified output q_i^r from $\Pi_{VPRox-7}(J_r)$, P_i invokes Π_{CC} with identifier r. **Decision:** On output c_r from $\Pi_{\rm CC}$, P_i waits until it also has a justified q_i^r and then acts according to the following rules: **Output:** if $g_i^r = 0$ (or $g_i^r = 6$): let $y_i = 0$ (or $y_i = 1$), multicast and output y_i justified by g_i^r being $\Pi_{\text{VPROX}-7}(J_r).J_{\text{OUT}}$ justified. **Maintain consistency:** if $g_i^r = 1$ (or $g_i^r = 5$): let $x_i^{r+1} = 0$ (or $x_i^{r+1} = 1$) be justified by g_i^r being $\Pi_{\text{VPROX}-7}(J_r).J_{\text{OUT}}$ justified. **Use coin:** otherwise: let $x_i^{r+1} = c_r$ be justified by g_i^r being $\prod_{VPROX-7} (J_r) \cdot J_{OUT}$ justified. If P_i did not yet output, it defines J_{r+1} in accordance with the rules above, lets r = r + 1, and goes back to the **Proxcensus** step. **Output by relay:** Upon receiving an output y_i , justified according to the output rules above for any round r with justified grade g_i^r , P_i lets $y_i = y_j$, multicasts and outputs y_i . The value is justified by g_i^r being a $\Pi_{\text{VPROX}-7}(J_r).J_{\text{OUT}}$ justified grade and (g_i^r, y_i) being either (0,0) or (6,1). Fig. 3.

- *Proof.* Justified Validity: We first use Justified Validity of the validated proxeensus protocol to argue Justified Validity of the VBA. If only one input bit b is justified, then only the output $6 \cdot b$ can be justified. So protocol terminates for all parties after the first iteration where the only possible justified output is b.
- **Justified Agreement:** Similarly we use graded agreement from the validated proxensus protocol to argue Justified Agreement. If a justified output b is possible, then the output b is justified (by grade $6 \cdot b$), then by design of our protocol all justified inputs to the next round of consensus would be on the same bit b, and this inductively results in all future iterations being given input b and giving output $6 \cdot b$. (In particular it means that all parties terminate at latest in the next iteration, as there are no justified inputs to future iterations.)
- Liveness: Finally, for liveness, note that there are three possible "middle grades" $\{4, 5, 6\}$ that make parties use a coin which is unpredictable to the adversary at the point where the first honest party gets output from proxensus. If no party uses the coin to decide their input to some iteration, then by design all parties have already terminated or all possible justified outputs lie on the same side of the middle grades and can only be used to justify some unique input in the next round. This leads to all parties terminating after that round. If all parties use the coin, then they also give the same input in the next round. So we only need to consider the case where grades are distributed between 1 and 2 (as the other case is symmetric). Since the adversary cannot predict the coin at the time the first party gives output from proxensus and the outputs have graded agreement, then this case results in the protocol terminating (in the next iteration) with probability $\geq 1/2$.

4.3 Reducing BA to VBA

We introduce a simple justified protocol, Π_{IVG} , which lets parties obtain a validated input for the VBA protocol, reducing BA to VBA as proposed by Cachin et al. [CKPS01]. Given these two building blocks the protocol shown in Fig. 4 implements binary agreement as defined in Definition 7.

Binary Agreement Π_{BA}

- On input x_i , P_i runs Π_{IVG} with input x_i .

- On output b_i (which is justified by $\Pi_{\text{IVG}}.J_{\text{OUT}}$) from Π_{IVG} , run $\Pi_{\text{VBA}}(\Pi_{\text{IVG}}.J_{\text{OUT}})$ with input b_i .

Fig. 4.

The protocol Π_{IVG} corresponds to the step where inputs are validated using threshold signature in [CKPS01], however in our setting it is a bit more involved than sending and collecting shares of a t + 1 threshold signature scheme.

We define its security formally in Definition 11.

Definition 11 (Input Validation Gadget). Let Π be a protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i holds an input $x_i \in \{0,1\}$ and outputs a value $y_i \in \{0,1\}$ satisfying an output justifier J_{OUT} upon terminating. We say that Π is a secure *IVG* if the following properties hold:

- **Liveness** If every honest party P_i has input $x_i \in \{0, 1\}$, then eventually every honest party P_j will have justified output $y_j \in \{0, 1\}$ (where $J_{OUT}(y_j, \mathsf{P}_j, \tau) = \top$ at the time of output τ).
- **Justified Validity** For all possible justified outputs y, some honest party P_i had input $y_i = y$.

Remark 2. The first concern to address is that until now, we only assumed that (in synchronous executions) Δ is a sound upper bound on δ , which in particular implies that all honest parties are able to receive inputs within Δ of the first party sending a message in any given protocol. This was sufficient because we followed a pattern where, whenever a party used a timeout to hear from any honest parties who might have potentially conflicting information, they also sent a justifier. This justifier allowed the receiver to immediately respond with its own message, if it had not previously sent anything. If nothing else, the receiver could simply reply with the same message and justifier. This approach falls apart when we move to problems where inputs have no objective justification. So instead, we assume that all honest parties receive input to Π_{IVG} and Π_{BA} within a duration of at most Δ .

To construct the input justifier in Fig. 5 we first have all parties multicast their input bits and optimistically try to collect signature shares on the same input bit from $t_s + 1$ parties, which would mean that is the input of at least one honest party regardless of the network setting. We let this be a certificate of validity for that bit. If all honest parties have the same

⁻ On output y_i from Π_{VBA} , output y_i .

input and the network is synchronous, then clearly this would happen within Δ of the last honest party sending.

On the other hand, if we assume that honest parties receive inputs at most Δ and a party waits 2Δ without receiving identical inputs from $t_s + 1$ parties, it can conclude assume that on of two things is true: either the network is not synchronous or there was not honest preagreement. So, all honest parties who did not terminate already, will then send a message that this is the case and try to collect $t_s + 1$ such messages and $t_a + 1$ signed inputs on a bit b. This is to demonstrate two things: First the $t_a + 1$ signatures on b demonstrate that if the network is asynchronous, then the bit is the input of at least one honest party. On the other hand, if the network is synchronous at least on honest party signed a message saying that they detected that either the network is asynchronous or that it is synchronous but both bits are valid inputs because there was not pre-agreement.

A final concern is that if most honest parties give output using the first rule without making this information propagate, then there might not be enough honest parties left to trigger the second rule. Because of this we make everyone forward a justified output when they terminate. To make communication $O(n^2\lambda)$, the output is justified by a threshold signature which combines shares from $t_s + 1$ parties directly on the bit, or alternatively only $t_a + 1$ directly on the bit and $t_s + 1$ shares from parties who claim that either there was not honest pre-agreement or that the network is asynchronous.

Input Validation Gadget Π_{IVG}

- On input b_i , P_i records current time τ_i , multicasts (b_i , ASYNC) with a signature, and shares of threshold signature schemes with thresholds $t_a + 1$ and $t_s + 1$.
- Upon receiving (b, ASYNC) for some bit b from $t_s + 1$ distinct parties, P_i combines the shares of the $t_s + 1$ threshold signature scheme, multicasts (b, ASYNCOUTPUT) along with the threshold signature and gives output b.
- At time $\tau_i + 2\Delta$ if P_i did not yet give output, it multicasts ASYNCORBOTHVALID.
- Upon receiving ASYNCORBOTHVALID from $t_s + 1$ distinct parties and (b, ASYNC) for some bit b from $t_a + 1$ distinct parties, P_i combines the shares of the $t_s + 1$ threshold signature scheme on ASYNCORBOTHVALID and the $t_a + 1$ threshold signature scheme on (b, ASYNC) and multicasts (b, SYNCOUTPUT) along with the threshold signatures and gives output b.

Fig. 5. An input validation gadget reducing BA to VBA.

Note that if there is at most t_a corrupted parties, then the majority bit among the remaining $n - t_a > 2t_s$ honest parties will give a justifier for that bit which travels at network speed. In other words: the protocol is optimistically responsive.

Lemma 9. The protocol Π_{IVG} given in Fig. 5 is a secure IVG as defined in Definition 11.

Proof. We consider the synchronous and asynchronous case separately. In asynchrony we first consider justified validity. Since any justified output is includes at least $t_a + 1$ identical signed inputs. At least one honest party had this as input. For liveness observer that eventually $n - t_a > 2t_s$ inputs propagate and the majority bit among them is signed by at least $t_s + 1$. This also gives optimistic responsiveness whenever corruption is bounded by t_a .

In synchrony, the liveness holds because $n - t_s > t_s + t_a > t_a$, so the majority bit among honest parties form a certificate within 3Δ , unless some parties gave optimistic output, which in that case propagates within δ . Validity when following the optimistic rule holds as $t_s + 1 > t_s$. Validity of the alternative path holds because some honest party signed AsyNCORBOTHVALID. This party waited long enough to conclude that it heard all honest inputs, and since $n - t_s \ge t_s + 1$ and it did not yet give output, there could not have been honest pre-agreement. So the justified validity property is an empty statement.

Corollary 1. The protocol Π_{BA} given in Fig. 4 is a secure binary agreement protocol as defined in Definition 7.

Proof. Follows directly from the security definitions shown to be satisfied by Π_{VBA} and Π_{IVG} .

5 Agreement on a Core Set

To construct ACS for the network agnostic setting, we now take a detour in order to provide a more general result. Namely, using protocols satisfying the definitions of reliable broadcast and common coin in a black box fashion, we first present a protocol Π_{ACS} for asynchronous ACS based on ideas from DAG-Rider. We will define the protocol for a set of parties $\mathcal{P} =$ $\{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ of which we assume at most t < n/2 are adaptively corrupted. Then, when we subsequently instantiate the reliable broadcast using the protocol from Section 3 and the common coin as we did for the BA construction, the resulting protocol is secure in the network agnostic setting with thresholds $n > t_a + 2t_a$.

DAG-Rider with Honest Majority from Reliable Broadcast and Common Coin. We provide some background on DAG-Rider and intuition for how the proof can be adapted to honest majority. The full construction and proofs appear in Appendix A

First, the DAG-Rider protocol is phrased as an atomic broadcast, rather than as ACS. At a high level: atomic broadcast implements a growing ledger of entries where parties up to a prefix relation agree on the ordering of these records. DAG-Rider provides such a ledger by letting parties include entries in every block, and then when a block is committed impose a deterministic ordering on all outstanding entries in the transitive closure of blocks that can be reached via pointers from the selected block (which we will refer to as the causal past of the block). To get a single shot ACS from this, one can view DAG-Rider as a way to select a single block, then run it with empty blocks except for including a set of n-t inputs in the first round, and output the set of inputs in the (causal past of the) first selected block.

In the original protocol, this block selection process is run in a pipelined fashion which for instance involves dealing with consistency between the views of parties who commit different leaders in different waves. While the protocol is remarkably simple, pipelined commits results in monolithic proof. We choose to disentangle this pipelined presentation into a roughly equivalent single shot ACS. This allows deconstructing the wave-based protocol into logical units for which simpler security properties can be proved separately. The protocol still works based on the same principles, but some care and revised proof techniques have to be applied to show that it works for n > 2t. We attempt to give the intuition for DAG-Rider only needing honest majority in a nutshell: the DAG is built in waves consisting of 4 rounds where parties add a block pointing to n-t blocks from the previous round. After completing a wave (by receiving n-t blocks from its fourth round), a coin is flipped to elect a leader. If a party's local view of the DAG has n-t blocks in the last round that include in their causal past the leader's block in the first round, then this party commits the leader's block. Since all parties might not commit in the same wave, consistency is enforced by noticing that it is impossible to completely miss a leader that someone else might have committed, since they would have been included in the causal past of n-t blocks, of which you have seen at least n-2t. Had the blocks just been sent through point to point channels, then you would normally need to use the n > 3tthreshold to argue that they have at least t + 1 i.e. one honest party in common. But since the blocks are sent through reliable broadcast it does not matter if they are Byzantine, and a threshold of n > 2t would suffice.

By inspecting the proofs of Keidar et al. it becomes clear that this holds for almost all argued properties. The crucial exception is that they rely on n > 3t to invoke a result by Dolev and Gafni [DG16] which is used to argue liveness. Namely, that at the end of each wave before the coin is flipped there is always at least n - t parties that would be committed by all parties if they were chosen as the leader. Keidar et al. observe that the first 3 rounds of a wave matches the protocol logic in the "get core" protocol of Dolev and Gafni. This protocol is in turn an adaptation of a protocol by Attiya and Welch [AW04] from a setting with t < n/2 crash faults to be Byzantine fault tolerant when n > 3t. To claim resilience against Byzantine corruption, Dolev and Gafni have to rely on a different proof technique.

We show that the proof technique of Attiya and Welch can be translated to the Byzantine setting (using the Dolev and Gafni protocol or equivalently 3 rounds of a DAG-Rider wave) while keeping the n > 2t threshold assuming black box use of reliable broadcast. Intuitively, this is not terribly surprising. The protocol follows a pattern where (1) all messages are sent through reliable broadcast, (2) all messages include pointers to the information that prompted them to be sent in accordance with some activation rule, and (3) the messages are verified to be consistent with an honest execution of said activation rule and otherwise ignored. (1) removes the ability of equivocating from Byzantine parties, (2) and (3) imply that the whenever their messages are received (in the sense that they are not rejected as invalid) they are consistent with the input output behaviour of an honest party. It follows that a Byzantine faulty party is for most purposes equivalent to a crash faulty party.⁹

With this insight we construct an ACS protocol that is information theoretically secure against an adversary which adaptively corrupts up to t < n/2 parties in an asynchronous network from black box use of reliable broadcast and common coin. It is additionally expected constant round and responsive in the sense that when the reliable broadcast delivers messages in $O(\delta)$ time, then it terminates in expected $O(\delta)$ time.

The full construction and security proof appears in Appendix A.

⁹ A notable exception is that a Byzantine party might claim to not have received some information, and that it cannot be trusted for validity guarantees that cannot be externally verified as discussed in [DG16].

5.1 Network Agnostic ACS

After this aside, we reenter the network agnostic setting and assume thresholds $n > t_a + 2t_s$. The ACS presented in Appendix A runs in expected O(1) rounds of responsive communication, which since we are dealing with subprotocols has to be further qualified: In an infinite execution, let the maximum time it takes for reliable broadcast protocol $\Pi_{\rm RB}$ to deliver the message of an honest sender to an honest receiver or between delivering a message to two different honest receivers be $\Delta_{\rm RB}$ (see Remark 1), and the maximum duration between the common coin $\Pi_{\rm CC}$ getting input from all honest parties until it gives output to all honest parties be $\Delta_{\rm cc}$, then $\Pi_{\rm ACS}(\Pi_{\rm RB}, \Pi_{\rm CC})$ terminates within time $O(\Delta_{\rm max(RB,CC)}) = \max(\Delta_{\rm RB}, \Delta_{\rm CC})$.

In particular this provides a network agnostic ACS which when Π_{RB} is instantiated with the protocol from Fig. 1 and Π_{CC} is instantiated with common coin based on threshold signatures, Π_{ACS} terminates within expected time $O(\delta)$ when at most t_a parties are corrupted and otherwise within $O(\Delta)$.

Theorem 3. In the network setting with $n > t_a + 2t_s$ the protocol Π_{ACS} given in Fig. 11 is a secure ACS with a t_s -Large Core according to Definition 9 when instantiated with the reliable broadcast protocol in Fig. 1. It is expected constant time and optimistically responsive against t_a corruptions.

Proof. The generic Π_{ACS} protocol makes black box use of a protocol satisfying Definition 6, a common coin (which we assume instantiated from threshold signatures) and the assumption that n > 2t. The reliable broadcast in Fig. 1 satisfies Definition 6 for the network agnostic setting where $\Delta_{\max(RB,CC)}$ is bounded by Δ in synchronous executions and by δ when at most t_a parties are corrupted. The threshold t can be replaced with t_s which satisfies a strictly stronger constraint.

An MPC protocol with the same properties follows from standard technique using threshold fully-homomorphic encryption (FHE) and non-interactive zero-knowledge (NIZK) proofs. See for instance Blum et al. [BKLZL20]. Parties can encrypt their input and use the ciphertext with a proof of plaintext knowledge as inputs to the ACS. Then compute the function of the encrypted ciphertexts and finally distribute decryption shares of the output. Crucially, this only requires using threshold FHE with threshold t + 1 when there are no more then t < n/2 corruptions. This only adds a single round of responsive communication, so in the network agnostic setting we can use threshold $t_s + 1$ to get constant time optimistically responsive network agnostic MPC.

5.2 On the Number of Honest Inputs in the Core Set

As discussed in Section 1.3, Constantinescu et al. [CGWW24] require a stronger security definition for network agnostic ACS than what we define, namely that in synchronous executions it satisfies the Honest Core property, i.e. it contains all honest inputs. This property is inherently infeasible for any optimistically responsive protocol if the optimistic condition does not require all parties to be honest. (At which point ACS with Honest Core is trivially implemented by waiting for all parties to send an input.)

Theorem 4 (Informal). If a protocol for ACS can terminate responsively when t parties are corrupted, then it may output a core that does not contain the input of t honest parties in a synchronous execution.

Proof. Consider a protocol for ACS that is guaranteed to terminate responsively when no more than t parties are corrupted. Observe that the adversary can simulate the optimistic condition and an arbitrary low latency. Concretely, it may choose to obey some concrete network delay $\delta < \Delta$, corrupt a set of t parties A but let them follow to the protocol, and partition a set of t honest B parties from the rest of the network for a duration of δ . Finally, simulate an execution with a much shorter network delay $\delta^* << \delta$ among the remaining honest parties C and the corrupted parties A.

Now, since the protocol should terminate responsively in the face of up to t corruptions, and the t corrupted parties A act honestly, the honest parties B that have not been partitioned from the network must terminate within a duration depending only on δ^* and in particular before δ for any sufficiently small δ^* . Since the parties in B who terminated early were partitioned from C, their output cannot include the inputs of C.

We conclude that in general: if a protocol for ACS can terminate responsively in the face of t corruptions, it can terminate without the inputs of t honest parties.

An ACS with Honest Core and Minimal Latency. We sketch a simple gadget that ensures an Honest Core in synchronous executions while only sacrificing responsiveness in an initial input round. We will assume that parties receive input within a duration of Δ . Then parties initially multicast their inputs, and collects inputs until 2Δ have passed and at least $n - t_s$ inputs have been collected. At this point they use the collected set as input to Π_{ACS} . In a synchronous execution, all honest parties have received the inputs of all honest parties. And since at least one honest party makes it into the core, another core which in synchronous executions satisfies Honest Core can be formed by taking the union of sets of inputs included in the original core. The resulting ACS is constant time as before, but only optimistically responsive after the initial input round.

An Optimistically Responsive ACS with Honest Core Majority. Since Honest Core is impossible to get with a responsive ACS, but can be achieved by one round of unresponsive communication, it raises the question: what meaningful properties can we ensure from an optimistically responsive ACS beyond the current guarantee of a t_s -Large Core? Concretely, from Theorem 4 and our protocol being responsive when $t \leq t_a$, we conclude that it must allow t_a honest inputs not making it into the core. But currently it allows that t_s honest inputs are left out of the core, which in the worst case only guarantees that the core contains a single honest input.

The most obvious approach is to improve the unresponsive gadget with the natural optimistic condition: If we start the process of waiting and collecting inputs as above, but short circuit the process whenever $n - t_a$ inputs are collected before the timeout, then the gadget and overall construction is responsive when $t \leq t_a$ and in this case the ACS will have a core of size at least $n - t_a$. If the execution is synchronous, then some honest part had their input set included in the core, and either provided an input set with all honest inputs or

provided an input set of size at least $n-t_a$. In either case the input set contains at least t_s+1 honest contributions, which means that union of all input sets in the core consists primarily of honest inputs. In asynchrony the process results in a core of size at least $n-t_s$ as before, but since at most t_a parties are corrupted, the majority of the inputs in the selected core set are honest.

We will say that a protocol Π is an ACS protocol with an Honest Core Majority if for any set S obtained as output of Π by an honest party, majority of the elements in S are inputs of honest parties. Observe that this corresponds the setting in optimally resilient asynchronous MPC with n = 3t + 1, where the best guarantee one can hope for is to establish a core of n - t inputs which matches Honest Core Majority. To demonstrate it usefulness, notice that it can be used to solve BA by giving bits as input to an ACS with Honest Core Majority and defining the output to be the most prevalent bit in the core. We leave further investigations into the usefulness of the Honest Core Majority property and its relation to Honest Core as future work.

References

ABKL22.	Andreea B. Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. State machine replication under changing network conditions. In Shweta Agrawal and Dongdai Lin, editors, ASIACRYPT 2022,
	Part I, volume 13791 of LNCS, pages 681–710, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.
ACC22.	Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly-secure synchronous MPC with asynchronous fallback guarantees. In Alessia Milani and Philipp Woelfel, editors, <i>41st ACM PODC</i> , pages 92–102, Salerno, Italy, July 25–29, 2022. ACM.
$AMN^+20.$	Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In 2020 IEEE Symposium on Security and Privacy, pages 106–118, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.
AW04.	Hagit Attiya and Jennifer L. Welch. Distributed computing - fundamentals, simulations, and advanced topics (2. ed.). Wiley series on parallel and distributed computing. Wiley, 2004.
BCLZL23.	Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. Network-agnostic security comes (almost) for free in DKG and MPC. In Helena Handschuh and Anna Lysyanskaya, editors, <i>CRYPTO 2023, Part I</i> , volume 14081 of <i>LNCS</i> , pages 71–106, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.
BKL19.	Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, <i>TCC 2019, Part I</i> , volume 11891 of <i>LNCS</i> , pages 131–150, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.
BKL21.	Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, ASIACRYPT 2021, Part II, volume 13091 of LNCS, pages 547–572, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.
BKLZL20.	Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous by zantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, <i>TCC 2020, Part I</i> , volume 12550 of <i>LNCS</i> , pages 353–380, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland.
Bra87. BZL20.	Gabriel Bracha. Asynchronous byzantine agreement protocols. <i>Inf. Comput.</i> , 75(2):130–143, nov 1987. Erica Blum, Chen-Da Liu Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, <i>CRYPTO 2020, Part II</i> , volume 12171 of <i>LNCS</i> , pages 707–731, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.
CGWW24.	Andrei Constantinescu, Diana Ghinea, Roger Wattenhofer, and Floris Westermann. Convex consensus with asynchronous fallback. In <i>DISC</i> , volume 319 of <i>LIPIcs</i> , pages 15:1–15:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

- CKPS01. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.
- CKS00. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, 19th ACM PODC, pages 123–132, Portland, OR, USA, July 16–19, 2000. ACM.
- Coh16. Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 183–207, Taipei, Taiwan, March 6–9, 2016. Springer Berlin Heidelberg, Germany.
- CR93. Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In 25th ACM STOC, pages 42–51, San Diego, CA, USA, May 16–18, 1993. ACM Press.
- DE24. Giovanni Deligios and Mose Mizrahi Erbes. Closing the efficiency gap between synchronous and networkagnostic consensus. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 432–461, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.
- DG16. Danny Dolev and Eli Gafni. Some garbage in some garbage out: Asynchronous t-byzantine as asynchronous benign t-resilient system with fixed t-trojan-horse inputs. *CoRR*, abs/1607.01210, 2016.
- DHLZ21. Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 623–653, Raleigh, NC, USA, November 8–11, 2021. Springer, Cham, Switzerland.
- DY81. Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In 22nd FOCS, pages 350–357, Nashville, TN, USA, October 28–30, 1981. IEEE Computer Society Press.
- DYMM⁺20. Thomas Dinsdale-Young, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Afgjort: A partially synchronous finality layer for blockchains. In Clemente Galdi and Vladimir Kolesnikov, editors, SCN 20, volume 12238 of LNCS, pages 24–44, Amalfi, Italy, September 14–16, 2020. Springer, Cham, Switzerland.
- FM97. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. SIAM J. Comput., 26(4):873–933, 1997.
- GGLZ22. Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. Round-optimal byzantine agreement. In Orr Dunkelman and Stefan Dziembowski, editors, EUROCRYPT 2022, Part I, volume 13275 of LNCS, pages 96–119, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.
- GLZW22. Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In Alessia Milani and Philipp Woelfel, editors, 41st ACM PODC, pages 70–80, Salerno, Italy, July 25–29, 2022. ACM.
- HMW18. Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. CoRR, abs/1805.04548, 2018.
- Kam25. Simon Holmgaard Kamp. A new way to achieve round-efficient asynchronous byzantine agreement. 2025.
- KKNS21. Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, 40th ACM PODC, pages 165–175, Virtual Event, Italy, July 26–30, 2021. ACM.
- KNTT22. Simon Holmgaard Kamp, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. Enig: Player replaceable finality layers with optimal validity. Cryptology ePrint Archive, Report 2022/201, 2022.
- LN24. Julian Loss and Jesper Buus Nielsen. Early stopping for any number of corruptions. In Marc Joye and Gregor Leander, editors, EUROCRYPT 2024, Part III, volume 14653 of LNCS, pages 457–488, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.
- MR21. Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In Giovanni Vigna and Elaine Shi, editors, ACM CCS 2021, pages 1686–1699, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.
- PS18. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, EUROCRYPT 2018, Part II, volume 10821 of LNCS, pages 3–33, Tel Aviv, Israel, April 29 May 3, 2018. Springer, Cham, Switzerland.

A Constant Time Honest Majority Asynchronous ACS from RB and CC

The implementation of Π_{ACS} is defined via a series of protocols where each party has an input $B_i \in \{0, 1\}^*$, which we will call a block below. A *block set* is a set of pairs $U = \{(\mathsf{P}_j, B_j)\}_{\mathsf{P}_j \in \mathsf{P}}$, where $P \subseteq \mathcal{P}$ and $|P| \ge n - t$. We think of $(\mathsf{P}_j, B_j) \in U$ as P_j having input B_j . We offer the reader the intuition that the flow of protocols presented in this section can be viewed as a single shot version of the DAG-Rider protocol [KKNS21], but with the crucial difference being that we show that only honest majority is needed.

Most of the primitives are to the best of our knowledge not commonly studied. So, in order to present each primitive with a clear understanding of its building blocks, we present the construction from the ground up. To assist the reader with an intuition of where we are headed, we provide the following roadmap of the steps we take to reduce ACS to simpler primitives:

- Agreement on a Core Set (Appendix A.7): The end goal is to construct ACS, which allows parties to reach agreement on a set of inputs. We achieve this by letting each party send their inputs, then collect a set of n t inputs and finally, use select block to pick one of these sets.
- Select Block (Appendix A.6): Select block takes from each party a single input, which satisfies an input justifier. It outputs one of these inputs with agreement. We implement this from a weaker version which only has graded agreement.
- **Graded Select Block (Appendix A.5):** The graded version lets each party obtain a justified output block with a grade. If a justified output has grade 2, then all outputs have grade 1 or 2 and the justifier can be propagated as a proof of termination being safe. This is achieved by running graded gather on the blocks and interpreting the output using a common coin.
- **Graded Gather (Appendix A.4):** Graded gather takes inputs from all parties, and to each party outputs two sets U and T, each with at least n t inputs. All of the sets have a large common core in the sense that their intersection is of size at least n t, additionally each T set is a subset of the intersection of all U sets. It is implemented by running gather on the inputs, multicasting the outputs and taking the union and intersection of n t outputs of gather.
- **Gather (Appendix A.3):** Gather takes inputs from all parties and lets them each obtain a set of at least n t inputs, with the guarantee that all these sets have a common core of at least n t inputs. Our core contribution is to show that this can be implemented in asynchrony with n > 2t from causal cast.
- Causal Cast (Appendix A.1): Causal cast is a framework used to impose a causal order on consistent messages by sending them through reliable broadcast with references to other messages.

A.1 Causal Cast

We first present a framework for describing protocols for DAG-style protocols (cf. [KKNS21]) in a modular way and in combination with non-DAG style protocols. It can also be seen as a generalization of the CO_Send protocol by Dolev and Gafni [DG16]. In a DAG-style protocol all messages m are reliably broadcast and they point to the other reliably broadcast messages they were computed from. Therefore the receiver can recompute and check the message m. In fact, the message m never has to be sent, the receiver can compute it itself. This allows to compress the communication complexity. All that needs to be reliably broadcast are the pointers to the messages used to compute m. This implements reliable, causal communication against a Byzantine adversary. We will therefore call our system below causal cast.

Each message m to be causal cast will have a message identifier mid. We let mid encode one of three *types* (free-choice, computed and constant) which partitions the set of message identifiers into sets with the following semantics:

- **Free-Choice Message:** Each free-choice identifier specifies a sender $\mathsf{P}^{\mathsf{mid}}$ and a justifier J^{mid} . Messages with a free-choice mid are used to introduce information from the local view of a party to the causal cast protocol.
- **Computed Message:** Each computed message identifier specifies a sender $\mathsf{P}^{\mathsf{mid}}$ and a next message function $\mathsf{NextMessage}^{\mathsf{mid}}$. This function is PPT and takes as input a set of pairs $M = \{(\mathsf{mid}_j, m_j)\}_{j=1}^{\ell}$ and outputs $m = \mathsf{NextMessage}^{\mathsf{mid}}(M)$, where $m = \bot$ indicates that M is not a valid set of inputs for computing the message for mid. When a party sends a computed message, it only needs to transmit the identity of the messages that made it send the message. The remaining parties will locally simulate what the sender would have said. This saves communication and guarantees that computed messages from Byzantine parties are consistent with messages from an honest party.
- **Constant Message:** The constant identifiers are a tool to define that some values on which the parties already agree have been "delivered" and thus justified. We will for instance use this to introduce the output from a common coin into the framework, such that computed messages can be based on it.

The system guarantees liveness, agreement on all messages, and that all messages are valid inputs, valid outputs of a leader election, or computed correctly from other valid messages.

Definition 12 (Causal Cast). A protocol for n parties P_1, \ldots, P_n is called a causal cast if it has the following properties.

- **Free-Choice Send:** A party P_i can have input (CAUCAST-SEND, mid, m) where mid is a free choice identifier $P_i = P^{mid}$ and $J^{mid}(m) = \top$ at P^{mid} at the time of input.
- **Computed-Message Send:** A party P_i can have input (CAUCAST-SEND, mid, m, mid₁,..., mid_{ℓ}), where mid is a computed-message identifier, $P_i = P^{mid}$, P_i earlier gave outputs (CAUCAST-DEL, mid_j, m_j) for $j = 1, ..., \ell$, and $\perp \neq m =$ NextMessage^{mid}((mid₁, m_1), ..., (mid_{ℓ}, m_{ℓ})). We let J^{mid} be the predicate that is satisfied when NextMessage^{mid}(M) $\neq \perp$, i.e. as soon as a party sees the referenced messages as delivered and is able to successfully recompute the message.

- **Constant Send:** A party P_i can have input (CAUCAST-SEND, mid, m) where mid is a constant identifier. In that case it is guaranteed that all honest parties eventually have the same input (CAUCAST-SEND, mid, m). We let J^{mid} be the trivial justifier that the message was received as local input.
- **Free-Choice Validity:** A party P_i can have output (CAUCAST-DEL, mid, m), where mid is a free-choice identifier. It then holds that $J^{mid}(m) = \top$ at P_i at the time of output. Furthermore, if $P_i = P^{mid}$ is honest, then P_i had input (CAUCAST-SEND, mid, m).
- **Computed-Message Validity:** A party P_i can have output (CAUCAST-DEL, mid, m, mid₁,..., mid_l), where mid is a computed-message identifier. In that case P_i earlier gave outputs (CAUCAST-DEL, mid_j, m_j ,...) for j = 1, ..., l, and $\perp \neq m = \text{NextMessage}^{\text{mid}}((\text{mid}_1, m_1), ..., (\text{mid}_l, m_l)).$
- **Constant Validity:** A party P_i can have output (CAUCAST-DEL, mid, m). In that case it immediately before had input (CAUCAST-SEND, mid, m).
- **Liveness:** If an honest party P_i had input (CAUCAST-SEND, mid, ...) or some honest party had output (CAUCAST-DEL, mid, ...) and all honest parties are running the system, then eventually all honest parties have output (CAUCAST-DEL, mid, ...).
- **Agreement:** For all possible justified outputs (CAUCAST-DEL, mid, m, ...) and (CAUCAST-DEL, mid, m', ...) it holds that m' = m.

Below is a protocol implementing causal cast from black box use or reliable broadcast.

Protocol Π_{Causal} in the view of party P_i .

- **Free-Choice Send:** On input (CAUCAST-SEND, mid, m) at P_i where mid is a free choice P_i inputs (mid, m) to Π_{RB} ,
- **Computed-Message Send:** On input (CAUCAST-SEND, mid, m, mid₁,..., mid_l) at P_i , where mid is a computed-message identifier, P_i inputs (mid, (mid₁,..., mid_l)) to Π_{RB} .

Free-Choice Deliver: On Π_{RB} delivering (mid, m) where mid is a free-choice identifier, wait until $J^{\text{mid}}(m) = \top$ (possibly waiting forever if this never happens) and then output (CAUCAST-DEL, mid, m). Computed-Message Deliver: On Π_{RB} delivering (mid, (mid_1, ..., mid_{\ell})) wait until outputs (CAUCAST-DEL, mid_j, m_j) were given for $j = 1, \ldots, \ell$ (possibly waiting forever if this never happens), compute $m = \text{NextMessage}^{\text{mid}}((\text{mid}_1, m_1), \ldots, (\text{mid}_{\ell}, m_{\ell}))$, and output (CAUCAST-DEL, mid, m, mid_1, \ldots, mid_{\ell}) if $m \neq \bot$.

Constant Deliver: On input (CAUCAST-SEND, mid, m) where mid is a constant identifier, output (CAUCAST-DEL, mid, m).

Fig. 6. Protocol Π_{CAUSAL} implementing Causal Cast from reliable broadcast and common coin.

Lemma 10. The protocol Π_{CAUSAL} in Fig. 6 is a causal cast protocol cf. Definition 12 assuming the same constant inputs are given across honest parties, that Π_{RB} satisfy Definition 6.

Proof. As long as honest parties give the same constant inputs, agreement holds by agreement of Definition 6 and the NextMessage function being deterministic. All validity properties holds by the validity of Π_{RB} and design of Π_{CAUSAL} .

The protocols presented in Appendix A follow a common pattern: The protocols are chained together with the input of each being computed from the output of a previous protocol and only the first in the chain being given free-choice inputs satisfying some J_{IN} predicate.

Definition 13 (Justified Causal Cast Protocols). A Justified Causal Cast protocol is a special case of a Justified protocol as defined Definition 5, where all message are sent through Causal Cast. We let the justifier J^{mid} of a message m at P_i be that (CAUCAST-DEL, mid, m, \ldots) was output by P_i . The input m of each party, identified by mid_{IN} , is a message with $J_{\text{IN}}(m) = J^{\text{mid}_{\text{IN}}}(m) = \top$. The output m of each party, identified by mid_{OUT} , can be sent as a computed message in which case $J_{\text{OUT}}(m) = J^{\text{mid}_{\text{OUT}}}(m) = \top$. In particular this means that ouputs are computed as a function, NextMessage^{mid_{OUT}}, of previously Causal Cast messages and can be forwarded as a computed message.

A.2 Causal Cast Complexity

We address communication complexity. We represent a session identifier sid with κ bits as we can always hash the session identifiers. We let all parties P_i number the messages they send using a counter $c_i = 1, 2, \ldots$. Then message identifiers can be of the form (sid, i, c_i) . We do not need to send sid along with all mid as it is the same for all mid in the protocol. Using that $n, n \in \mathsf{poly}(\lambda)$ and that c_i can become at most polynomially large in a poly-time run of the system each mid can be represented in $\log(\kappa)$ extra bits. We can therefore represent $(\mathsf{mid}, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$ as $\kappa + \ell \log(\kappa)$ bits. So, the communication complexity is that of reliably broadcasting all free-choice messages m plus the complexity of reliably broadcasting $\kappa + \ell \log(\lambda)$ per computed message. In many cases a computed message is computed from n - toutputs of n possible messages with session identifiers $\mathsf{sid}_1, \ldots, \mathsf{sid}_n$ known by all parties. In these cases we can send an n-bit vector indicating the n - t session identifiers to use. Then the communication is only $\mathcal{O}(\lambda + n) = \mathcal{O}(\lambda)$ bits per computed message. We return to this when analysing the complexity of concrete protocols.

Definition 14 (Complexity). We say that a protocol Π using causal cast has (expected communication) complexity

$$\mathcal{O}(c_1 \operatorname{IN} + c_2 \operatorname{RB} + c_3 \operatorname{RB}_{\#} + c_4)$$

if the following holds in expectation, using the above methods for compression, and under \mathcal{O} : c_1 is the total number of bits that the protocol needs to RB^{10} as inputs, c_2 is the total number of bits that the protocol needs to RB as intermediary values and outputs, c_3 is the number of RB instances run, and c_4 is the total number of bits sent otherwise. Notice that we count all messages sent by all parties.

The reason why we single out the complexity of inputs is that when inputs are given as computed messages, c_1 IN only needs to accommodate the description of the set of messages used to compute the inputs.

 $^{^{10}}$ I.e., the sum of the length of messages input to a RB.

A.3 Justified Gather

We describe and analyse our Justified Gather protocol. The Gather primitive says that each party P_i has a block set U_i as output and there is a large common core, i.e., a set U of size n-t which is a subset of each U_i . So all parties to some extend agree on a large set U, but some P_i might have extra elements in U_i and they do not know which are the extra ones.

Definition 15 (Justified Gather). A protocol for n parties P_1, \ldots, P_n . There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at P_i at the time of input.

- **Liveness:** If all honest parties start running the protocol with a J_{IN} -justified input then eventually all honest parties have a J_{OUT} -justified output.
- **Justified Blocks:** For all possible justified outputs U and all (potentially corrupt) P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that $J_{\mathrm{IN}}(B_i) = \top$.
- **Validity:** For all possible justified outputs U and all honest P_i and all $(P_i, B_i) \in U$ it holds that P_i had input B_i .
- **Agreement:** For all possible justified outputs U and U' and all $(\mathsf{P}_i, B_i) \in U$ and $(\mathsf{P}_i, B'_i) \in U'$ it holds that $B_i = B'_i$.

Large Core: For all possible justified outputs (U^1, \ldots, U^m) it holds that $|\bigcap_{k=1}^m U^k| \ge n-t$.

We present a Justified Gather protocol, Π_{GATHER} , in Fig. 7. The protocol follows the structure of the get-core protocol presented by Attiya and Welch in [AW04] and attributed to Gafni. The get-core protocol tolerates crash failures of up to half of the parties and works as follows: All parties gossip an input (U^0) and then in two rounds gather sets of inputs from n-t parties, take the union $(U^1 \text{ and } U^2)$ and gossips it. When taking the union of the U^2 sets it can be shown that all resulting sets (U^3) have a common core of size n-t. The proof goes by arguing that the U_i^1 set of some party, P_i must be included in the majority of U^2 sets and thus in all U^3 sets. The get-core protocol was adapted to the Byzantine setting by Dolev and Gafni in [DG16] using an abstraction similar to Causal Cast. However, the proof relies on an honest supermajority sending U sets. We stress that the proof of Lemma 11 implies that the protocol presented in [DG16] would also be secure against a minority of Byzantine parties when given a RB functionality.

We show that Π_{GATHER} is a Justified Gather protocol. The proof largely follows the idea from [AW04] of describing a table and counting entries with ones. While they only consider crash failures, we allow Byzantine corruptions. However, since all the messages are justified and sent through Causal Cast, the adversary must either follow the protocol or stay silent. This allows the proof from [AW04] to go through with the original combinatorial argument, but a slightly different interpretation of the table. Dolev and Gafni, who previously adapted the protocol to tolerate Byzantine corruptions in [DG16], altered the table to only have a row and column for each honest party, which gives a simpler proof but also means that it needs an honest supermajority to go through.

Lemma 11. If t < n/2 then Π_{GATHER} is a Justified Gather. If $\beta = \sum_{i=1}^{n} |B_i|$ then it has complexity $\mathcal{O}(\beta \operatorname{IN} + n \operatorname{RB}_{\#} + n^2 \operatorname{RB})$.

Protocol Π_{Gather} .

- 1. The input of P_i is B_i with $J_{\text{IN}}(B_i) = \top$. Party P_i lets $U_i^0 = \{(\mathsf{P}_i, B_i)\}$. The singleton set is justified by B_i satisfying J_{IN} .
- 2. For $r \in = 1, 2, 3$ Party P_i causal casts U_i^{r-1a} and collects incoming U_j^{r-1} from parties P_j , lets P_i^r be the set of P_j it heard from, waits until $|P_i^r| \ge n-t$, and lets

$$U_i^r = \bigcup_{\mathsf{P}_j \in P_i^r} U_j^{r-1} \; .$$

The set is justified by being computed from the set P_i^r where $|P_i^r| \ge n - t$.

3. Party P_i outputs U_i^3 .

^a Whether U_i^{r-1} is causal cast as a free-choice or computed message is determined by its justifier.

Fig. 7. Protocol for finding a common core corresponding to the first 3 rounds of a DAG-Rider wave.

Proof. We first address the complexity. In the first round n parties each causal cast their input, possibly as a computed message. This contributes $\beta \text{ IN} + n \text{ RB}_{\#}$. Then in a constant number of rounds each party RBs a set described by n bits, adding $\mathcal{O}(n \text{ RB}_{\#} + n^2 \text{ RB})$ and resulting in total of $\mathcal{O}(\beta \text{ IN} + n \text{ RB}_{\#} + n^2 \text{ RB})$.

Liveness, Justified Blocks, Validity, and Agreement are all trivial so we only show Large Core. Initially singleton sets of inputs, U^0 , are sent through causal cast with the only restriction being that the block satisfies J_{IN} . If the message of a corrupted party reaches any honest party then it reaches all honest parties and satisfies J_{IN} . As this is all we require of a message from an honest party, the only way to deviate from the protocol is to not send a valid message. Similarly, in the following rounds $r \in [1; 2]$ the accumulated set U^r is sent through causal cast as a computed message based on the messages from previous round. So, the adversary must choose to either stay silent or send a message identical to what would have been sent by an honest party if they had received messages from the claimed set of parties P^r . The result now follows from the counting argument in [AW04]. We show that there is at least one party P_i , whose U_i^1 set is included in all *possible justified outputs*. This is sufficient for the Large Core property as all justified U^1 sets include n - t input values.

Let T be an n by n table. For each row i: if P_i sends a message U_i^2 which at some point is received by an honest party, then each entry T[i, j] is one if $\mathsf{P}_j \in P_i^2$ and zero otherwise. Alternatively: U_i^2 is never received by an honest party and we let T[i, j] be one if and only if U_j^1 is eventually received by an honest party. Since all rows contain at least n - t ones, there are at least n(n - t) ones in the table. So, at least one of the n columns, k, must contain n - t ones. This means that set of parties whose U^2 set will eventually be received by an honest party and which does not include U_k^1 has size at most t. Call this set of parties S. By Computed-Message Validity, if a message which is never received by an honest party is referenced in a computed message, then that computed message is justified in the view of an honest party. In particular if U_i^2 is never received by an honest party, and P_j sends U_j^3 justified by P_j^3 which points to U_i^2 , then U_j^3 will never be justified in the view of an honest party.¹¹ So, when in the next round any *possible justified output* U^3 is justified by taking the

¹¹ Note that Definition 4 only concerns outputs that can be sent and satisfy J_{0UT} .

union of $n - t U^2$ sets, at least $n - t - |S| \ge n - 2t > 1$ of them will not be in S, i.e. it will contain U_k^1 . Thus U_k^1 is a subset of all *possible justified outputs* U^3 and these all satisfy Large Core.

A.4 Justified Graded Gather

We describe our Justified Graded Gather protocol. This is just a Justified Gather, where each party also has knowledge about the common core. Each P_i outputs a set T_i of size at least n - t which is a subset of the common core U.

Definition 16 (Justified Graded Gather). A protocol for n parties P_1, \ldots, P_n . There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at P_i at the time the input is given.

- **Liveness:** If all honest parties start running the protocol with a J_{IN} -justified input then eventually all honest parties have a J_{OUT} -justified output.
- **Justified Blocks:** For all possible justified outputs (U,T) and all (potentially corrupt) P_i and all $(\mathsf{P}_i, B_i) \in U$ it holds that $J_{\mathrm{IN}}(B_i) = \top$.
- **Sub Core:** For all possible justified outputs $((U^1, T^1), \ldots, (U^m, T^m))$ it holds that $T^i \subseteq \bigcap_{k=1}^m U^k$ for all $i \in [m]$.
- **Validity:** For all possible justified outputs (U,T) and all honest P_i and all $(P_i, B_i) \in U$ it holds that P_i had input B_i .
- **Agreement:** For all possible justified outputs (U,T) and (U',T') and all $(\mathsf{P}_i, B_i) \in S$ and $(\mathsf{P}_i, B_i') \in U'$ it holds that $B_i = B_i'$.
- **Large Sub Core:** For all possible justified outputs $((U^1, T^1), \ldots, (U^m, T^m))$ it holds that $|\bigcap_{k=1}^m T^k| \ge n-t.$

$\Pi_{\mathbf{GradedGather}}$

- 1. The input of P_i is B_i with $J_{\text{IN}}(B_i) = \top$. All parties run Π_{GATHER} with P_i inputting B_i justified by J_{IN} . Let the output of P_i be U'_i .
- 2. Party P_i causal casts U'_i as a computed-message justified by Π_{GATHER} . J_{OUT} and collects justified U'_j from parties P_j , lets P_i be the set of P_j it heard from and waits until $|P_i| \ge n t$.
- 3. Party P_i outputs

$$(U_i, T_i) = \left(\bigcup_{\mathsf{P}_j \in P_i} U'_j, \bigcap_{\mathsf{P}_j \in P_i} U'_j\right)$$

The outputs are justified by being computed as above from justified sets.

Fig. 8. $\Pi_{\text{GradedGather}}$

Lemma 12. If t < n/2 then $\Pi_{GRADEDGATHER}$ is a Justified Graded Gather. If $\beta = \sum_{i=1}^{n} |B_i|$ and $\Pi_{GRADEDGATHER}$ uses Π_{GATHER} from Fig. 7 as sub-protocol, then it has complexity $\mathcal{O}(\beta IN + n^2 RB + n RB_{\#})$. *Proof.* Complexity, Liveness, Justified Blocks, Validity and Agreement follow from the same properties of Justified Gather. We have that $\bigcap_{k=1}^{n} T_k = \bigcap_{k=1}^{n} \bigcap_{\mathsf{P}_j \in P_i} U'_j$, so Large Sub Core follows from Large Core of Justified Gather. Sub Core follows from the below lemma.

Lemma 13 (Sub Core). Let (\cdot, T_i) and (U_i, \cdot) be any possible justified outputs. Then $T_i \subseteq U_i$.

Proof. It is enough to argue that if $(\mathsf{P}_k, B_k) \in T_i$ then $(\mathsf{P}_k, B_k) \in U_i$. If $(\mathsf{P}_k, B_k) \in T_i$ then $\mathsf{P}_k \in U'_j$ for all $\mathsf{P}_j \in P_i$. Since $|P_i| \ge n - t \ge t + 1$ it follows that any party receiving n - t justified sets U'_j will also receive a set U'_j with $\mathsf{P}_k \in U'_j$. Namely, the sets are reliably broadcast so if two parties receive justified U'_j and \hat{U}'_j then $U'_j = \hat{U}'_j$. Since all parties collect n - t justified sets U^ℓ_{\cdot} to justify $U_{\iota} = \bigcup_{\mathsf{P}_j \in P^\ell_{\cdot}} U^\ell_j$ it follows that $(\mathsf{P}_j, B_k) \in U_{\iota}$.

A.5 Justified Graded Block Selection

We now present our graded block selection protocol. Here each party has as input a block B_i and as output a block C_i . The goal is to let C_i be one of the inputs and to agree on C_i . Since corrupted parties can pick their own input and we allow that $C_i = B_i$ for a corrupt P_i we simply define validity by saying that the output should be some justified input. Note that this implies that if there is only one possible justified input, then that will become the only justifiable output. We will not always be able to perfectly agree on the output, instead the output will have a grade $g \in \{0, 1, 2\}$. The grades are never more than 1 apart and if the grade is 2 then there was agreement on C_i . Finally, we want that with some non-zero probability the grade will be 2.

Definition 17 (Justified Graded Block Selection). A protocol for n parties P_1, \ldots, P_n . There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at the time the input B_i is given. The output of the protocol is a C_i justified by J_{OUT} .

Liveness: If all honest parties start running the protocol with a J_{IN} -justified input then eventually all honest parties have a J_{OUT} -justified output.

Justified Output: $J_{IN}(C_i) = \top$ holds for all possible J_{OUT} -justified outputs (C_i, \cdot) .

- **Graded Agreement:** For all possible justified outputs (C_i, g_i) and (C_j, g_j) it holds that $|g_i g_j| \leq 1$. Furthermore, if both $g_i, g_j > 0$ then $C_i = C_j$.
- **Positive Agreement:** There exists $\alpha > 0$ such that with probability at least α negl all possible justified outputs of at least n t parties will have grade $g_i = 2$.
- **Stability:** If there are possible justified outputs (C_i, \cdot) and (C_j, \cdot) with $C_i \neq C_j$ then there exist two justified inputs B_i and B_j with $B_i \neq B_j$.

Lemma 14. If t < n/2 then $\Pi_{GRADEDSELECTBLOCK}$ is a Justified Graded Block Selection. When $\beta = \sum_{i=1}^{n} |B_i|$ and when using $\Pi_{GRADEDGATHER}$ from Fig. 8 as sub-protocol the complexity is $\mathcal{O}(\beta \operatorname{IN} + n^2 \operatorname{RB} + n \operatorname{RB}_{\#} + \operatorname{ELECT})$ where ELECT is the complexity of Π_{CC} .

$\Pi_{\mathbf{GradedSelectBlock}}$

- 1. The input of P_i is B_i with $J_{\mathrm{IN}}(B_i) = \top$.
- 2. The parties run $\Pi_{\text{GRADEDGATHER}}$ with input B_i and input justifier J_{IN} . Let the output of P_i be (U_i, T_i) and causal cast this as a computed message.^{*a*}
- 3. Upon receiving (U_j, T_j) from n t parties $\mathsf{P}_j \in P_i$, the parties use $\Pi_{\rm CC}$ to elect a king P_k , input P_k to causal cast as a constant send.
- 4. Party P_i outputs

$$(C_i, g_i) = \begin{cases} (B_k, 2) & \text{if } \exists (\mathsf{P}_k, B_k) \in T_i \\ (B_k, 1) & \text{if } \exists (\mathsf{P}_k, B_k) \in U_i \setminus T_i \\ (B_i, 0) & \text{if } \nexists (\mathsf{P}_k, \cdot) \in U_i . \end{cases}$$

The output is justified by being computed as above from causal cast messages.

^a The point of this message is not to distribute the sets, but to commit n - t parties to their output of $\Pi_{\text{GRADEDGATHER}}$ before an honest party invokes Π_{CC} .

Fig. 9. $\Pi_{\text{GRADEDSELECTBLOCK}}$

Proof. We start with the complexity. $\Pi_{\text{GRADEDGATHER}}$ has complexity $\mathcal{O}(\beta \text{ IN} + n^2 \text{ RB} + n \text{ RB}_{\#})$. In addition to this $\Pi_{\text{GRADEDSELECTBLOCK}}$ only does one leader election. It has to send no more causal cast information as the justifier for (C_i, g_i) is the justified B_i , the justified (U_i, T_i) , and the justified P_k , which have all been causal cast already. Liveness is straight forward. We argue Justified Output. Let (C_i, g_i) be any justified output. If $g_i = 0$ then by definition $C_i = B_i$ is a justified input. If $g_i > 0$ then $B_i = B_k$ for $(\mathsf{P}_k, C_k) \in U_i$ and therefore C_k is a justified input to the Graded Gather which also used J_{IN} as input justifier. Then use the Justified Blocks property. To argue Graded Agreement let (C_i, g_i) and (C_j, g_j) be any justified outputs. To argue that $|g_i - g_j| \leq 1$ it is sufficient to prove that if $g_i = 2$ then $g_j \neq 0$. So assume that $g_i = 2$. Then $(\mathsf{P}_k, C_i) \in T_k$ for some justified T_k . Therefore, by Sub Core, $(\mathsf{P}_k, C_i) \in U_j$, and therefore $g_j \geq 1$. Assume then that $g_i, g_j > 0$. In that case $(\mathsf{P}_k, C_i) \in U_i$ and $(\mathsf{P}_k, C_i) \in U_i$, so by Agreement of the Graded Gather it follows that $C_i = C_i$. We then argue Positive Agreement for $\alpha = 1/2$. Consider the first honest P_i to start running the leader election. When this happens P_i already received (U_i, T_i) from n-t parties P_i . By Large Sub Core of $\Pi_{\text{GRADEDGATHER}}$ these T_j sets have an intersection of size at least n-t. Since $|\bigcap_{\mathsf{P}_i \in P_i} T_i| \ge n - t > n/2$ it follows that $\mathsf{P}_k \in T_j$ for all $\mathsf{P}_j \in P_i$ with probability at least 1/2 – negl. Whenever this happens, no party in P_i can justify an output with grade less than 2. To argue Stability just note that it holds for both C_i and C_i that they are justified inputs of $\Pi_{\text{GRADEDSELECTBLOCK}}$.

A.6 Justified Block Selection

We now present our (ungraded) block selection protocol. The difference from graded block selection is that all possible justified outputs C_i should be identical.

Definition 18 (Justified Block Selection). A protocol for n parties P_1, \ldots, P_n . There is an input justifier J_{IN} and an output justifier J_{OUT} . All honest P_i have an input B_i for which $J_{IN}(B_i) = \top$ at the time the input was given. The output of the protocol is a block C_i justified by J_{OUT} . **Liveness:** If all honest parties start running the protocol with a J_{IN} -justified input then eventually all honest parties have a J_{OUT} -justified output.

Justified Output: $J_{IN}(C_i) = \top$ holds for all possible J_{OUT} -justified outputs C_i . Agreement: For all possible justified outputs C_i and C_j it holds that $C_i = C_j$.

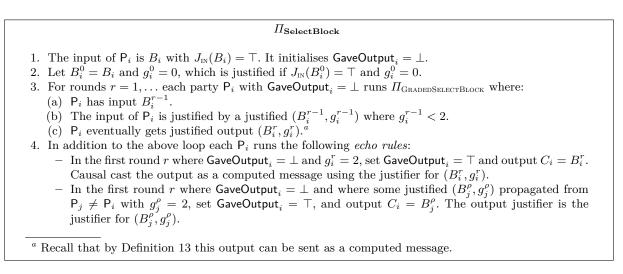


Fig. 10. $\Pi_{\text{SelectBlock}}$

Lemma 15. If t < n/2 then $\Pi_{\text{SELECTBLOCK}}$ is a Justified Select Block Protocol. When $\beta = \sum_{i=1}^{n} |B_i|$ and using the protocol $\Pi_{\text{GRADEDSELECTBLOCK}}$ from Fig. 9 as sub-protocol the complexity is $\mathcal{O}(\beta \text{ IN} + n^2 \text{ RB} + n \text{ RB}_{\#} + \text{ELECT})$ where ELECT is the complexity of Π_{CC} .

Proof. We start with the complexity. The first run of $\Pi_{\text{GRADEDSELECTBLOCK}}$ has complexity $\mathcal{O}(\beta \operatorname{IN} + n^2 \operatorname{RB} + n \operatorname{RB}_{\#} + \operatorname{ELECT})$, and each following run has complexity $\mathcal{O}(n^2 \text{ RB} + n \text{ RB}_{\#} + \text{ELECT})$, where we ignore the IN component as the size of the message identifiers needed to send the outputs as computed messages (n bits for each) is dominated by other costs. Besides this the protocol only causal casts computed values for which the receiver knows the message identifier, so there is no more information to causal cast. The protocol terminates in expected $\mathcal{O}(1)$ rounds as argued below. This gives the desired complexity. Liveness follows from Positive Agreement: at some point all possible justified outputs from at least n-t parties of the r^{th} iteration of $\Pi_{\text{GRADEDSELECTBLOCK}}$ will have $g_i^r = 2$. These parties cannot give justified input to $\Pi_{\text{GRADEDSELECTBLOCK}}$ in round r+1, which means that it will deadlock. Additionally one of these parties is honest and will have $g_i^r = 2$, and then the protocol will eventually terminate by construction of the echo rules. Justified Outputs is clear by the Justified Output rule of $\Pi_{\text{GRADEDSELECTBLOCK}}$ which maintains that $J_{\text{IN}}(B_i^r) = \top$ for all r. We then argue Agreement. Assume that some P_i outputs C_i . Then it saw a justified $(B_j^r = C_i, 2)$. Let r be the smallest r for which a justified $(B_j^r, 2)$ was seen by an honest party. Then by graded agreement all justified (B_i^r, g) for round r will have $B_i^r = C_i$. Therefore, by Stability, it holds for all justified (B_i^{ρ}, g) for rounds $\rho \geq r$ that $B_i^{\rho} = C_i$. Now consider any other honest party P_k which outputs C_k . Then it saw some justified $(B_j^{r'} = C_k, 2)$. Since we picked r to be minimal we have that $r' \ge r$. From this it follows that $B_j^{r'} = C_i$. Ergo $C_j = C_i$.

A.7 Agreement on a Core Set

We finally present a protocol for Agreement on a Core Set (ACS). It just lets each party propose a set and then picks n - t of them.

 Π_{ACS}

- 1. The input of P_i is B_i .
- 2. Party P_i causal casts B_i as a free-choice message.
- 3. Party P_i collects at least $n t B_j$ from parties $\mathsf{P}_j \in P_i$ and lets $U_i = \{(\mathsf{P}_j, B_j)\}_{\mathsf{P}_j \in P_i}$. This value is justified by $|P_i| \ge n t$.
- 4. Run $\Pi_{\text{SELECTBLOCK}}$ where P_i inputs U_i . The input justifier of $\Pi_{\text{SELECTBLOCK}}$ is that U_i is computed from P_i as in the above step.
- 5. Party P_i gets output C_i from $\Pi_{\text{SELECTBLOCK}}$ and outputs C_i . The output justifier is that C_i is a justified output from the above $\Pi_{\text{SELECTBLOCK}}$.

Fig. 11. Protocol to Agree on a Core Set Π_{ACS}

Lemma 16. If t < n/2 then Π_{ACS} satisfies Definition 9. When $\beta = \sum_{i=1}^{n} |B_i|$ and when using $\Pi_{SELECTBLOCK}$ from Fig. 10 as sub-protocol the complexity is $\mathcal{O}(\beta IN + n^2 RB + n RB_{\#} + ELECT)$ where ELECT is the complexity of Π_{CC} .

Proof. Safety and liveness properties follows directly from those of $\Pi_{\text{SELECTBLOCK}}$. We address the complexity. The causal cast of all blocks B_i costs β IN. Causal casting the inputs to $\Pi_{\text{SELECTBLOCK}}$, U_i , costs n^2 RB to specify the sets P_i . Running $\Pi_{\text{SELECTBLOCK}}$ then costs an additional expected $\mathcal{O}(n^2 \text{RB} + n \text{RB}_{\#} + \text{ELECT})$, where we ignore the IN component as it is run on computed messages. There are no further costs.