On the Number of Synchronous Rounds Required for Byzantine Agreement

Matthias Fitzi[1] and Jesper Buus Nielsen[2]

[1] ETH Zürich
[2] University pf Aarhus

Abstract. Byzantine agreement is typically considered with respect to either a fully synchronous network or a fully asynchronous one. In the synchronous case, either $t+1$ deterministic rounds are necessary in order to achieve Byzantine agreement or at least some expected large constant number of rounds.

In this paper we examine the question of how many initial synchronous rounds are required for Byzantine agreement if we allow to switch to asynchronous operation afterwards.

Let $n = h + t$ be the number of parties where h are honest and t are corrupted. As the main result we show that, in the model with a public-key infrastructure and signatures, $d + O(1)$ deterministic synchronous rounds are sufficient where d is the minimal integer such that $n-d > 3(t-d)$. This improves over the $t+1$ necessary deterministic rounds for almost all cases, and over the exact expected number of rounds in the non-deterministic case for many cases.

Table of Contents

1 Introduction

Two standard timing models are typically considered for the communication among parties in distributed tasks such as Byzantine agreement or general multi-party computation. In the synchronous model, the parties operate in synchronous clock cycles where messages being sent at the beginning of a given clock cycle are guaranteed to have arrived by the end of the same cycle. In the asynchronous model, messages being sent at a certain point in time are only guaranteed to be delivered eventually.

For the case of Byzantine agreement in the synchronous model, it has been observed by Dolev, Reischuk, and Strong [DRS90] that certain tasks can be achieved more efficiently when fully synchronous termination of all parties is not required: simultaneous agreement requires that all parties terminate during the same round (or clock cycle) whereas eventual agreement only requires that all parties eventually reach the same decision. The latter problem can typically be solved more efficiently.

Recently, Beerliova, Hirt, and Nielsen [BHN08] considered a different but related model for general multi-party computation where the protocol is first run in a synchronous environment and then switched to asynchronous operation — the idea being to try to minimize the synchronicity requirements for such protocols. Indeed, they were able to show that one single initial synchronous round of broadcast (followed by asynchronous communication) is sufficient to achieve multi-party computation secure against a faulty minority. In this paper, we address a similar question to the Byzantine agreement problem itself: what is the worst-case number of initial synchronous rounds required in order to achieve eventual Byzantine agreement in an asynchronous environment? Combined with the solution in [BHN08] this would in particular answer what is the worst-case number of initial synchronous rounds required in order to achieve general multi-party computation, but we find the question intriguing in its own right. Let n be the number of parties, t the number of corrupted parties, and $h = n-t$ the number of honest parties. We show that, for any $n > t$, one can do with $t-h/2+O(1)$ initial synchronous rounds in the worst-case. For many parameters this is an improvement over the straight-forward approach of using a protocol where all rounds are synchronous — where $t+1$ is optimal for deterministic protocols [DRS90], and some large constant number of expected rounds is necessary (but not even guaranteed) for probabilistic protocols [FM97,CR93,KK06].

Furthermore, our technique, when applied to the fully synchronous standard model, directly improves over the result by Garay et al. [GKKO07] by reducing the round complexity for Byzantine agreement with a surplus of k dishonest parties from $\Omega(k^2)$ to linear in k.

1.1 Outline

The main protocol is given Section 2. Our improvement over [GKKO07] and some other observations are stated in Section 3.

2 The Protocol

Definition 1 (Broadcast). *A protocol among n parties $P = \{p_1, \ldots, p_n\}$ where a sender $p_s \in P$ inputs a value $x_s \in \{0,1\}$ and each party p_i computes an output $y_i \in \{0,1\}$ achieves broadcast if the following conditions are satisfied:*

1. *(Validity). If p_s is honest then each honest party p_i computes $y_i = x_i$.*
2. *(Consistency). All honest parties compute the same output value.*

Definition 2 (Consensus). *A protocol among* n *parties* $P = \{p_1, \ldots, p_n\}$ *where each party* p_i *inputs a value* $x_i \in \{0, 1\}$ *and each party* p_i *computes an output* $y_i \in \{0, 1\}$ *achieves broadcast if the following conditions are satisfied:*

1. *(Validity). If every honest party* p_i *holds the same input value* $x_i = b$ *then each honest party* p_i *computes* $y_i = x_i$*.*
2. *(Consistency). All honest parties compute the same output value.*

2.1 Overview

With n parties $P = \{p_1, \ldots, p_n\}$ of which $t < n/2$ are corrupted, one can achieve broadcast in $\lceil t/2 \rceil + 4$ synchronous rounds followed by a fully asynchronous protocol. In general, $d + 4$ synchronous rounds are sufficient for d such that $3(t - d) < n - d$. When $t \geq n/2$ we need $d + 5$ rounds.

The synchronous part is an n-party protocol that either detectably achieves agreement or wherein, alternatively, all honest parties detect a common set of some d parties that are corrupted — similar to a single phase in the protocols by Bar-Noy et al. [BDDS92]. We call this protocol Correct-Or-Detect Broadcast, d-CoD . We will choose d such that $n - d > 3(t - d)$, i.e., that out of the $N = n - d$ remaining non-detected parties at most $T = t - d < N/3$ are corrupted.

The d-CoD protocol is followed by a protocol constructing proofs of participation, called the PoP protocol. There exists a verification algorithm ver which takes as input a bit string pop and party id p_j and outputs $\text{ver}(\text{pop}, p_j) \in \{0, 1\}$. Below we write pop_j to mean that pop is a bit string for which $\text{ver}(\text{pop}, p_j) = 1$ and we call such a pop_j a proof of participation for p_j. After the execution of PoP all honest parties will hold some pop_j for all other honest p_j. Furthermore, no pop_j will ever be constructed for a commonly detected p_j. For p_j which is not honest nor commonly detected some honest parties might hold a pop_j and some might not. In addition the proofs pop_j are transferable. I.e., they can be sent along with messages in the asynchronous phase and will be accepted by the recipient. The PoP protocol adds one extra synchronous rounds when $t \geq n/2$.

After the PoP protocol follows the asynchronous part, which is a consensus protocol where the parties only consider messages from parties for which they saw a proof of participation. This will have the effect that the *commonly* detected parties will be no more powerful than being fail-stop corrupted, i.e., having crashed. Thus, if d-CoD achieves common detection of d parties then the asynchronous part basically is a consensus protocol among $N = n - d$ parties with $T = t - d < N/3$ active corruptions or, alternatively, among n parties with T active corruptions and where d parties are fail-stop corrupted (crashed) from the beginning, and where $n > 3T + d$. Note, however, the complicating twist that the parties will not agree on the set of participating parties.

Additionally, the asynchronous part will also guarantee termination even when t parties are actively corrupted but all honest parties hold the same input. So, if the initial d-CoD does not achieve common detection of d parties, then it will achieve agreement, which will still ensure that the asynchronous part terminates.

We now proceed as follows. We give a protocol for synchronous d-CoD in Section 2.2. In Section 2.3 we describe the construction of proofs of participation and describe how to use them to implement the pre-crash model with T active corruptions and where d parties are fail-stop corrupted from the beginning of the protocol. In Section 2.4 we then give an asynchronous consensus protocol for the pre-crash model: we call this protocol *pre-crash consensus, PCC*.

This protocol uses a coin-flip protocol described in Section 2.5. In Section 2.6, we finally show how to combine d-CoD with PCC.

2.2 Correct-or-Detect Broadcast (CoD)

Definition 3 (d-CoD). *A protocol among n parties $P = \{p_1, \ldots, p_n\}$ where a sender $p_s \in P$ inputs a value $x_s \in \mathcal{D}$ and each party p_i outputs a triplet $(y_i, \mathcal{F}_i, \det_i) \in \{0,1\} \times 2^P \times \{C, D\}$ achieves* CORRECT-OR-DETECT BROADCAST WITH d (d-COD) *if the following conditions are satisfied:*

1. ($\mathcal{F}$-SOUNDNESS) *An honest party's set $\mathcal{F}_i$ only contains corrupted parties.*
2. (C-CORRECTNESS) *If any honest party computes* $\det_i = C$ *then the protocol achieves standard broadcast with respect to input x_s and outputs y_i. If standard broadcast is achieved we say that* THE PROTOCOL IS CORRECT. *Furthermore, if p_s is honest then* $\det_i = C$ *for every honest party p_i.*
3. (D-SOUNDNESS) *If any honest party computes* $\det_i = D$ *then* $\left|\bigcap_{p_j \in H} \mathcal{F}_j\right| \geq d$*, where H is the set of honest p_j. In the case of such common detection of d parties we say that* THE PROTOCOL HAS DETECTION.

Let the given instance of the final broadcast protocol to be achieved be defined by ID number id. The protocol below is a $(d+4)$-round construction for d-CoD. The protocol basically proceeds like the first $d+4$ rounds of the protocol in [DS82] for synchronous broadcast. In the first round, if the input is $x_s = 1$, the sender creates a signature on id and sends it to all parties. The first time when a party, during some round $r-1$, receives a chain of $r-1$ different signatures then he accepts the respective input value, appends its own signature, and sends the new chain to all parties in round r. Let r_i be the first round where party p_i receives such a set of signatures where $r_i = d+4$ may also stand for "there is no such round." Depending on r_i party p_i decides in the following way.

r_i	$\leq d+1$	$d+2$	$d+3$	$d+4$
$(y_i, \det_i)$	(1, C)	(1, D)	(0, D)	(0, C)

It is easy to see that $|r_i - r_j| \leq 1$ for all honest p_i, p_j. It will also be easy to see that the parties can commonly detect d corrupted parties if some honest p_i has $r_i \in \{d+2, d+3\}$. Furthermore, as can be seen in the table above, if the honest parties disagree on y_i, then (by $|r_i - r_j| \leq 1$) all honest p_l have $r_l \in \{d+2, d+3\}$. In Protocol 1, we use the following notions:

- A party p_i's signature on a value z is denoted by $\sigma_{p_i}(z)$.
- An 1-CHAIN is a triplet (id, s, σ_s) where σ_s is a valid signature by p_s on id. An ℓ-CHAIN is a tuple $(id, p_{i_1}, \sigma_{i_1}, \ldots, p_{i_{\ell-1}}, \sigma_{i_{\ell-1}}, p_{i_\ell}, \sigma_{i_\ell})$ where $C_{\ell-1} = (id, p_{i_1}, \sigma_{i_1}, \ldots, p_{i_{\ell-1}}, \sigma_{i_{\ell-1}})$ is an $(\ell-1)$-chain and σ_{i_ℓ} is a valid signature by p_{i_ℓ} on $C_{\ell-1}$, and where the parties $p_{i_1}, \ldots, p_{i_\ell}$ are distinct.
- An ℓ-CHAIN WITH RESPECT TO p_i is an ℓ-chain where p_i acts as the last signer in the chain.

Protocol 1: d-CoD

- Round 1:
  - p_s: if the input is $x_s = 1$ then p_s sends 1-chain $C_1 = (id, p_s, \sigma_s)$ to all parties and outputs $(y_s = 1, \mathcal{F}_s = \emptyset, \det_s = \text{C})$. Otherwise, p_s sends nothing and outputs $(y_s = 0, \mathcal{F}_s = \emptyset, \det_s = \text{C})$.

- p_i $(i \neq s)$: $r_i = d + 4$ (sentinel).
- Rounds $2 \leq r \leq d + 4$:
  - p_i $(i \neq s)$: If an $(r-1)$-chain $C_{r-1} = (id, p_{j_1}, \sigma_{j_1}, \ldots, p_{j_{r'}}, \sigma_{j_{r'}}, \ldots, p_{j_{r-1}}, \sigma_{j_{r-1}})$ was received during (previous) round $r-1$ then:
    - If $r_i = d+4$ then $r_i := r - 1$ (mark first round where a sufficiently large chain was received)
    - For one such chain C_{r-1}, send r-chain $C_r = (C_{r-1}, p_i, \sigma_{p_i}(C_{r-1}))$ to all parties.
- Epilogue:
  - p_i $(i \neq s)$:
    - If $r_i \leq d+2$ then $y_i := 1$. If $d+2 \leq r_i \leq d+3$ then $\text{det}_i := \text{D}$ else $\text{det}_i = \text{C}$.
    - For each received ℓ-chain $(id, p_{j_1}, \sigma_{j_1} \ldots, p_{j_\ell}, \sigma_{j_\ell})$ add $p_{j_1}, \ldots, p_{j_{r_i - 1}}$ to $\mathcal{F}_i$. $\diamond$

Lemma 1. *The given protocol efficiently achieves d-CoD in $d + 4$ rounds.*

Proof. If $\text{det}_i = \text{C}$ for some honest party p_i then either $r_i \leq d+2$ or $r_i = d+4$. In the former case, every honest p_j have $r_j \leq d+2$ and thus $y_i = y_j = 1$. In the latter case, every honest p_j have $r_j \geq d+3$ and thus $y_i = y_j = 0$. Finally, if the sender p_s is honest then $y_i = x_s$ and $\text{det}_i = \text{C}$ since the adversary cannot forge signatures. This gives the C-correctness. By construction, no honest p_j signs a chain in round r_j or earlier. Since p_i knows that $r_j \geq r_i - 1$ for all honest p_j, party p_i knows that no honest p_j signed a chain in round $r_i - 1$ or earlier. So, if p_i sees a chain signed by parties $p_{j_1}, \ldots, p_{j_\ell}$, then p_i knows that the parties $p_{j_1}, \ldots, p_{j_{r_i - 1}}$ are corrupted. This implies $\mathcal{F}$-soundness. Let $\mathcal{F}_i^0 = \mathcal{F}_i$ and define a set $\mathcal{F}_i^1$, where p_i for the $(r_i + 1)$-chain it sent in round $r_i + 1$, adds $p_{j_1}, \ldots, p_{j_{r_i - 2}}$ to $\mathcal{F}_i^1$. From $r_j \geq r_i - 1$, the party p_i knows that when an honest p_j saw this chain, then p_j at least added the parties $p_{j_1}, \ldots, p_{j_{r_i - 2}}$ to $\mathcal{F}_j^0$. So, p_i knows that $\mathcal{F}_i^1 \subseteq \mathcal{F}_j^0$ for all honest p_j. Since p_i only sets $\text{det}_i = \text{D}$ if $r_i \in \{d+2, d+3\}$, it follows that $|\mathcal{F}_i^1| \geq r_i - 2 \geq d$, which implies D-soundness. $\square$

2.3 Proofs of Participation (PoP), Minority Case

After running *CoD*, the parties construct proofs of participation. This construction depends on whether $t < n/2$ or $t \geq n/2$. We give the simple construction for $t < n/2$. To not interrupt the flow of presentation of the overall protocol, we defer the description of the construction for $t \geq n/2$ to Section 2.7.

When $t < n/2$ a proof of participation pop_l for p_l is a collection of $n - t$ signatures on $(id, part, p_l)$ from distinct parties. These proofs are clearly transferable. They are constructed asynchronously as follows. Let $\mathcal{P}_i^0 = \{p_1, \ldots, p_n\} - \mathcal{F}_i^0$ and $\mathcal{P}_i^1 = \{p_1, \ldots, p_n\} - \mathcal{F}_i^1$. Each p_i will for each $p_l \in \mathcal{P}_i^1$, send a signature on $(id, part, p_l)$ to p_j. Since $\mathcal{P}_j^0 \subseteq \mathcal{P}_i^1$, p_j knows that all $n - t$ honest p_i will send a signature on $(id, part, p_l)$ for all $p_l \in \mathcal{P}_j^0$. Therefore p_j can wait for $n - t$ such signatures for all $p_l \in \mathcal{P}_j^0$ and thus get a pop_l for all $p_l \in \mathcal{P}_j^0$, which includes the honest parties. No honest party signs $(id, part, p_l)$ for any commonly detected p_l so, since $t < n - t$, no pop_l is constructed for a commonly detected p_l.

2.4 Asynchronous Pre-Crash Consensus (PCC)

Among the n parties, we assume T to be actively corrupted and d to have crashed already before the execution of the protocol, and $n > 3T + d$. We call the $N = n - d$ parties that have not crashed before the execution the PARTICIPATING PARTIES IN PCC . In particular, we have $N > 3T$ among the participating parties. In this section, we make the following assumptions:

- All honest parties detect each other as participating.
- Once an honest p_i detects p_l as participating, all other honest parties detect p_l as participating before they receive their next message from p_i.

As usual we assume the adversary to fully control the actively corrupted parties. From the crashed parties the adversary is allowed to learn their internal states but the crashed parties send no messages during the PCC-protocol. The honest parties do not know the identities of the crashed or the actively corrupted parties.

This model is implemented by relaying all newly received proofs of participation pop_l with the next outgoing message to each of the other parties and ignoring all messages from parties p_l for which no pop_l was yet received.

Our PCC protocol is inspired by the protocol in [CKS00]. Some changes have been made to deal with the fact that we cannot use threshold signature schemes in our setting (the excluded parties can still create signature shares, lending the corrupted parties an unfair advantage). Other changes have been made to simplify the protocol. The well-known standard structure stays the same: repeating rounds over a weak form of agreement (committed crusader consensus) followed by a weak coin-flip protocol.

Definition 4 (Committed Crusader Consensus (CCC)). *A protocol among n parties $P = \{p_1, \ldots, p_n\}$ where every party p_i inputs a value $x_i \in \mathcal{D}$ and outputs a value $y_i \in \{0, \perp, 1\}$ is called* COMMITTED CRUSADER CONSENSUS (CCC) *if the following conditions are satisfied:*

1. (VALIDITY) *If all honest parties have the same input x then every honest party p_i outputs $y_i = x$.*
2. (CONSISTENCY) *If some honest party p_i outputs $y_i = 0$ then no honest party p_j outputs $y_j = 1$.*
3. (COMMITMENT) *As soon as some honest party p_i terminates the protocol, a value $y \in \{0, 1\}$ is fixed such that no honest party p_j can terminate the protocol with output $y_j = y$.*
4. (TERMINATION) *All honest parties terminate the protocol.*

Note that the commitment property defends against the adversary adapting the crusader-consensus outcome to its following coin-flip outcome. This might be possible since the protocol is asynchronous.

Protocol 2: Committed Crusader Consensus — CCC (local code of p_i)

1. Send a signature on $(id, \texttt{vote}, x_i)$ to all parties.
2. Wait for signatures from $N - T$ *participating* parties and pick $u_i \in \{0, 1\}$ to be the value for which $N - 2T$ signatures on $(id, \texttt{vote}, u_i)$ was received. Then send u_i to all parties along with the $N - 2T$ signatures.
3. Wait for $N - T$ *participating* parties p_j to send u_j along with $N - 2T$ signatures on $(id, \texttt{vote}, u_j)$ from *participating* parties.
   - If all u_j are identical then let v_i be the common value and send `ok!` to all parties.
   - Otherwise, let $v_i = \perp$, pick $N - 2T$ of the signatures on $(id, \texttt{vote}, 0)$ and $N - 2T$ of the signatures on $(id, \texttt{vote}, 1)$ and combine them to a PROOF OF DISAGREEMENT, and send this proof to all parties.[3]
4. Wait to receive `ok!` or a proof of disagreement from $N - T$ *participating* parties. If all sent `ok!`, then let $y_i = v_i$. Otherwise, let $y_i = \perp$. Then send `done!` to all parties.

[3] A proof of disagreement shows that at least one honest party had input 0 and that at least one honest party had input 1.

5. Wait for $N-T$ *participating* parties to send `done!`, and then terminate with output y_i.◇

Lemma 2. *The above protocol achieves CCC.*

Proof. 1. (VALIDITY). Straight forward.
2. (CONSISTENCY). Assume that p_i is honest and that $y_i = b \in \{0,1\}$. Then p_i received $N-T$ votes u_j on b. At least $N-2T$ of these votes were sent by honest parties, so any other honest p_k sees at least one vote u_j on b and thus outputs b or $\perp$.
3. (COMMITMENT). As for commitment, assume that some honest party terminated. This means that it saw $N-T$ parties send `done!`. Therefore at least $N-2T$ honest parties sent `done!`. We consider these $N-2T$ parties and distinguish two cases:
   - Assume that one of the considered parties had $v_i = b \in \{0,1\}$. In this case it is easy to see that $v_j = 1-b$ is impossible for all other honest parties p_j. Since $y_j = 1-b$ implies $v_j = 1-b$, it follows that $v_j = 1-b$ is impossible for each honest party p_j.
   - Assume that $v_i = \perp$ for the at least $N-2T$ considered honest parties. Then these $N-2T$ parties sent a proof of disagreement to all parties. Therefore every honest party p_j receives at least one proof of disagreement and thus outputs $y_j = \perp$, making both $y_k \in \{0,1\}$ impossible for every honest party p_k.
4. (TERMINATION). Straight forward. □

We can now combine committed crusader consensus with an unpredictable coin-flip protocol. We assume a coin-flip protocol where the parties input $(id, \texttt{flip}, r)$ to flip coin number r and where they receive an outcome $(C_r \in \{0,1\}, g \in \{0,1\})$. We assume that if $N-T$ honest parties input $(id, \texttt{flip}, r)$, then they eventually all receive an outcome (v, g). The outcome should guarantee that if some honest party has outcome $(v, 1)$, then all honest parties have outcome $(v, 1)$ or $(v, 0)$. Furthermore, a fraction p of the coins should have the property that all parties output $(C_r, 1)$ and that C_r cannot be predicted with probability negligibly better than $\frac{1}{2}$ until the first honest party gets input $(id, \texttt{flip}, r)$. We call such a coin GOOD. See Section 2.5 for the implementation of such a coin.

Protocol 3: Pre-Crash Consensus — PCC (local code of p_i)

1. Let $r = 1$ and let x_i be the input.
2. Run CCC on input x_i to get an output z_i.
3. Input $(id, \texttt{flip}, r)$ to the coin-flip protocol and wait for an output $(C_r \in \{0,1\}, g)$.
4. If $z_i = C_r$ and $g = 1$, then $y_i := z_i$ but *do not terminate.*
5. If $z_i = \perp$ then $x_i = C_r$ else $x_i = z_i$.
6. Let $r := r + 1$ and go to Step 2. ◇

Lemma 3. *The above protocol achieves pre-crash consensus (except for termination) in expected* $2/p$ *"rounds".*

Proof. The protocol has the following properties and thus fulfills the claims:

- (PERSISTENCY: IF THE HONEST PARTIES AGREE ON THE x_i AT THE BEGINNING OF ROUND r, THEN THEY WILL ALL HAVE $y_i = x_i$ AND WILL END UP WITH THE SAME x_i IN ROUND $r+1$). This follows by validity of CCC.
- (VALIDITY). Follows from persistency.

- (MATCHING COIN: IF NO HONEST PARTY RECEIVES OUTPUT $z_i = z \in \{0,1\}$ FROM THE CCC PROTOCOL AND COIN C_r IS GOOD, THEN WITH PROBABILITY NEGLIGIBLY CLOSE TO $\frac{1}{2}$, THE COIN PRODUCES $C_r = 1 - z$ AND $g = 1$ FOR ALL PARTIES). When the first honest party p_i inputs $(id, \texttt{flip}, r)$ to the coin-flip protocol, at least p_i terminated the CCC protocol. Therefore there exists $z \in \{0,1\}$ such that no honest party can end up with $z_i = z$. Since C_r is unpredictable until the first honest party inputs $(id, \texttt{flip}, r)$ there is probability negligibly close to $\frac{1}{2}$ that $C_r = 1 - z$.
- (CONSISTENCY). By the matching-coin condition, all honest players eventually end up with the same value y_i.
- (CONSISTENCY DETECTION: WHEN AN HONEST PARTY p_i SEES THAT $z_i = C_r$ AND $g = 1$ THEN p_i KNOWS THAT ALL HONEST PARTIES p_j WILL FINALLY COMPUTE OUTPUT $y_j = z_i$). If this happens then, after Step 5 of the same iteration, all honest parties will agree on $x_j = z_i$ which will persist by the persistency condition.
- (NUMBER OF "ROUNDS"). Follows from the properties of the coin.

As of now, the protocol runs forever. Allowing all honest parties to terminate in a constant number of rounds can be achieved by adding the following rules to the above protocol.

- After computing y_i in Step 4, send a signature on $(id, \texttt{result}, y_i)$ to all parties.
- On receiving a valid signature on $(id, \texttt{result}, y)$ by any party, send it to all parties.
- If for some $y \in \{0,1\}$ and $h = N - T$ distinct participating parties p_j a valid signature on $(id, \texttt{result}, y)$ by p_j has been (received and) resent to all parties, terminate.

Theorem 1. *The above protocol together with the given termination augmentation achieves pre-crash consensus among n parties with d pre-crashes and T actively corrupted players when $n > 3T + d$. The protocol terminates in $2/p$ expected rounds where p is the unpredictability of the coin.*

Proof. Follows from Lemma 3 and the above discussion. □

2.5 The Coin

We first describe a generic construction that can be based on any existing coin protocol in order to get a coin for the pre-crash model that can be set up during the last synchronous round and opened during the asynchronous part of the protocol. The coin itself is pre-shared by some designated party p_i and is reliable when p_i is honest. The iterations of pre-crash consensus can then be done with respect to different designated parties. The advantage of this approach is that 1 synchronous round is sufficient and that it can be described generically. The disadvantage is that expected $O(t)$ asynchronous "rounds" will be required in order to hit a reliable matching coin.

Second, we sketch how to produce a coin along the lines of Katz and Koo [KK06] solely based on digital signatures (and a PKI). The advantage of this construction is that only an expected constant number of asynchronous "rounds" will be required to hit a reliable matching coin. The disadvantage is that more than 1 synchronous round is required to set it up (but still only $O(1)$).

Generic Construction. We let each party P_i prepare a coin C_i in the last synchronous round, by picking $C_i \in \{0,1\}$ uniformly at random and secret sharing C_i. In asynchronous round i, the parties then try to reconstruct C_i. To get ℓ coins, each p_i prepares ℓ/n coins and the coins of p_i are used in rounds $i + nq$. We pick ℓ large enough that the PCC protocol will have terminated after ℓ phases except with negligible probability when there is detection (and thus $N > 3T$). If the parties run out of coins, they conclude that there was not detection, but agreement already in CoD. In the following, we let $\mathcal{P}_i = \{1, \ldots, n\} - \mathcal{F}_i$.

Protocol 4: Coin Flip

- A coin of p_i is prepared as follows:
  - Last synchronous round: p_i picks C_i uniformly at random and creates a Shamir sharing of C_i among n parties with degree T. Let C_{ij} denote the share of p_j. For $p_j \in \mathcal{F}_i$ it deletes C_{ij}. For $p_j \in \mathcal{P}_i$ it signs $(id, \texttt{flip}, i, j, C_{ij})$ and sends it securely to p_j.
- The flipping of the coin proceeds as follows:
  - p_j: On input $(id, \texttt{flip}, r = i + nq)$, send the signed $(id, \texttt{flip}, r, j, C_{ij})$ to all p_k, if it was received, and otherwise sign and send $(id, \texttt{flip}, r, j, \perp)$.
  - p_k: Wait for $N - T$ participating parties p_j from $\mathcal{P}_k$ to send $(id, \texttt{flip}, r, j, C_{ij})$ signed by p_i or $(id, \texttt{flip}, r, j, \perp)$ signed by p_j. If $N - 2T$ participating parties sent a signed $(id, \texttt{flip}, r, j, \perp)$, collect the signatures to a PROOF THAT p_i IS CORRUPT,[4] and send the proof to all parties. Otherwise, if $N - 2T$ participating parties sent a signed $(id, \texttt{flip}, i, j, C_{ij})$, then interpolate a degree T polynomial $f(\texttt{X})$ with $f(j) = C_{ij}$ for all $N - 2T$ values, let $C_i = f(0)$ and collect the $N - 2T$ signed values to a PROOF THAT $C_i = f(0)$ IS JUSTIFIED, and send the proof to all parties along with a signature on $(id, \texttt{flip}, r, f(0))$.
  - p_j: Wait for $N - T$ participating parties p_k to send a message as required above. If one of them sent a proof that p_i is corrupt, store this proof. Otherwise, if one of them sent a proof that $C_i = 0$ is justified and one of them sent a proof that $C_i = 1$ is justified, pool these proofs to a PROOF THAT p_i IS CORRUPT. Otherwise, the $N - T$ parties all sent a PROOF THAT $C_i = v$ IS JUSTIFIED for the same v. Collect the corresponding $N - T$ signatures on $(id, \texttt{flip}, r, v)$ to a PROOF THAT $C_i = v$ IS UNIQUELY JUSTIFIED.[5] In both cases, send the obtained type of proof to all parties.
  - p_k: Wait for $N - T$ participating parties p_j to send a proof that p_i is corrupt or that some v is uniquely justified. If all $N - T$ parties sent a proof that v is uniquely justified, then output $(v, 1)$. If at least one party sent a proof that v is uniquely justified, then output $(v, 0)$. Otherwise, output $(0, 0)$

It is straight-forward to see that at most one value v will have a proof that it is uniquely justified. Furthermore, all pairs of parties receive a message from at least one common honest party in the last step. So, as p_j having output $(v, g = 1)$ implies that it received a proof that v is uniquely justified from all parties, it knows that all other honest parties received at least one such value, and therefore has output $(v, \cdot)$. Therefore output $(v, 1)$ implies that all parties agree on the coin. Finally, if p_i is honest, no proof that p_i is corrupt will be constructed. Therefore all parties will have output $(C_i, 1)$. So, all coins prepared by honest parties are good, and they make up a fraction $p = (n - t)/n$ of all coins.

Construction after [KK06]. We adapt the construction in [KK06] in the following way:

- Party p_i marks all parties in $\mathcal{F}_i$ as untrusted.

[4] At least one honest party is claiming that it did not get a signed share from p_i.

[5] At least $N - 2T$ honest parties signed for v and no honest party signs for both 0 and 1. Therefore at most one value v will have such a proof.

– The moderated VSS protocol is run with respect to threshold t.
– In the moderated VSS protocol the dealer p_D *openly* distributes the shares for the parties it detected in $\mathcal{F}_D$.
– In the moderated VSS protocol party p_i marks the dealer as untrusted (and moves it to $\mathcal{F}_i$) if the set S_D of parties whose shares are openly distributed by p_D satisfies $S_D \cap \mathcal{F}_i < d$.

The resulting moderated VSS protocol will now be valid and secret when p_D is honest but a dishonest dealer p_D is still committed, i.e., the dishonest dealer may refuse to have the secret reconstructed but cannot change the secret anymore. The leader is now elected in the same way as in [KK06] — the moderator of the minimal outcome among the moderators that are still trusted. Among those there is an honest majority because of common detection during d-CoD. The coin can now for instance be implemented by opening a value that the leader shared out during the synchronous phase of the protocol, as in the generic construction.

2.6 The Final Protocol: Putting Things Together

We now demonstrate how to combine synchronous d-CoD with asynchronous pre-crash consensus (PCC).

Let $h = n - t$ be the number of honest parties. Pick $T < h/2$, let $d = t - T$ and let $N = n - d$. We first run Protocol d-CoD with respect to n and t. Protocol d-CoD is either correct or has detection. The difficulty now is that the parties do not necessarily agree on their values $\det \in \{\mathrm{C}, \mathrm{D}\}$ that stand for knowing that correctness or detection was achieved. Therefore we always unconditionally append Protocol Pre-Crash Consensus (PCC) which is run among all n parties and with respect to d crashes and $T < N/3$ active corruptions. However, only the parties p_i with $\det_i = \mathrm{D}$ will adopt the output of PCC, whereas the parties with $\det_i = \mathrm{C}$ already accept their outputs from d-CoD. In more detail:

Protocol 5: Broadcast (local code of p_i)

1. Run the d-CoD on input x_i and let $(v_i, \mathcal{F}_i, \det_i)$ be its output.
   <FROM NOW ON EVERYTHING IS ASYNCHRONOUS>
2. If $\det_i = \mathrm{C}$ then output $y_i = v_i$ but *do not terminate.*
3. Run PoP to create proofs of participation and use these to simulate a pre-crash model, where all parties without a proof of participation are considered crashed.
4. Run PCC on input v_i in the simulated pre-crash model; if $\det_i = \mathrm{D}$ then let y_i by the output of PCC. ◇

The final protocol has the following properties:

1. If $\det_i = \mathrm{C}$ for all honest p_i then all honest parties eventually output some y_i and the outputs y_i are correct.
   Proof: When $\det_i = \mathrm{C}$ for all honest p_i then all honest p_i have $y_i = v_i$ where v_i is the output of d-CoD, which is correct since $\det_i = \mathrm{C}$ for just one honest P_i.
2. If $\det_i = \mathrm{D}$ for some honest p_i, then the asynchronous BA eventually terminates with some common output y which is equal to some v_i held by an honest party p_i.
   Proof: Even a party with $\det_i = \mathrm{C}$ will run the asynchronous PCC protocol. Therefore the asynchronous PCC is run by *all* honest parties, but as if all parties without a proof of participation were crashed before the protocol began. The only malicious thing a party without a proof of participation can do is therefore to leak its secrets to the corrupted parties which do have proof of participation. Therefore the PCC protocol is essentially run

in a pre-crash model with N being the number of parties with a proof of participation and T being the number of corrupted parties with a proof of participation. When $\det_i = \mathrm{D}$ for some honest p_i, then $N > 3T$ as no commonly detected party gets a proof of participation and all honest parties get a proof of participation. Since $N > 3T$, it follows from the properties of PCC that it eventually terminates with some common output y which is equal to some v_i held by an honest party p_i.

3. If $\det_i = \mathrm{D}$ for all honest p_i, then all parties eventually output some y and the output y is correct.
   Proof: When $\det_i = \mathrm{D}$ for all honest p_i, then all honest p_i take the output y_i of PCC to be the output of the final protocol. If p_s is honest, then no honest p_i has $\det_i = \mathrm{D}$, so when $\det_i = \mathrm{D}$ for all honest p_i, p_s is corrupted and any common output $y_i = y$ is correct. It is therefore sufficient that PCC has termination and consistency. This follows from Property 2.
4. If $\det_i = \mathrm{C}$ for some honest p_i and $\det_j = \mathrm{D}$ for some honest p_j then all honest parties eventually output some y and the output y is correct.
   Proof: From $\det_j = \mathrm{D}$ for some honest p_j it follows from Property 2 that PCC eventually terminates with some common output y, which is equal to some v_i held by an honest party. By $\det_i = \mathrm{C}$ for some honest p_i it follows that the v_i held by the honest parties are the same, meaning that each honest p_i gets output $y_i = v_i$ from PCC where v_i is its output from d-CoD. Therefore every honest p_i will output v_i which is correct since p_s must be corrupted. □

Theorem 2. *The above protocol achieves broadcast for n parties secure against $t < n/2$ actively corrupted parties in*

- $\lceil \frac{2t-h}{2} \rceil + 4 \leq \lceil t/2 \rceil + 4$ *deterministic synchronous rounds followed by an expected-$O(t)$-"round" asynchronous protocol when using the generic coin.*
- $t - h/2 + O(1)$ *deterministic synchronous rounds followed by an expected-$O(1)$-"round" asynchronous protocol when using the specific coin.*

Proof. From Property 1, Property 3 and Property 4 it follows that the protocol achieves broadcast. The rest can be verified by inspection. □

2.7 Proofs of Participation (PoP), Majority Case

We now describe the construction of proof of participation for the case $t \geq n/2$. The first modification is that we run CoD for one more round and let the parties decide as follows.

r_i	$\leq d+2$	$d+3$	$d+4$	$d+5$
$(y_i, \det_i)$	$(1, \mathrm{C})$	$(1, \mathrm{D})$	$(0, \mathrm{D})$	$(0, \mathrm{C})$

I.e., just run $(d+1)$-CoD. For the (r_i+1)-chain sent in round (r_i+1), p_i adds $p_{i_1}, \ldots, p_{i_{r_i-3}}$ to a set $\mathcal{F}_i^2$. In all rounds, for all incoming chains, it adds $p_{i_1}, \ldots, p_{i_{r_i-2}}$ to a set $\mathcal{F}_i^1$ and adds $p_{i_1}, \ldots, p_{i_{r_i-1}}$ to a set $\mathcal{F}_i^0$. Except in the last round, it then relays the chain, which will make other parties p_j see the chain and do the same, possible using $r_j = r_i - 1$. It can be seen that this results in sets with $\mathcal{F}_i^2 \subseteq \mathcal{F}_j^1 \subseteq \mathcal{F}_k^0$ for all honest p_i, p_j, p_k. The output of this modified d-CoD is taken to be $\mathcal{F}_i = \mathcal{F}_i^2$. We call $\mathcal{P}_i^b = \{1, \ldots, n\} - \mathcal{F}_i^b$ the b-PARTICIPATING PARTIES (seen by p_i) and we have that $\mathcal{P}_i^0 \subseteq \mathcal{P}_j^1 \subseteq \mathcal{P}_k^2$ for all honest p_i, p_j, p_k. We need that $N > 3T$ for all sets $\mathcal{P}_j^l$ when an honest p_i has $\det_i = \mathrm{D}$. This follows from $r_i \geq d+3$.

Now, for $b = 2, 1, 0$ we define a b-PROOF OF PARTICIPATION FOR p_l (from the viewpoint of p_i) to be $2T + 1$ signatures on $(id, part, p_l)$ from parties in $\mathcal{P}_i^b$.

Invariant. We will maintain the invariant that if there exists any b-proof of participation for p_l, then at least one honest party did not exclude p_l initially (we say p_i excluded p_l if $p_l \notin \mathcal{P}_i^2$). In particular, there will exist no b-proof of participation for a commonly detected party p_l. Furthermore, all honest parties will initially hold a 0-proof of participation for all other honest parties, and 0-proofs of participation will be transferable. Therefore 0-proofs of participation can be used to simulate the pre-crash model, as desired.

Establishing the invariant. Every p_i sends a signature on $(id, part, p_l)$ for all $p_l \in \mathcal{P}_i^0$ to all p_j. Below we call a signature on $(id, part, p_l)$ a SIGNATURE OF PARTICIPATION FOR p_l. Since all honest parties are in all $\mathcal{P}_i^0$, and there are at least $N - T$ honest parties, all honest p_j can wait to collect a 0-proof of participation for all $p_l \in \mathcal{P}_j^1$, which includes all honest parties. Furthermore, if all honest parties excluded p_l, then less than $N - T$ signatures of participation are constructed for p_l, so no b-proof of participation is constructed for a commonly detected party.

Upgrading. To build transferability, we first observe that if p_j holds a 1-proof of participation for p_l, it can upgrade it to a 0-proof of participation: It sends the 1-proof of participation to all p_k. Any p_k receiving it will see the $N - T$ signatures from $\mathcal{P}_j^1$. Since $\mathcal{P}_j^1 \subset \mathcal{P}_k^2$, this will be a 2-proof of participation for p_l to p_k. Therefore p_k knows, by the invariant, that at least one honest party did not exclude p_l initially. Therefore p_k can safely sign $(id, part, p_l)$ and send the signature to p_j. All honest p_k will eventually do this. Since all honest parties are in $\mathcal{P}_j^0$, p_j will eventually receive $N - T$ signatures on $(id, part, p_l)$ from parties in $\mathcal{P}_j^0$. These signatures p_j collects to a 0-proof of participation for p_l.

Transfer. Assume now that p_i holds a 0-proof of participation for p_l. It can send this to all p_j, who will see it at least as a 1-proof of participation, as $\mathcal{P}_i^0 \subseteq \mathcal{P}_j^1$. Therefore p_j can upgrade it to a 0-proof of participation for p_l. This gives transferability.

Theorem 3. *The above protocol achieves broadcast for n parties secure against $t < n$ actively corrupted parties where d is the minimal integer for which $n - d > 3(t - d)$*

- *in $d + 5$ deterministic synchronous rounds followed by an expected-$O(t)$-"round" asynchronous protocol when using the generic coin.*
- *in $d + O(1)$ deterministic synchronous rounds followed by an expected-$O(1)$-"round" asynchronous protocol when using the specific coin.*

Proof. The theorem follows along the lines of the proof of Theorem 2 and the above discussion. □

3 Observations and Applications

3.1 Multi-Valued Broadcast.

Above, our protocols were stated with respect to a binary value domain. We note that, for arbitrary value domains $\mathcal{D}$, the protocol can be adapted such that the number of synchronous rounds stays the same whereas the number of asynchronous rounds remains of same order (one additional round per coin flip). For instance, this can be achieved by modifying Protocol PCC such that every player gradecasts his value z_i *ahead* of the coin flip, and having multiple coins to elect a leader to dictate a default value.

3.2 Broadcast Without a PKI.

We note that it is easy to see that, when not given a PKI in our model, broadcast is achievable if and only if $n > 3t$.

3.3 Fully Synchronous Byzantine Agreement with a PKI.

We observe that our approach can also be used to improve over the result in [GKKO07]. There it was shown how to achieve Byzantine agreement in the fully synchronous model among $n = 2h + k$ parties with a minority of h honest parties in expected $\Omega(k^2)$ rounds. Applying our approach together with the specific coin from [KK06] directly to the fully synchronous case yields a protocol that requires only expected $k + O(1)$ rounds. Note that, in this case, the coin does not have to be pre-shared as described in Section 2.5 but the leader can simply dictate it on the spot. The resulting protocol only relies on signatures and a PKI and works for any value domain.

References

[BDDS92] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: Changing algorithms on the fly to expedite byzantine agreement. *Inf. Comput.*, 97(2):205–233, 1992.

[BHN08] Zuzana Beerliova, Martin Hirt, and Jesper Buus Nielsen. Almost-asynchronous multi-party computation with faulty minority. Manuscript, 2008.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In *PODC*, pages 123–132. ACM, 2000.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience (extended abstract). In *STOC*, pages 42–51, 1993.

[DRS90] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990.

[DS82] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *STOC*, pages 401–407. ACM, 1982.

[FM97] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[GKKO07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *FOCS*, pages 658–668. IEEE Computer Society, 2007.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462. Springer, 2006.