How Byzantine is a Send Corruption?

Karim Eldefrawy[1], Julian Loss[2], and Ben Terner[3]

[1] SRI International, Menlo Park, CA karim.eldefrawy@sri.com
[2] University of Maryland, College Park, MD julianloss@gmail.com
[3] UC Irvine, Irvine, CA bterner@uci.edu

Abstract. Consensus protocols enable n parties, each holding some input string, to agree on a common output even in the presence of corrupted parties. While the problem is well understood in the classic byzantine setting, recent work has pushed to understand the problem when realistic types of failures are considered and a majority of parties may be corrupt. Garay and Perry [11] consider a model with both byzantine and crash faults and develop a corruption-optimal protocol with perfect security tolerating t_{cra} crash faults and t_{byz} byzantine faults for $n > t_{\mathsf{cra}} + 3t_{\mathsf{byz}}$. Follow up work by Zikas, Hauser, and Maurer extends the model by considering *receive-corrupt* parties that may not receive messages sent to them, and *send-corrupt* parties whose sent messages may be dropped. Otherwise, receive-corrupt and send-corrupt parties behave honestly and their inputs and outputs are considered by the security definitions. Zikas, Hauser, and Maurer gave a perfectly secure, linear-round protocol for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 3t_{\mathsf{byz}}$, where t_{rcv} and t_{snd} represent thresholds on the number of parties that are receive-or-send-corrupted.

In this paper we ask *"what are optimal thresholds in the cryptographic setting that can be tolerated with such mixes of corruptions and faults?"* We develop an expected-constant round protocol tolerating $n > t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$. We are unable to prove optimality of our protocol's corruption budget in the general case; however, when we constrain the adversary to either drop all or none of a sender's messages in a round, we prove our protocol achieves an optimal threshold of $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$. We denote this weakening of a send corruption a *spotty send corruption*.

In light of this difference in corruption tolerance due to our weakening of a send corruption, we ask *"how close (with respect to corruption thresholds) to a byzantine corruption is a send corruption?"* We provide a treatment of the difficulty of dealing with send corruptions in protocols with sublinear rounds. As an illustrative and surprising example (even though not in sublinear rounds), we show that the classical Dolev-Strong broadcast protocol degrades from $n > t_{\mathsf{byz}}$ corruptions in the byzantine-only model to $n > 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ when send-corrupt parties' outputs must be consistent with honest parties; we also show why other recent dishonest-majority broadcast protocols degrade similarly. We leave open the question of optimal corruption tolerance for both send- and byzantine corruptions.

Keywords: Consensus · byzantine agreement · constant rounds · dishonest majority

1 Introduction

Consensus protocols, also known as byzantine agreement protocols, enable n parties, each holding some input value, to agree on common outputs even in the presence of some threshold of byzantine corrupted parties. While protocols designed for byzantine failures achieve the strongest security guarantees, they often do not accurately reflect the real world; in practice, crashing a party[4] is much easier and cheaper than corrupting it.

Toward a more realistic failure model in which a majority of parties may be corrupted, a line of work has explored mixed models in which both crash faults and byzantine faults are permitted. In the error-free setting, Garay and Perry [11] and Altmann, Fitzi, and Maurer [4] show that byzantine agreement is possible if and only if $n > t_{\mathsf{cra}} + 3t_{\mathsf{byz}}$. In the asynchronous model, the work of Backes and Cachin [5] showed that reliable broadcast within the mixed model is possible if and only if $n > 2t_{\mathsf{cra}} + 3t_{\mathsf{byz}}$. For byzantine agreement, Kursawe [15] developed a protocol for the same bound ($n > 2t_{\mathsf{cra}} + 3t_{\mathsf{byz}}$) assuming a public key infrastructure (PKI). Recent work in the dishonest majority setting by Wan et al. [22, 21] showed round efficient broadcast protocols in the majority dishonest setting. Expanding further into even more realistic failures, Zikas, Hauser and Maurer [23] gave a protocol in the error-free synchronous model for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 3t_{\mathsf{byz}}$, where t_{rcv} bounds the number of receive corruptions, t_{snd} bounds the number of send corruptions, and t_{byz} bounds the number of byzantine corruptions.

Faulty Parties With Consistent Outputs. The work by Zikas, Hauser and Maurer is notable because their model introduces parties which may be faulty *but the faulty processors' outputs must be consistent with honest parties' outputs* because they otherwise behave honestly. In all other corruption models, the output of any faulty party need not be considered towards the definition. We extend their work by asking: *"what are optimal thresholds in the cryptographic setting that can be tolerated with mixed corruptions?"* We show that the duality of a send-corrupt party whose outputs must nonetheless be consistent with honest parties introduces new challenges. In this model, is it currently not known how to push corruption tolerance for sublinear-round broadcast to the dishonest majority setting, and for sublinear-round consensus we do not know how to do better than treating a send-corrupt party as fully byzantine.

1.1 Send and Receive Corruptions: Honest-but-Faulty

Send-corrupt parties participate in a protocol as honest parties do, but an adversary has the power to determine which messages sent by a send-corrupt party are delivered and which are not. Nevertheless, they still listen to the protocol and their outputs must be consistent with the honest parties' outputs.

Receive-corrupt parties may cease to receive messages, but are guaranteed that the messages they send are delivered. In any protocol, a receive-corrupted

[4] Crashing parties may be achieved via denial-of-service (DoS) attacks, one of the most frequent attacks on distributed systems and on the Internet.

party may detect that it is receive-corrupted if it does not receive messages that it is expecting. If a receive-corrupted party detects that it is corrupted, then – as in [23] – the party moves into a *zombie* state. A party that becomes a zombie stops sending and receiving messages, and outputs $\perp$, becoming the functional equivalent to a crashed party in the common literature. If a receive-corrupted party has not detected that it is corrupt then it may continue to participate, and we require that its output agrees with the honest parties' outputs, even though it may not receive all protocol messages.

Send corruptions and receive corruptions are both strict generalizations of crash corruptions, as in either case a crashed party ceases to send and receive messages. However, unlike send- and receive-corrupted parties, crashed parties' outputs are never required to be consistent with the honest parties' outputs.

Zombies and Live Parties. Because send-corrupted and receive-corrupted parties may still continue to participate without intentionally deviating from the protocol, our definitions require that their outputs (if they produce outputs) are consistent with those of the honest parties. We call all honest, send-corrupt, and non-zombie receive-corrupt parties *live parties*; this denotes that the party continues to (try to) participate as if it were an honest party.

We use the convention that whenever a party becomes a zombie, it sends a special message (zombie) to all other parties. Upon receiving such a message, a party deducts one from its count of n the number of parties, as well as deducts one from its threshold for the number of receive-corrupted parties. Note that send-corrupt parties may fail to send their zombie declarations, and receive-corrupt parties may fail to receive other parties' declarations.

1.2 The Pathology of Send Corruptions

We consider two forms of send corruptions, one more pathological than the other.

1. *Standard send corruption:* In the general case (denoted as simply a send corruption) as in [23], the adversary may adaptively drop any of a send-corrupt party's outgoing messages in any round.
2. *Spotty send corruption:* In a weaker case, an adversary adaptively drops either *all or none* of a send-corrupt party's outgoing messages in a round.

We consider a strongly adaptive, rushing adversary that can observe all of a party's messages and then remove them after they have sent.

Pathology of a (standard) send corruption. Our standard model of a send corruption permits the adversary to selectively drop messages by send-corrupting a party. Because this behavior is a subset of a byzantine corruption, one would expect that corruption bounds follow directly from the byzantine case. We show that this is not the case in general. The consistency property of many current protocols breaks under the specific attack that some (corrupt) party selectively sends a message to some honest parties which other honest parties may never receive. In our model, a send-corrupt party may receive a message that would change its output *and fail to inform any honest party about the message.*

As an illustrative example (embodying a common technique), the Dolev-Strong broadcast protocol requires that if some honest party – whose output is constrained by definition – receives a message, then all other honest parties will receive that message before the protocol terminates. But as we show in Section 3.2, Dolev-Strong breaks down in our model because a send-corrupt party may receive a message that would change its output but fail to forward it.

Generalizing this theme, an adversary can undermine current techniques by dividing the execution such that send-corrupt parties are unable to send messages to honest parties, but send-corrupt parties receive all messages sent by honest parties. Crucially, a situation results where there are two sets of parties with different sets of messages received by the end, but their outputs are constrained by consistency. For example, divide an execution into sets such that S contains all send-corrupt parties and H contains all honest parties, and let $|S| > |H|$. Then it may be the case that *a majority of parties cannot communicate with the honest parties*, but all of their outputs must be consistent.

Although we cannot prove it, it appears very difficult to tolerate more send-corrupt parties than honest parties because this partition requires the use of some threshold scheme to ensure sufficiently many parties are "aware" of a message to allow it to influence the output. Specifically, we do not know how to generate and use information that an honest party has *not* received a message sent by a send-corrupt party as part of the protocol. On the other hand, impossibility proofs that depend on partitioning techniques also fail in this model because it is impossible to completely separate the send-corrupt group from the honest group, since send-corrupt parties always receive all of the honest parties' messages.

The spotty corruption model, described below, alleviates the above issue because it enforces unanimity: if any send-corrupt party or honest party receives a message sent by a send-corrupt party, then all honest and send-corrupt parties receive the message. This is sufficient to enforce consistency of honest and send-corrupt views that permits thresholds over the number of byzantine parties.

Pathology of a spotty send corruption. We argue that although our "spotty" send corruption is limited in some ways, it is still rich enough to cause failure of some popular techniques for synchronous consensus. In particular, it is unclear how to construct a protocol that employs leader election in order to reach constant expected round complexity in our model. Specifically, a strongly rushing adversary as described above can wait for a leader to be elected – and even to send messages that attest to its election (e.g., based on a Verifiable Random Function, VRF, as in [19, 12]) – spotty-corrupt the party, and force it to fail as leader for the duration of its tenure, without even expending its budget towards byzantine corruptions. While in the purely byzantine model this type of attack can be easily mitigated by using threshold signatures (see, e.g., [16, 2]), this approach completely fails in our model, as electing a leader in this way would most likely elect one of the potentially t_{snd} send-corrupt parties (since t_{snd} can be much larger than the number of honest and in-synch parties). For this reason, recent protocols for dishonest majority broadcast that rely on the player-replaceable paradigm, such as [6] and [1] fail in our model.

1.3 Contributions

We provide the first systematic treatment of the pathology of send-corruptions, and show that considering send-corrupt parties as “nearly” honest in the definition *either completely breaks or substantially deteriorates the corruption tolerance of both classical and recent broadcast protocols.*

We then provide an expected-constant-round byzantine agreement protocol that is secure in the strongly adaptive setting against t_{snd} send-corruptions, t_{rcv} receive corruptions, and t_{byz} byzantine corruptions where $t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}} < n$. Our protocol builds consensus from graded consensus and a common coin [8, 14], with subtle adaptations for our corruption model, with a parallelization of the implementation of FixReceive from [23]. When send-corruptions are *spotty*, we show our protocol achieves optimal corruption tolerance of $t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}} < n$.

To our knowledge, our consensus protocol for the spotty send model is the first sublinear-round consensus protocol where a majority of online parties may be faulty *in any model.*

1.4 Comparison with Related Work and Obvious Solutions

Recent Advances in Dishonest-Majority Broadcast. One might expect that because dishonest-majority broadcast protocols tolerate $n > t_{\mathsf{byz}}$ corruptions, they are sufficient for building a consensus protocol tolerating $n > t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ corruptions via folklore reductions (which we discuss in detail in Appendix B), which would achieve better corruption tolerance than our construction in Section 4. We show that this is not true and that recent advances in dishonest-majority for adaptive adversaries by Wan et al [22, 21] also fail in our model.

The work of Wan et al. [22] provides an expected constant round protocol for dishonest majority broadcast under a weakly adaptive adversary. However, their “Trust Graphs” assume that only byzantine parties do not send messages, and any party that fails to send a message can be excluded. This fails in our model because send-corrupt parties must be consistent with honest parties.

Another recent work [21] uses time-lock puzzles to provide a round-efficient broadcast protocol in the presence of dishonest majority and a strongly adaptive adversary. However, the approach also fails because honest parties may never learn the puzzles sent by send-corrupt parties. It is possible to construct an execution in which honest parties solve a set of puzzles T, and the send-corrupt parties solve another set of time-lock puzzles $T' = T \cup S$, where S are puzzles that are never distributed to the honest parties. However, our definitions require that send-corrupt parties' outputs match those of the honest parties.

Adapting ZHM [23] to an Expected Constant-Round Protocol. A natural attempt to achieve sub-linear round consensus tolerating $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 3t_{\mathsf{byz}}$

Table 1. Comparison with relevant consensus protocols in mixed corruption models. $\widehat{R}$ indicates the round complexity R is given in expectation; otherwise the round complexity is worst-case. DS denotes Dolev-Strong.

Protocol	**Faults**	**# Rounds**
Modified DS (Section 3.2)	Send & Byz: $2t_{\mathsf{snd}} + 2t_{\mathsf{byz}} < n$	$O(n)$
GP [11]	Crash & Byz: $t_{\mathsf{cra}} + 3t_{\mathsf{byz}} < n$	$O(n)$
ZHM [23]	Receive, Send, Byz: $t_{\mathsf{snd}} + t_{\mathsf{rcv}} + 3t_{\mathsf{byz}} < n$	$O(n)$
This paper	Receive, Spotty Send, Byz: $t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}} < n$	$\widehat{O(1)}$
This paper	Receive, Send, Byz: $t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}} < n$	$\widehat{O(1)}$

is to adapt the protocol by Zikas, Hauser and Maurer (ZHM) [23] to an expected constant-round protocol using the standard construction [8, 14] via graded consensus and a common coin protocol. The protocol by ZHM is a linear-round protocol because it depends on the phase-king paradigm [10]; the protocol must run long enough to guarantee that the king is honest in at least one round. To move to an expected-constant round protocol, phase king is replaced with a common coin primitive; however, all common coin constructions that we know require some threshold scheme. Threshold schemes work in our model when $n - t_{\mathsf{rcv}} > 2(t_{\mathsf{snd}} + t_{\mathsf{byz}})$, meaning there are more honest parties than send-corrupt or byzantine parties. In the dishonest majority setting where send-corrupt plus byzantine parties outnumber honest parties, the construction suffers from the partitioning attack described above: a group of send-corrupt parties reach the threshold independently of and without knowledge of honest parties, and honest parties therefore output a different coin than send-corrupt parties. The ZHM construction and corruption bound therefore fail in sublinear rounds.

Expected Constant-Round Consensus Protocols. There are a number of expected constant-round consensus protocols for the honest-majority setting that consider only byzantine faults. Feldman and Micali [8] gave an expected constant round scheme for $n > 3t_{\mathsf{byz}}$. Katz and Koo [14] later gave a protocol tolerating $n > 2t_{\mathsf{byz}}$, assuming a PKI and signatures. Micali [18] gave another simple protocol assuming $n > 3t_{\mathsf{byz}}$. Abraham et al [2] gave the most efficient scheme and tolerate a strongly rushing, adaptive adversary for $n > 2t_{\mathsf{byz}}$.

Mixed Corruption Models. In Table 1 we overview the results most relevant to our work: consensus protocols in mixed corruption models. We include a construction by modifying Dolev-Strong broadcast (Section 3.2) via the reduction of consensus to broadcast.

To our knoweldge, our "spotty" send-corrupt protocol exceeds the corruption bounds of all comparable models with "exotic" corruptions, who always require that a majority of online nodes are honest. For example, recent work has generalized crash corruptions into "sleepy" [20] or "sluggish" [13] faults. In the sluggish model [13], a (mobile) sluggish party can be temporarily disconnected from honest parties due to network partition, but can later rejoin. While disconnected, messages sent by or to a party are delayed until the party is reconnected. However, in that work it is (inherently) required that at least half of the parties are not sluggish and participate in the protocol at all times, and the adversary is static. This is a sharp contrast to our model, which allows a majority of dishonest parties and an adaptive adversary. Abraham et al [3], also in the sluggish model, require a majority of online parties to be honest at all times.

Pass and Shi [20] introduce a model in which the adversary can make parties "fall asleep" and later wake them up (i.e., temporarily crash them) at which point all messages that they missed are delivered at once, along with potentially some adversarially-inserted messages. They show that in their model, a protocol requires only that a majority of the *awake* parties are honest, which closely resembles our result. However, in the sleepy model, there are no send-or-receive-corruptions, meaning all awake parties are full participants in the protocol, so their sends always succeed and no incoming messages are dropped; this avoids the difficulties studied in this paper.

Malkhi et al [17] consider yet another mixed model of corruption, but require that a majority of online players behave honestly at any time. The protocol of Garay and Perry [11], mentioned above, runs in $O(n)$ round complexity, but only works when $n > 3t_{\mathsf{byz}} + t_{\mathsf{cra}}$.

1.5 Paper Outline

The rest of the paper is organized as follows: Section 2 covers preliminaries and definitions required in the rest of this paper. Section 3 discusses the pathology of send corruptions by illustrating how common paradigms for broadcast fail for send-corrupt parties. Section 4 introduces our new expected-constant round consensus protocol for send and receive corruptions. In Section 5, we show that the construction in Section 4 has improved corruption tolerance in the spotty send model, and prove its optimality.

In Appendix A we recall the lowerbound proof by Dolev and Strong for round complexity of deterministic broadcast and their protocol for authenticated broadcast. In Appendix B we give a (folklore) construction of consensus from broadcast that completes the reduction of optimal fault tolerance for consensus in the presence of (general) send corruptions to optimal fault tolerance of broadcast in the presence of send corruptions.

2 Model and Definitions

2.1 Model

In this work we consider a set of n parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ who may send and receive messages over a network. A protocol specifies the messages that parties send to each other, how they change their internal states, and how they produce their outputs. An execution of a protocol proceeds in a series of time steps, in which in each step each party may send and receive messages. In each time step in which a party receives and sends messages, first it receives all messages and then it sends messages. We assume that all parties start an execution at the same time and have internal clocks that advance at the same rate.

Network. We assume that the network is managed by an adversary that is constrained by synchronization requirements. Parties are connected via peer-to-peer authenticated channels. We assume a synchronous network; this means there is a known bound Δ such that during normal operation, any message that is sent at time t must be delivered to its intended recipient by time $t + \Delta$. Within this bound, the adversary may arbitrarily schedule delivery of messages via peer-to-peer channels, subject to the constraint imposed by Δ.

Corruptions. The adversary may adaptively corrupt parties that participate in an execution. We allow an adversary to corrupt a party in one of three modes, which we describe in the following. A party that is not corrupted must follow the protocol specification and is called *honest*. Once a party is corrupted, it may not become honest again.

– A *receive* corruption allows the adversary to selectively drop messages sent to the party.
– A *send* corruption allows the adversary to selectively drop messages sent by the party.
– A *byzantine* corruption allows the adversary to control all messages sent by the party and view its internal state.

Where we categorize send corruptions in two types:

1. A send corruption allows the adversary to drop arbitrary messages sent by the party without constraint.
2. A *spotty* corruption allows the adversary to drop messages sent by the party, although the party continues to follow the protocol specification (and the adversary does not get to see its state). Specifically, the adversary may adaptively drop all undelivered messages sent by a spotty party p by issuing an instruction (drop, p) to the network. The drop instruction is constrained such that if any message sent by p within the last Δ time has been delivered, then the instruction fails. However, the adversary may wait for a spotty party to send all of its messages in any span of Δ time (without delivering them to their recipients) before choosing to drop them all. Because all messages sent by a party within the last Δ time must either be delivered or dropped, we say that the adversary must *uniformly* drop or deliver messages for the party.

We note that the adversary does not need to corrupt both a sender and a receiver in order to drop a message between them; it suffices for the adversary to corrupt only one of them. We also do not require that the sets of send-corrupt and receive-corrupt parties are disjoint. However, any party that is both send-corrupt and receive-corrupt is counted as both a send corrupted party and a receive corrupted party.

Strongly Rushing Adversary. We consider an adversary that is *strongly rushing*, similar to that of [2, 1], but we extend it to drop messages from send-corrupt parties. In our model, a *strongly rushing* adversary is permitted to read messages that are sent by an honest party over the network and then choose to corrupt the party in the same time step. If the adversary chooses to send-corrupt the party, then it can drop messages sent by the party in that step; similarly, if the adversary chooses to receive-corrupt the party, then it can drop messages sent to the party in that step. In either case, the party is send- or receive-corrupted from that step forth. If the adversary chooses to byzantine corrupt a party in some step, it removes all messages sent by the party at that time step. The adversary then chooses what messages the party sends in that step, and to which parties it sends what messages. The corrupted party is byzantine corrupted from that time forth.

2.2 Digital Signatures and Coin Flipping

Our constructions require the use of a digital signature scheme. In particular, we assume that parties have access to a public key infrastructure (PKI) for a digital signature scheme, meaning each party is aware of a set of public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, where pk_i is associated with p_i for $i \in [n]$. We consider that all parties choose their own public and private keys; in particular, some parties may adversarially choose their key pairs. Our constructions will assume an idealized signature scheme for which signatures are perfectly unforgeable; when using signature schemes that achieve unforgeability against computationally bounded adversaries, our protocols achieve our desired properties except with negligible probability.

Additionally, our construction requires the use of an unbiasable coin flipping protocol Π^{coin}. Our protocols assume idealized access to such a primitive, which may be considered to be implemented by an ideal functionality that takes no input (or more formally, takes as input the empty string) and delivers a uniformly random bit to all parties. At a high level, we require that:

- Until at least one live party queries Π^{coin} in the r-th invocation, the output for that invocation is uniformly distributed for the adversary.
- All live parties output the same value in Π^{coin}.

Such a coin flipping protocol may be instantiated (assuming a trusted setup) by augmenting threshold signatures [16] (using threshold $t_{\mathsf{byz}} + 1$, see below) with a protocol for reliable sends in our model, such as FixReceive ([23], or ours below).

2.3 Defining Broadcast and Consensus

We provide new definitions for the considered mixed model by adapting the standard definitions of consensus protocols and constraining the behavior of all live parties, including send-corrupt parties and receive-corrupt parties that have not become zombies. Note that our definitions quantify over all the inputs of live parties that participate starting at the beginning of an execution, and over the outputs of only parties that are not zombies by the end of the execution.

Towards the definitions, we introduce thresholds on the number of corruptions that we permit the adversary to make per execution. We use t_{snd}, t_{rcv}, and t_{byz} to denote thresholds on the number of send, receive, and byzantine corruptions, respectively, in an execution. We introduce the following definition of an execution in which some parties may be corrupted in order to facilitate the definitions of our consensus problems.

Definition 1 (($t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}}$)-Compliant Execution). *For a protocol Π, we say that an execution of Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ compliant if at most t_{snd}, t_{rcv}, and t_{byz} parties are send-corrupted, receive-corrupted, and byzantine-corrupted, respectively, in the execution.*

We now define broadcast and binary consensus in our model. Our constructions will make use of weaker primitives; we define those as necessary for the constructions.

Broadcast In a broadcast protocol, a dealer $D \in \mathcal{P}$ wishes to send a message $m \in \{0,1\}^*$ to the parties in $\mathcal{P}$. Each party $p \in \mathcal{P}$ outputs a message $m' \in \{0,1\}^* \cup \{\perp\}$, subject to the following constraints:

Definition 2 (Broadcast). *Let Π be a protocol for parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ in which a distinguished party $D \in \mathcal{P}$ holds an input $m \in \{0,1\}^*$. Π is a* Broadcast with Unanimity for Send-Corruptions *protocol if the following properties hold except with negligible probability.*

1. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-**Validity:** Π is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which D is honest or receive corrupt (but not send-corrupt), every live party outputs m.*
2. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-**Consistency:** Π is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which any live party outputs $m' \in \{0,1\}^* \cup \{\perp\}$, every live party outputs m'.*
3. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-**Termination:** Π is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution, every live party outputs some $m' \in \{0,1\}^* \cup \{\perp\}$ and terminates within finitely many steps.*

If Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid, $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent, and $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating then we call it $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure.

Consensus In a (binary) consensus protocol, each party has an input $b \in \{0,1\}$. Each party is expected to output a bit $v \in \{0,1\}$.

Definition 3 (Consensus). *Let Π be protocol for parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ in which each party has an input $b \in \{0,1\}$. Π is a* Consensus *protocol if the following properties hold except with negligible probability.*

1. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$***-Validity:*** *$\Pi$ is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid *if in every* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-compliant execution in which all live parties have the same input $b \in \{0,1\}$, all honest parties output b.*
2. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$***-Consistency:*** *$\Pi$ is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent *if in every* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-compliant execution in which any live party outputs v, every live party outputs v.*
3. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$***-Termination:*** *$\Pi$ is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating *if in every* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-compliant execution, every live party outputs $v \in \{0,1\}$ and terminates within finitely many steps.*

If Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid, $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent, and $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating then we call it $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure.

3 On the Difficulty of Optimal Corruption Tolerance for Send-Corrupt Parties

In this section we discuss the pathology of "standard" send corruptions with respect to current techniques in the literature, and describe why send corruptions appear as deleterious as full byzantine corruptions. Although our focus is on consensus protocols, we consider techniques for both consensus and broadcast; the two are related by a (folklore) reduction, which we discuss in Appendix B.

1. We first recall the proof by Dolev and Strong that any deterministic broadcast protocol requires at least $t_{\mathsf{byz}} + 1$ rounds (for at most t_{byz} byzantine corruptions), and we show that the impossibility result immediately requires that when send-corrupt parties exist, any deterministic broadcast protocol requires at least $t_{\mathsf{snd}} + 1$ rounds (for at most t_{snd} send-corruptions).
2. We show that the Dolev-Strong broadcast protocol fails as written when considering send corruptions. We modify the protocol and show that without new ideas, its corruption threshold degrades from $n > t_{\mathsf{byz}}$ (in the original model) to $n > 2(t_{\mathsf{snd}} + t_{\mathsf{byz}})$.
3. We visit recent techniques for security against strongly rushing, adaptive adversaries – who have the ability to adaptively remove messages from the network after they have been sent – and show that these also fall short of a corruption threshold better than $n > 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ (which our construction in Section 4 achieves). One might expect these techniques would apply to send-corrupt parties because of the adversary's ability to adaptively drop messages from a majority of parties. However, the techniques fail when requiring send-corrupt parties' outputs to be consistent with honest parties' outputs.

3.1 Dolev and Strong's Lowerbound with Send Corruptions

The classical lowerbound for deterministic broadcast by Dolev and Strong transfers directly to the case of send corruptions, applying the observation (discussed in their paper) that all of the byzantine parties in their proof are constrained to dropping messages that should be sent, but otherwise behave honestly. We provide a full exposition of Dolev and Strong's lowerbound in Appendix A.2. We restate the theorem here.

Theorem 1 (Dolev and Strong [7]). *There is no deterministic broadcast protocol tolerating t_{snd} send corruptions which terminates in fewer than $t_{\mathsf{snd}} + 1$ rounds, even assuming an idealized PKI and signatures.*

We remark that there has been recent work by Chan, Pass, and Shi [6] to extend the lowerbound by Dolev and Strong to randomized protocols. Because their adaptation also requires only dropping sent messages, their lowerbound also directly transfers to the send-corrupt model.

3.2 Modifying Dolev-Strong Broadcast

As an example of the pathology of send-corruptions, we now recall the classical authenticated broadcast protocol by Dolev and Strong [7]. Because the protocol is canon, we defer the original to Appendix A.1 but review it here.

The protocol uses a data structure that we will call a sig-chain. A 1-sig-chain is a pair (m, σ), where σ is a signature on string m. For $i > 1$, an i-sig-chain is a pair (m, σ), where m is an $(i-1)$-sig-chain and σ is a signature on m. A *valid* i-sig-chain is a sig-chain with the property that no two signatures in the sig-chain are computed using the same key. An i-sig-chain *contains* a message m' if m' is the message of the 1-sig-chain on which the sig-chain is built.

The protocol operates as follows: In the first round, the dealer creates a 1-sig-chain containing its input and sends the sig-chain to all parties. In every subsequent round i, any party that received a valid $i-1$ chain in the previous round that did not contain a signature that it had computed creates an $i-1$ sig-chain by appending its own signature to the chain. It then sends the i-sig-chain to all parties. In any round i, if a party receives a valid i-sig-chain, then it adds the message m contained in the sig-chain to a set of candidate outputs. If the set of candidate outputs contains only one candidate at the end, then the party outputs that message. Otherwise it outputs $\perp$.

Where Dolev-Strong Fails. In the send-corruption model, the Dolev-Strong protocol fails because it is possible for send-corrupt parties to output some message m while honest parties output $\perp$. Consider an execution in which the parties are partitioned into three sets: H contains all of the honest parties, S contains all send-corrupt parties, and B contains all byzantine parties. (For the sake of this argument, we need not consider receive-corrupt parties.) Let the dealer be send-corrupt. It is possible that in this execution, the send-corrupt parties communicate only with parties in $S \cup B$. Then send-corrupt and byzantine parties can collectively build a $t_{\mathsf{byz}} + 1$-chain containing m and no honest parties ever receives the dealer's message or any sig-chain containing the message. But this violates consistency, because all send-corrupt and honest parties are required to output the same thing.

Protocol 1 Modified Dolev Strong Broadcast Protocol Π^{modDS}

Shared Setup: Public Key Infrastructure (PKI) for a signature scheme.
Inputs: The dealer $D \in \mathcal{P}$ has an input $m \in \{0,1\}^*$.
Outputs: Each party $p \in \mathcal{P}$ outputs a value $m' \in \{0,1\}^* \cup \{\bot\}$.
Local Variable: Each party $p \in \mathcal{P}$ maintains a local variable S, which is a set initialized to $\{\}$.
Protocol: The protocol begins at time 1 and proceeds in rounds. Each round party p proceeds as follows:

1. **Round 1: Dealer's Messages** The Dealer D signs its input $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(m)$ and sends (m, σ) to all parties.
2. **Sig Chains:** For every round i from 2 to $t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1$: For every valid $(i-1)$-sig-chain c that p received at the end of round $i-1$ in which none of the signatures were constructed by p, p computes $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(c)$ and sends (σ, c) to all parties.
3. **Output:** For every valid $(t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1)$-sig-chain c that p received at the end of round $t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1$, let m' be the message contained by c and update $S = S \cup \{m'\}$. If $|S| = 1$, then p outputs the element $m' \in S$. If $|S| \neq 1$, then p outputs $\bot$.

Fig. 1. Modified Dolev-Strong Broadcast Protocol Π^{modDS}

Modifications. In order to resolve this problem, we must make two modifications to the protocol. First, a party must receive an $t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1$-sig-chain for any message that it will output; no chain of less than $t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1$ length may add a message to the set of candidate outputs. (This additionally requires that the protocol is run for $t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1$ rounds.) Second, we update the bounds to require that $n > 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$. A majority of honest parties is necessary to ensure that honest parties can always build a $t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1$-sig-chain without the assistance of byzantine or send-corrupt parties, which is necessary for validity.

We present our modified Dolev-Strong protocol (Π^{modDS}) in Figure 1.

Theorem 2. *Π^{modDS} is a $(t_{\mathsf{snd}}, t_{\mathsf{byz}})$-secure broadcast protocol for $n > 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.*

Proof. The proof is similar to the original by Dolev and Strong, subject to modifications described above. Validity follows from the fact that when $n > 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ and the dealer is honest, the honest parties build a $(t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1)$ sig-chain, and that no sig-chain can exist containing some m' that the dealer did not send. Consistency follows from the fact that if a $(t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1)$ sig-chain exists, then some honest party's signature must be included. It follows that if any honest party output m, then all honest parties receive a $(t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1)$ sig-chain containing m. Assume that some honest party receives a $(t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1)$ sig-chain containing m and another honest party receives a $(t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1)$ sig-chain containing m'. Then both sig-chains must include an honest signature, and therefore there must be $(t_{\mathsf{snd}} + t_{\mathsf{byz}} + 1)$ sig-chain containing m and m' in the view of every honest party. It follows that every honest and send-corrupt party outputs $\bot$.

Can Dolev-Strong Be Fixed to Support $n > t_{\mathsf{snd}} + t_{\mathsf{byz}}$? We have shown that without new ideas, Dolev-Strong cannot be updated to tolerate $n > t_{\mathsf{snd}} + t_{\mathsf{byz}}$ (which it is easy to prove is an optimal corruption budget). However, we cannot rule out such a threshold. In the pathological execution described above, honest parties do not send any messages if they do not receive any valid sig-chains. However, honest parties may send messages in each round containing $\perp$, indicating "I have not received a message," which conveys that the party's sent message *was not dropped.* This provides more information to the protocol, but we do not know how to use such a technique to improve broadcast.

3.3 Recent Techniques for Adaptive, Strongly Rushing Adveraries

We explain that recent techniques for byzantine agreement and broadcast against a strongly rushing adversary also fail when requiring consistency between send-corrupt parties' outputs and honest parties' outputs, even when the adversary is not strongly adaptive. For example, the byzantine agreement protocol by Abraham et al [2] and the broadcast protocol by Wan et al [21] achieve security against a strongly adaptive adversary by effectively committing to any leader's messages early in the protocol, and then revealing a leader in a later round. This thwarts strongly rushing adaptive adversaries because by the time a leader is elected, it is too late to corrupt the leader and remove the messages it has sent.

In the partitioning attack, send-corrupt parties are able to communicate with each other but not with the honest parties, and are able to reach signature thresholds on messages that no honest party ever receives. For example, in [2], messages often require $b + 1$ distinct signatures (implying at least one honest party signed a message) in order to be recognized by an honest party. But when there are more send-corrupt parties than honest parties, any threshold number of signatures that honest parties must be able to attain on their own must also be attainable by send-corrupt parties only. This can cause send-corrupt parties to adopt a different leader in some step than the honest parties. Similarly, in [21], send-corrupt parties' puzzles may never be delivered to honest parties. When honest parties choose a leader based on the solutions to a set of time-lock puzzles, send-corrupt parties may make a decision based on a larger set than the honest parties, and their decisions may differ. This form of attack is prevented by the implicit echoing assumption in [21], but it does not carry into the send-corrupt model. In our model, this attack is thwarted by requiring the number of honest parties be greater than $2(t_{\mathsf{snd}} + t_{\mathsf{byz}})$, as thresholds on the number of signatures can enforce that some honest party signs a message.

4 Constant-Round Synchronous Consensus for $n > t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$

We now present a protocol for consensus in synchronous networks in the presence of send corruptions, receive corruptions, and byzantine corruptions where digital signatures are available. We prove that the protocol is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure for

Protocol 2 All-To-All FixReceive Protocol $\Pi^{FR}(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$

Inputs: Each party $p \in \mathcal{P}$ has an input $m \in \{0,1\}^*$.
*Outputs:*Each party $p \in \mathcal{P}$ outputs some message for every other party in $\mathcal{P}$, or outputs zombie.
Protocol: The protocol proceeds in two rounds, in which every party sends its input m to every other party, and then parties forward the unique messages they have received, as follows:

1. **Send Messages:** Each party sends its signed input m to every other party.
2. **Replay:** Every party forwards every unique message that it received in Round 1 to every other party. If a party did not receive any unique messages in Round 1, it sends $\perp$ to every other party.
3. **Output:** If a party p does not receive more than $n - t_{\mathsf{snd}} - t_{\mathsf{byz}} > t_{\mathsf{byz}} + t_{\mathsf{rcv}}$ messages (including $\perp$) in either round, then it sends zombie to all other parties, outputs $\perp$, and becomes a zombie. Otherwise, p outputs the set of unique messages that it received in Round 2.

Fig. 2. All-to-all FixReceive Protocol Π^{FR}

$n > t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$. In Section 5, we show the same protocol is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ when send corruptions are *spotty*, and that corruption budget is optimal.

Towards presenting our consensus protocol, we first present protocols for weak broadcast, weak consensus, and graded consensus. Each protocol is used as a building block in our ultimate consensus protocol. However, before these building blocks, we introduce another protocol for reliable sending when all parties send messages to each other. A party detects whether it is receive-corrupt based on the number of messages it receives; if so, it becomes a zombie and notifies the other parties. If not, it continues to participate and outputs the messages that it received.

4.1 All-To-All FixReceive

We present a protocol that is similar to FixReceive from [23], tuned for the common scenario in our future protocols, in which all parties attempt to send a message to all other parties. The parties all forward unique messages that they receive, in order to ensure that every party either receives message that was sent, or detects that it is receive-corrupted. The parties output all unique messages that they receive during the protocol.

We prove that a receive-corrupt party that does not become a zombie must receive a message from another honest or send-corrupt party. We then prove that if some honest party attempts to send a message m via the protocol, then every non-zombie party must receive that message.

Lemma 1. *Any party p becomes a zombie during Π^{FR} only when it is receive-corrupt. If p does not become a zombie then it received a message from at least one honest or send-corrupt party.*

Proof. If p receives fewer than $n - t_{\mathsf{snd}} - t_{\mathsf{byz}}$ then it must be receive-corrupt, since at most t_{snd} send-corrupt parties and t_{byz} byzantine parties may not send messages to p. If p does not become a zombie, then it must receive at least $n - t_{\mathsf{snd}} - t_{\mathsf{byz}} > t_{\mathsf{byz}} + t_{\mathsf{rcv}}$ messages. Therefore, one of the messages it received must have been from an honest or send-corrupt (but not also receive-corrupt) party.

Lemma 2. *If an honest party or receive-corrupt party (but not send-corrupt) sends a message m using Π^{FR}, then every live party receives m or becomes a zombie.*

Proof. Follows from the fact that in the first round, all honest parties and send-corrupt receive m. In the second step, if any party p does not become a zombie, then it must receive a message either some honest or send-corrupt party, which must include m.

4.2 Weak Broadcast

Our first building block is a weak broadcast primitive. In a weak broadcast protocol, a dealer $D \in \mathcal{P}$ wishes to send a message $m \in \{0,1\}^*$ to the parties in $\mathcal{P}$. Each party $p \in \mathcal{P}$ outputs a message $m' \in \{0,1\}^* \cup \{\perp\}$, subject to the following constraints:

Definition 4 (Weak Broadcast). *Let Π be a protocol for parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ and a distinguished party $D \in \mathcal{P}$ holds an input $m \in \{0,1\}^*$. Π is a* Weak Broadcast *protocol if the following properties hold except with negligible probability.*

1. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$***-Validity:*** *$\Pi$ is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-*valid *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which D is honest or receive corrupt (but not send-corrupt), every live party outputs m.*
2. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$***-Unanimity:*** *$\Pi$ is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-*unanimous *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which D is live, either every live party outputs $m \in \{0,1\}^*$ or every live party outputs $\perp$.*
3. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$***-Consistency:*** *$\Pi$ is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-*consistent *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which any honest party outputs $m' \in \{0,1\}^*$, every live party outputs m' or $\perp$.*
4. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$***-Termination:*** *$\Pi$ is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-*terminating *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution, every live party outputs some $m' \in \{0,1\}^* \cup \{\perp\}$ and terminates within finitely many steps.*

If Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid, $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent, and $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating then we call it $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure. If Π is additionally $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-unanimous, then we call it $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure with unanimity.

Protocol 3 Weak Broadcast Protocol Π^{WB}

Shared Setup: Public Key Infrastructure for a signature scheme, every party knows the identity of the dealer and its public key pk.
Inputs: The dealer $D \in \mathcal{P}$ has an input $m \in \{0,1\}^*$.
Outputs: Each party $p_i \in \mathcal{P}$ outputs a value $m' \in \{0,1\}^* \cup \{\perp\}$.
Protocol: The protocol begins at time 0 and proceeds in rounds, in which each round lasts for Δ time. Each round party p proceeds as follows:

1. **Dealer's Messages**: The Dealer D signs its input $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(m)$ and sends $(\mathsf{deal}, m, \sigma)$ to all parties, where σ is the signature on m using its secret signing key sk.
2. **Echo Dealer's Value**: Parties run Π^{FR} based on the messages they received from D. If p received a message from D, let (m', σ) be the message and signature that p received. p inputs $(\mathsf{echo}, m', \sigma)$ to Π^{FR}. Otherwise, p inputs $(\mathsf{echo}, \perp, \perp)$ to Π^{FR}.
3. **Replay:** Parties again run Π^{FR} based on the messages they received in the previous round, where each party provides all of the unique messages it received in the previous Π^{FR} as input.
4. **Verification and Output**: If p did not output any messages signed with D's key from the first run of Π^{FR}, then it outputs $\perp$. If in the outputs of the second run of Π^{FR}, p receives any two pairs (m'_i, σ_i) and (m'_j, σ_j) such that $m'_i \neq m'_j$ but $\mathsf{ver}_{\mathsf{pk}}(\sigma_i) = 1$ and $\mathsf{ver}_{\mathsf{pk}}(\sigma_j) = 1$, then p outputs $\perp$. Otherwise, p outputs the unique message m' that it received in the first run of Π^{FR} whose signature verifies with D's public key.

Fig. 3. Weak Broadcast Protocol Π^{WB}

Our protocol for weak broadcast is presented in Figure 3. It follows a standard construction, adapted for our corruption model by invoking Π^{FR} to distribute messages. It permits a designated *dealer* to send an arbitrary message m to all parties, with the guarantee that every party outputs either m or $\perp$.

Lemma 3. *Protocol $\Pi^{\mathsf{WB}}(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ is a $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure weak broadcast protocol for $n > t_{\mathsf{snd}} + t_{\mathsf{rcv}} + 2t_{\mathsf{byz}}$.*

Proof. Termination is trivial. We prove validity and consistency.

$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-Validity: If the dealer is honest or receive-corrupt (but not send-corrupt) then every honest party receives a valid signature on the dealer's input m from the dealer. Then, by Lemma 2, all non-zombie parties output m from the first run of Π^{FR}. By the unforgeability of our idealized signature scheme, no signed message m' under the dealer's key can be forged. Therefore, every live party outputs m.

$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-Consistency: Assume that honest party p outputs m and live party q outputs m'. Then p forwards m and its signature to q via the second invocation of Π^{FR}. By Lemma 2, q must output m and its signature from the second invocation of Π^{FR}, or become a zombie. Because q is not a zombie, it received a signed message containing m that verifies with the dealer's public key, and therefore does not output m', a contradiction.

We provide an additional statement about the outputs of Π^{WB} when the sender is corrupt but not byzantine. Specifically, consistency holds over the outputs of all live parties when the dealer is send-corrupt (and not only when some honest party outputs $m \neq \bot$).

Lemma 4. *When the dealer is send-corrupt, if one live party outputs $m \neq \bot$, then every live party outputs $m' \in \{m, \bot\}$*

Proof. Follows directly from unforgeability of the idealized signature scheme.

4.3 Weak Consensus

We use weak consensus as a stepping stone to achieve consensus. In a weak consensus protocol, all honest parties have an input $b \in \{\bot, 0, 1\}$, and all honest parties are expected to output a value $v \in \{\bot, 0, 1\}$, subject to the following:

Definition 5 (Weak Consensus). *Let Π be a protocol for parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ in which every party $p \in \mathcal{P}$ has an input $b \in \{0, 1\}$. Π is a* Weak Consensus *protocol if the following properties hold except with negligible probability.*

1. *$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-**Validity:** Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-valid *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which all honest parties have the same input b and no live parties have input $1-b$, all honest parties output b.*
2. *$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-**Consistency:** Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-consistent *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which any live party outputs $v \in \{0, 1\}$, no live party outputs $1 - v$.*
3. *$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-**Termination:** Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-terminating *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution, every live party outputs $v \in \{\bot, 0, 1\}$ and terminates within finitely many steps.*

If Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid, $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent, and $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating then we call it $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure.

We present our weak consensus protocol Π^{WC} in Figure 4. The protocol is an adaptation of the reduction from Weak Consensus to Weak Broadcast [9], modified for our corruption setting. Specifically, the protocol proceeds in two synchronous rounds. First, in parallel, each party signs its protocol input and sends its signed input to all parties. Second, upon receiving all other parties' inputs, each party attempts to generate a *certificate* in favor of some output value. A certificate for a bit u is a set of $n - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - t_{\mathsf{byz}}$ unique, valid signatures on u. If a party is able to generate a certificate, it sends the certificate to all other parties.

Protocol 4 Weak Consensus $\Pi^{\mathsf{WC}}(t_{\mathsf{cra}}, t_{\mathsf{byz}})$

Shared Setup: Public Key infrastructure for a signature scheme.
Inputs: Each party $p \in \mathcal{P}$ has an input $b \in \{\perp, 0, 1\}$ and a secret signing key for the signature scheme.
Outputs: Each party $p \in \mathcal{P}$ outputs a value $v \in \{\perp, 0, 1\}$.
Protocol: The protocol begins at time 0 proceeds in rounds, in which each round lasts for Δ time. Each party p_i proceeds as follows:

1. **Sign Inputs:** In parallel, each party signs its input bit and sends its signed input to all other parties.
2. **Construct Certificates and WB:** Each party collects all of the signed input bits from the other parties. If there is a $v \in \{0, 1\}$ for which $n - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - t_{\mathsf{byz}}$ valid signed messages are received, p constructs a *certificate* composed of $n - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - t_{\mathsf{byz}}$ signatures from distinct parties on v. The parties then invoke n weak broadcasts in parallel, in which p_i is the dealer in the ith weak broadcast, and p_i provides its certificate as input if it has one; otherwise p_i provides $\perp$ as its input.
3. **Output:** Each party receives any certificates sent to it in Round 2. If p constructed a certificate for some v in round 2 AND p has received at least $n - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - t_{\mathsf{byz}}$ certificates for v by the end of round 2 from distinct parties AND p has not received a valid certificate for $1 - v$, then p outputs v. Otherwise, p outputs $\perp$.

Fig. 4. Weak Consensus Protocol Π^{WC}

A party outputs a bit v only if it meets three conditions: First, it must generate a certificate in the beginning of the second round; second, it must receive at least $n - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - t_{\mathsf{byz}}$ valid certificates from distinct parties; third, it must not receive a valid certificate for $1 - v$ from any other party. Otherwise it outputs $\perp$.

Intuitively, validity of the protocol is guaranteed by the fact that if all live parties have input b, then all honest parties will be able to construct a certificate for b, and there will not be enough corrupt parties to construct a certificate for $1 - b$. Consistency is guaranteed by the fact that if two live parties are able to generate certificates for opposite values, then they must share their certificates with each other, and then both output $\perp$.

Lemma 5. *Protocol $\Pi^{\mathsf{WC}}(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ is a $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure Weak Consensus protocol in synchronous networks for $n > t_{\mathsf{rcv}} + \frac{3}{2}t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.*

Proof. Termination is trivial. We separately prove validity and consistency.

$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-Validity : If all honest parties have input b, then because there are at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ honest parties, every honest party receives at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ signatures on b in the first round. It follows that at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ honest parties construct valid certificates for v and weak broadcast them to all live parties. By validity of Π^{WB}, all of these weak broadcasts are received by all live parties. Moreover, because no live parties have input $1-b$ and $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}} > t_{\mathsf{byz}}$, no certificate can be constructed by corrupt parties for $1-b$. Therefore, every live party outputs b.

$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-Consistency : Assume live party p outputs v and live party q outputs $1-v$. Then p must have received at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ certificates for v and q must have received at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ certificates for $1-v$.

Let A be the set of parties from which p received a certificate and B be the set of parties from which q received a certificate. Note that by validity and Lemma 4, no live party in A may also be in B, or vice versa; otherwise, q (respectively p) received a certificate for v (respectively $1-v$), and q (respectively p) did not output $1-v$ (respectively v). Therefore, only corrupt parties may be in both p and q, and there are at most b of them.

We proceed toward contradiction by showing that in fact, there must be some honest or receive-corrupt party in both A and B. We do so by arguing about the size of the union of A and B. If $|A \cup B| > t_{\mathsf{snd}} + t_{\mathsf{byz}}$, then there must be some honest or receive-corrupt party that weak broadcasted the same certificate to p and q, and by validity both p and q received that certificate. This is sufficient to conclude the proof, because then p or q does not output v or $1-v$, as argued above.

Recall that $|A \cup B| = |A| + |B| - |A \cap B|$. We have argued that $|A| \geq n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$, $|B| \geq n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$, and $|A \cap B| \leq t_{\mathsf{byz}}$. Then $|A \cup B| \geq 2(n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}) - t_{\mathsf{byz}}$, and when $n > t_{\mathsf{rcv}} + \frac{3}{2}t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$, $|A \cup B| > t_{\mathsf{snd}} + t_{\mathsf{byz}}$. As explained above, this is a contradiction.

4.4 Graded Consensus

We define an additional weakened form of consensus called *graded consensus*, which was originally introduced by Feldman and Micali [8]. In a graded consensus protocol, each party has an input $b \in \{0,1\}$. Each party is expected to output a pair $(v,g) \in \{0,1\}^2$, where v is the output bit and g is a *grade*.

Definition 6 (0/1 Graded Consensus). *Let Π be a protocol for parties $\mathcal{P} = \{p_1, \dots, p_n\}$ where each party has input $b \in \{\bot, 0, 1\}$. Π is a* 0/1 Graded Consensus *protocol if the following properties hold except with negligible probability.*

1. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-***Validity:*** *$\Pi$ is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which all honest parties have the same input $b \in \{0,1\}$ and no live parties have input $1-b$, all live parties output $(b,1)$.*
2. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-***Consistency:*** *$\Pi$ is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution in which any live party outputs $(v,1)$, every live party outputs $(v,g) \in \{0,1\}^2$.*
3. $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-***Termination:*** *$\Pi$ is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating *if in every $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution, every live party outputs $(v,g) \in \{0,1\}^2$ and terminates within finitely many steps.*

If Π is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid, $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent, and $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-terminating then we call it $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure.

Protocol 5 Graded Consensus $\Pi^{\mathsf{GC}}(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$

Inputs: Each party $p \in \mathcal{P}$ has an input $b \in \{\bot, 0, 1\}$
Outputs: Each party $p \in \mathcal{P}$ outputs a pair $(v, g) \in \{0, 1\}^2$
Protocol: The protocol begins at time 0 and proceeds in synchronous rounds, labeled below, where each round lasts long enough for its corresponding subprotocol to complete. Each party p proceeds as follows:

1. **Weak Consensus:** Run Π^{WC} with b as input. Let b' denote the output of Π^{WC}.
2. **Weak Broadcast:** In parallel, all parties invoke n copies of $\Pi^{\mathsf{WB}}(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$, where p_j is the dealer in the jth copy. p_j uses the value b' as its input to Π^{WB}. For $u \in \{\bot, 0, 1\}$, let n_u denote the number of weak broadcasts for which p outputs u.
3. **Output:**
   – Assign $v \leftarrow u \in \{0, 1\}$ for which $n_u > n_{1-u}$. Break ties by assigning $v \leftarrow 1$. Assign $g \leftarrow 1$ if $n_v \geq n - t_{\mathsf{byz}} - t_{\mathsf{rcv}} - t_{\mathsf{snd}}$. Else $g \leftarrow 0$. Output (v, g)

Fig. 5. Graded Consensus Protocol Π^{GC}

Our graded consensus protocol Π^{GC} is presented in Figure 5; it is an adaptation to our fault model of the reduction of graded consensus to weak broadcast discussed by Fitzi [9]. Specifically, Π^{GC} proceeds in synchronous rounds in which two subprotocols are invoked. First, parties invoke a weak consensus protocol, using their protocol inputs as input to the weak consensus protocol. Second, in parallel, all parties weak broadcast their outputs from the weak consensus protocol. Parties determine their outputs based on the weak broadcasts they receive. First, a party sets the bit v to the value $u \in \{0, 1\}$ for which it received more weak broadcasts carrying u than $1 - u$. Second, a party sets its grade g to 1 if it receives than $n - t_{\mathsf{byz}} - t_{\mathsf{rcv}} - t_{\mathsf{snd}}$ weak broadcasts carrying bit v, and sets its grade to 0 otherwise. It then outputs (v, g). Intuitively, each party outputs a bit v based on the majority of weak broadcasts that it has received. A party outputs grade 1 if it has received a large enough majority of weak broadcasts carrying v that it is guaranteed no other honest party has received a majority of weak broadcasts carrying $1 - v$. The proof follows from a standard quorum argument.

Lemma 6. *Protocol $\Pi^{\mathsf{GC}}(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ is a $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure graded consensus protocol in synchronous networks for $n > t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.*

Proof. Termination is trivial. We separately prove validity and consistency.

$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-Validity: By the validity of Π^{WC}, if all honest parties have the same input $b \in \{0, 1\}$ and no live parties have input $1 - b$, then every live party outputs b from Π^{WC}, and no live party outputs $1 - b$ from Π^{WC}. Next, every honest party weak-broadcasts b via Π^{WB}, so there are at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ weak broadcasts from which each honest party outputs b. Moreover, because no live party outputs $1 - b$ from Π^{WC}, there are at most t_{byz} executions of weak broadcast from which any party outputs $1-b$. Because $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}} > t_{\mathsf{byz}}$ (by assumption), each honest party outputs b as its value. Because at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ honest parties weak broadcasted b, each live party outputs 1 as its grade.

$(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-Consistency: Suppose a live party p_i outputs $(v, 1)$ for $v \in \{0, 1\}$ and a live party p_j outputs $(1-v, g)$ for some $g \in \{0, 1\}$.

First we establish that no live party weak-broadcasted $1-v$ in Round 2. It must be the case that p_i output v from at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ parties in Round 2. Because $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}} > t_{\mathsf{byz}}$, there must be some live party that sent v to p_i. Let this honest party be q. It must therefore be the case that q output v from the execution of Π^{WC}. By the consistency of Π^{WC}, no live party output $1-v$ from Π^{WC}, and therefore no live party weak-broadcasted $1-v$.

Next, consider that p_i received at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}}$ weak broadcasts of v. We now consider the view of p_j

1. In at most s weak broadcasts, the dealer was send-corrupt; therefore p_j output $\perp$ from at most s of the broadcasts with live dealer from which p_i output v.
2. Let b^* be the number of weak broadcasts with byzantine dealer from which p_i output v. By consistency of weak broadcast, p_j must output v or $\perp$ from those weak broadcasts.

Because p_j output $(1-v, g)$ it must be the case that $n_{1-v} > n_v$ in p_j view. By the above statements, n_v must be at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - s - b^*$ in p_j's view. Because only byzantine parties may have weak broadcasted $1-v$, and because b^* corrupt parties did not weak broadcast $1-v$, n_{1-v} is at most $t_{\mathsf{byz}} - b^*$ in p_j's view. This is a contradiction because $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - s - b^* > t_{\mathsf{byz}} - b^*$ when $n > r + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$, and therefore $n_v > n_{1-v}$ in p_j's view and p_j did not output $(1-v, g)$. This is a contradiction.

4.5 Expected Constant Round Consensus

In Figure 6 we present Π^*, our expected-constant round protocol for consensus. The protocol follows the standard coin-loop paradigm to go from graded consensus to byzantine agreement. To ensure termination, the protocol ensures that when a party terminates, it holds a certificate that it can send to all parties in order to make them terminate with the same value.

Theorem 3 (Main Theorem). *$\Pi^*(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ is a $\Pi^*(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure consensus protocol in synchronous networks for $n > \mathsf{t_{rcv}} + 2\mathsf{t_{snd}} + 2\mathsf{t_{byz}}$, where a common coin primitive is available.*

We proceed to prove the theorem via a sequence of lemmas. We start with validity.

Lemma 7 (Validity). *$\Pi^*(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-valid in synchronous networks for $n > \mathsf{t_{rcv}} + 2\mathsf{t_{snd}} + 2\mathsf{t_{byz}}$.*

Protocol 6 Expected Constant Round Protocol $\Pi^*(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$

Common Setup: The parties have access to a public key infrastructure for some signature scheme.
Inputs: Each party $p \in \mathcal{P}$ has an input $b \in \{0, 1\}$
Outputs: Each party $p \in \mathcal{P}$ outputs some $b' \in \{0, 1\}$
Internal Variable: Each party maintains a variable $v \in \{0, 1\}$ which is initialized to b. For each $u \in \{0, 1\}$, each party also maintains a set D_u of distinct (decide, u) messages that it has received.
Protocol: The protocol begins at time 0 and proceeds in synchronous rounds. Each party p proceeds as follows:

- **Loop** starting with iteration $i = 0$ until terminating:
  1. **Subround A (Graded Consensus):** Run $\Pi^{\mathsf{GC}}(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ with v as input. Let (u, g) denote p's output of Π^{GC}.
  2. **Subround B (Common Coin):** Invoke a common coin protocol Π^{coin} and assign to ψ_i the output.
  3. **Conditional Update:** If $g = 0$, then update $v \leftarrow \psi_i$. If $g = 1$, then update $v \leftarrow u$.
  4. **Conditional Decision:** If $g = 1$ and $v = \psi_i$: sign (decide, v), send the signed message to all parties, and output v.
  5. **Certificate Send:** All parties invoke Π^{FR}, where any party that has generated or received a *certificate* since the last invocation of Π^{FR} provides the certificate as input, and terminates after Π^{FR}. Any party that does not have a certificate inputs $\perp$.
- **Certificate:** Upon receiving a signed (decide, u) message from any party, add the message to D_u. When D_u contains at least $t_{\mathsf{byz}} + 1$ messages from distinct parties, construct a *certificate* of $t_{\mathsf{byz}} + 1$ (decide, u) messages from distinct parties. Upon receiving a certificate, output u (if have not already output).

Fig. 6. Expected Constant Round Consensus Protocol Π^*

Proof. If all live parties have input v, then by $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-validity of Graded Consensus, each live party outputs $(v, 1)$ from every iteration of Graded Consensus. It follows that the first time Π^{coin} outputs v, all live parties output v.

Before proving consistency and termination, we prove the following claim of any $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution of Π^* that will facilitate the proofs of both properties:

Claim 8 *Let i be the iteration in which the first live party p outputs u. Then in every iteration $s > i$ while no honest party has terminated, every live party outputs $(u, 1)$ from subprotocol Π^{GC}.*

Proof. We will show that at the end of iteration i, every live party updates its internal value v to u. The claim follows from the fact that in the following iteration, every live party inputs u to Π^{GC}. By validity of Graded Consensus, this implies that every live party outputs $(u, 1)$ from Π^{GC} in that iteration and maintains the value of its internal variable v.

Inductively, as long as no honest party terminates, no live party ever changes its internal variable v after iteration i. This follows from the fact that all honest parties maintain the value of their internal variable v as long as all honest parties have input v and no live party has input $1-v$; although send-corrupt or receive-corrupt parties may terminate before an honest party, their inputs are treated as $\perp$ in the subsequent executions Π^{GC}, and the honest parties' internal variable of v is maintained by validity.

Now we show that at the end of iteration i, every live party updates its internal value v to u. We consider the two following cases for any live party $q \neq p$, based on q's output from Π^{GC} in iteration i. By consistency of Graded Consensus, because p output $(u, 1)$, q may output either $(u, 0)$ or $(u, 1)$ from Π^{GC}:

1. q output $(u, 0)$. Then q updates its internal variable v to the value ψ from the coin tossing in that iteration. By the fact that p outputs v in iteration i, $\psi_i = v$.
2. q output $(u, 1)$. Then q updates its internal variable v to the value u by the protocol specification.

Lemma 9 (Consistency). *$\Pi^*(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-consistent in synchronous networks for $n > t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.*

Proof. To prove consistency, we show that if some live party p outputs u, then no live party q ever outputs $1-u$. Recall that are two methods by which a party may produce output. We enumerate them as follows:

1. First, a party may output in iteration i when its internal variable v matches ψ_i.
2. Second, a party may output by receiving a certificate of signed decision messages.

Assume that in a $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-compliant execution of Π^*, some live party p outputs u and another live party q outputs $1-u$. First we claim that no two live parties may output conflicting values if both output by method 1.

Claim 10 *In any execution of Π^*, no two live parties r and s may output u and $1-u$, respectively, by method 1.*

Proof. Let there be an execution in which live party r outputs u by method 1 and live party s outputs $1-u$ by method 1. If r and s both output by method 1, they must have different values of their internal variable v at the moments they output. This is because when any party p outputs a value u by method 1, it must hold u in its internal variable v. However, because in every iteration, r and s both output the same value from Π^{coin}, and because each party outputs only when the output of Π^{coin} matches the value of its internal variable v, r and

s may not both output in the same iteration. Therefore, r and s must produce their outputs in different iterations.

Without loss of generality, let r produce output before s, and let s be the first live party to output $1-u$ by method 1 after r outputs u. (If s was not already the first live party to output $1-u$ by method 1 after r outputs u, then proceed to derive contradiction with respect to the first such party.) By Claim 8, every live party must have u in the value of its internal variables v until some honest party terminates. Therefore, because s must have $1-u$ in the value of its internal variable v when it outputs $1-u$, some honest party must terminate between the time that r produces output and s produces output. Let w be the first honest party to terminate, and let it terminate in iteration i. Since w is the first honest party to terminate, by Claim 8 w must terminate after outputting u. But then w sends its certificate via Π^{FR} in round i, and by Lemma 2, s received w's certificate or s becomes a zombie at the end of Π^{FR}. Then s does not output $1-v$, which is a contradiction.

It follows from Claim 10 that p and q may not both have output by method 1. Therefore, at least one party must have output by method 2.

Before concluding the proof, we claim that if any live party outputs a value u by method 2, then some live party must have output u by method 1. This follows directly from the fact that a valid certificate requires $t_{\mathsf{byz}}+1$ signed (decide, u) messages. In particular, at least one live party's signed (decide, u) message must be included in any certificate, and parties only produce signed (decide, u) messages when producing output by method 1.

We conclude the proof using this claim. Without loss of generality, let p output u by method 1 and q output $1-u$ by method 2. Then by the previous claim, there must be two honest parties that output conflicting values by method 1, which is a contradiction with Claim 10. We derive a similar contradiction with Claim 10 if two live parties output conflicting values by method 2.

Lemma 11 (Termination). *For all $\kappa \geq 1$ and $n > t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$, with probability at least $1-\frac{1}{2^\kappa}$, every live party running $\Pi^*(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ terminates in at most $2\kappa+1$ iterations.*

Proof. To analyze the probability that the all live parties terminate after m iterations, we separate the analysis into two steps. First, we define a *unanimous iteration* as an iteration at the end of which all live parties set their internal variables v to the same value. We will denote the first unanimous iteration of the protocol by i^* and analyze how many iterations the protocol requires until its first unanimous iteration. Second, we analyze how many iterations the protocol requires until all honest parties terminate after i^*.

To analyze how long the protocol requires until the first unanimous iteration, we first analyze how a unanimous iteration may occur. There are two possible ways that a unanimous iteration may occur:

1. If every live party outputs $(\cdot, 0)$ from Π^{GC} in iteration i, then at the end of iteration i, every live party updates its internal variable v to ψ_i.

2. If some live party outputs $(u, 1)$ from Π^{GC} in iteration i and $\psi_i = u$, then at the end of the iteration, all live parties update their internal variable v to u.

Therefore, i^* is the first iteration in which either all live parties output $g = 0$ from Π^{GC} or in which some live party outputs $(v, 1)$ from Π^{GC} and Π^{coin} outputs v. Conditioned on the fact that some live party outputs $(\cdot, 1)$ from Π^{GC} in each iteration, it follows from the fact that Π^{coin} is not biasable that each iteration i is unanimous with probability $\frac{1}{2}$. It follows that i^* occurs by iteration ℓ with probability at least $1 - \frac{1}{2^\ell}$. Let all live parties set their internal variable v to some $v^* \in \{0, 1\}$ at the end of iteration i^*. Next we claim that every live party outputs a bit no later than the next iteration $s > i^*$ in which $\psi_s = v^*$. The claim follows as a direct consequence of Claim 8, since all live parties are guaranteed to output $(v^*, 1)$ from Π^{GC} in every iteration after i^*, and in the case that $\psi_s = v^*$, every party that has not yet produced output must output v^*. Because Π^{coin} is not biasable, $\psi_s = v^*$ with probability $\frac{1}{2}$ in each iteration $s > i^*$. It follows that every live party outputs v^* by iteration $i^* + \ell$ with probability at least $1 - \frac{1}{2^\ell}$. It follows from linearity of expectations that all live parties output a bit from Π^* within 2κ iterations with probability $1 - \frac{1}{2^\kappa}$. When all honest parties output a bit, it follows that they all sign and send (decide, v) messages to each other. At latest, each party receives a certificate at the end of the iteration in which all honest parties output. Therefore, all live parties terminate Π^* within 2κ iterations with probability $1 - \frac{1}{2^\kappa}$.

5 Optimal Synchronous Consensus for Spotty Send Corruptions

In this section, we present slight modifications to the proofs in Section 4 that show Π^* achieves better corruption bounds when send-corruptions are spotty. We then prove that protocol Π^* is optimal in the number of corruptions it tolerates when send-corruptions are spotty.

5.1 Analysis for Spotty Send Corruptions

We now show that Π^* achieves better bounds for send-corruptions when such send failures are spotty.

Theorem 4. *When send-corruptions are spotty and a common coin primitive is available, Π^* is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.*

The proof follows in the remainder of this section, by updating the bounds and proofs of the underlying building block protocols.

Weak Broadcast With Unanimity. First we show that Π^{WB} is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure *with unanimity* when send failures are spotty, for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$. Because the construction does not change, we simply append the proof of unanimity.

Lemma 12. Π^{WB} *is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-unanimous for* $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ *when send corruptions are spotty.*

Proof. We show that if the dealer is live and broadcasts m, then every live party either outputs m or every live party outputs $\perp$. By the unforgeability of our idealized signature scheme, no signed message m' under the dealer's key can be forged. Therefore, whether every party outputs m or every live party outputs $\perp$ is determined completely by whether D's send succeeds in the first round of Π^{FR}. If D is honest, receive-corrupt, or if its send succeeds, then every honest and send-corrupt party receives m in the first round of Π^{FR}. If any receive-corrupt party does not receive m in the first round of Π^{FR}, then it must receive m in the second round, or become a zombie.

Weak Consensus We show that Π^{WC} is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ when send corruptions are spotty. We do not need to update the protocol, and we update only the proof of consistency.

Lemma 13. Π^{WC} *is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-consistent for* $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ *when send corruptions are spotty.*

Proof. The proof is identical to the one for Lemma 5, except that if send-failures are spotty, then we require only that $|A \cup B| > b$. This is because if any send-corrupt party sends a weak-broadcast that is received by any honest party, it must be received by all others by unanimity of Weak Broadcast, Lemma 12.

Graded Consensus We next show that Π^{GC} is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ when send corruptions are spotty. Once again, we do not need to update the protocol, and for this protocol we only update the proof of consistency.

Lemma 14. Π^{GC} *is* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-consistent for* $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$ *when send corruptions are spotty.*

Proof. The proof is identical to the one in Lemma 6, except that we need not consider send-corrupt parties whose weak broadcasts are delivered to p_i but not to p_j.

Because p_j output $(1-v, g)$ it must be the case that $n_{1-v} > n_v$ in p_j view. Then n_v must be at least $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - b^*$ in p_j's view. Because only byzantine parties may have weak broadcasted $1-v$, and because b^* corrupt parties did not weak broadcast $1-v$, n_{1-v} is at most $t_{\mathsf{byz}} - b^*$ in p_j's view. We reach a contradiction because $n - t_{\mathsf{byz}} - t_{\mathsf{snd}} - t_{\mathsf{rcv}} - b^* > t_{\mathsf{byz}} - b^*$ when $n > r + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$, and therefore $n_v > n_{1-v}$ in p_j's view and p_j did not output $(1-v, g)$.

And it follows that Π^{GC} is $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure because the proof of validity requires only $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.

Expected Constant Round Consensus The protocol and proof of Π^* do not need to be updated, as they inherit the bounds of the building block protocols, all of which are secure for $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.

5.2 Optimality with Respect to Spotty Send and Byzantine Corruptions

We now prove that Π^* is optimal in the number of corruptions it tolerates with respect to send corruptions and byzantine corruptions when send-corruptions are *spotty*. The proof does not consider a model with receive-corrupt parties. Recall that this model generalizes the crash failure model.

Theorem 5 (Optimal Send- and Byzantine Fault Tolerance). *There is no protocol for synchronous consensus in the mixed fault model which permits zombie processes that tolerates t_{snd} send corruptions and t_{byz} byzantine corruptions for $n \leq t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$*

Proof. The proof considers only send-corrupt and byzantine parties. It can be trivially extended to include a factor for t_{rcv} parties simply by adding an additional group of receive corrupt parties and forcing them to become zombies as the protocol begins.

Assume there is a consensus protocol Π resilient to t_{snd} and t_{byz} faults for $t_{\mathsf{snd}} + 2t_{\mathsf{byz}} \geq n$. We proceed by analyzing three separate executions of Π. As a tool towards analyzing executions, we first divide the set of parties $\mathcal{P}$ into three groups: A, B, and S. Group A has $n - t_{\mathsf{snd}} - t_{\mathsf{byz}}$ parties, Group B has $t_{\mathsf{byz}} \geq n - t_{\mathsf{snd}} - t_{\mathsf{byz}}$ parties, and Group S has t_{snd} parties. In the following three executions, the schedules of messages sent by and delivered to all parties are identical. Only the corruption status of parties across groups A, B, and S differ.

1. **Execution 1:** In this execution, all of the parties in Group A are honest and all have input 1. All the parties in Group B are byzantine and act as if they were honest parties with input 0. All of the parties in Group S are send-corrupt and do not send any messages, but have input 1. By validity, the honest parties (Group A) must output 1.
2. **Execution 2:** In this execution, all of the parties in Group A are byzantine and act as if they have input 1. All of Group B are honest and have input 0. All of the parties in S are send corrupt and do not send messages, but have input 0. By validity, all of the honest parties (Group B) must output 0.
3. **Execution 3:** In this execution, all parties in Groups A and B are honest. Group A all have input 1 and Group B all have input 0. All of the parties in S are send corrupt and do not send messages, but have input 1.

We now analyze the outputs of the parties in Execution 3. Notice that to the parties in Group A, this execution is identically distributed to Execution 1, so they must output 1. To the parties in Group B, this execution is identically distributed to Execution 2, so they must output 0. This violates consistency.

References

1. Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.
2. Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected o(1) rounds, expected o(n$\hat{\{2\}}$) communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.
3. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. Cryptology ePrint Archive, Report 2019/270, 2019. https://eprint.iacr.org/2019/270.
4. Bernd Altmann, Matthias Fitzi, and Ueli M. Maurer. Byzantine agreement secure against general adversaries in the dual failure model. In *DISC*, volume 1693 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 1999.
5. Michael Backes and Christian Cachin. Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. In *DSN*, pages 37–46. IEEE Computer Society, 2003.
6. T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Round complexity of byzantine agreement, revisited. *IACR Cryptology ePrint Archive*, 2019:886, 2019.
7. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983. URL: https://doi.org/10.1137/0212045, http://dx.doi.org/10.1137/0212045 doi:10.1137/0212045.
8. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.
9. Matthias Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 3 2003. Reprint as vol. 4 of ETH Series in Information Security and Cryptography, ISBN 3-89649-853-3, Hartung-Gorre Verlag, Konstanz, 2003.
10. 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.
11. Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In *WDAG*, volume 647 of *Lecture Notes in Computer Science*, pages 153–165. Springer, 1992.
12. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM, 2017.
13. Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. Cryptology ePrint Archive, Report 2019/179, 2019. https://eprint.iacr.org/2019/179.
14. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462. Springer, 2006.
15. Klaus Kursawe. Distributed protocols on general hybrid adversary structures. 2004.
16. Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In *PODC*, pages 303–312. ACM, 2014.

17. Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. *CoRR*, abs/1904.10067, 2019.
18. S. Micali. Byzantine agreement , made trivial. 2017.
19. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society, 1999.
20. Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017.
21. Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In *TCC (1)*, volume 12550 of *Lecture Notes in Computer Science*, pages 412–456. Springer, 2020.
22. Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round byzantine broadcast under dishonest majority. In *TCC (1)*, volume 12550 of *Lecture Notes in Computer Science*, pages 381–411. Springer, 2020.
23. Vassilis Zikas, Sarah Hauser, and Ueli M. Maurer. Realistic failures in secure multiparty computation. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 274–293. Springer, 2009.

A Dolev and Strong's Results

Below we first review the classic result of Dolev and Strong for authenticated broadcast, and then review the lower-bound on the round complexity of a deterministic broadcast protocol.

A.1 Dolev and Strong's Authenticated Broadcast Protocol

We recall Dolev and Strong's authenticated broadcast protocol in Figure 7.

A.2 Dolev and Strong's Impossibility

In this section we provide an exposition of Dolev and Strong's lower-bound on the round complexity of a deterministic broadcast protocol. *We highlight the fact that the proof requires only send-corruptions and not fully byzantine corruptions, and therefore the impossibility result holds for only send-corruptions.*

Recall that the result by Dolev and Strong proves there is no deterministic broadcast protocol tolerating b byzantine parties that terminates in at most b rounds. The proof proceeds by assuming such a protocol and considering a "good" execution in which all messages of the protocol are delivered in every round. It then proceeds to define a series of "neighboring" executions such that every two neighboring executions are *identical* except that in one of the two executions, there is one exactly one round in which a message sent by one party is dropped. The transition between neighboring executions maintains two invariants:

1. In every pair of neighboring executions A and B, there is some honest party q such that the view of q is identical in A and B. (This means that q receives exactly the same messages in the two neighboring executions.)

Protocol 7 Dolev Strong Broadcast Protocol Π^{DS}

Shared Setup: Public Key Infrastructure for a signature scheme.
Inputs: The dealer $D \in \mathcal{P}$ has an input $m \in \{0,1\}^*$.
Outputs: Each party $p \in \mathcal{P}$ outputs a value $m' \in \{0,1\}^* \cup \{\perp\}$.
Local Variable: Each party $p \in \mathcal{P}$ maintains a local variable S, which is a set initialized to $\{\}$.
Protocol: The protocol begins at time 1 and proceeds in rounds. Each round party p proceeds as follows:

1. **Round 1: Dealer's Messages** The Dealer D signs its input $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(m)$ and sends (m, σ) to all parties.
2. **Sig Chains** For every round i from 2 to $t_{\mathsf{byz}} + 1$: For every valid $(i-1)$-sig-chain c that p received at the end of round $i-1$ in which none of the signatures were constructed by p, p computes $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(c)$ and sends (σ, c) to all parties. For every valid i-sig-chain c received in round i, let m' be the message contained by c. Update $S = S \cup \{m'\}$.
3. **Output** If $|S| = 1$, then p outputs the element $m' \in S$. If $|S| \neq 1$, then p outputs $\perp$.

Fig. 7. Dolev-Strong Broadcast Π^{DS}

2. In no execution are more than b parties corrupted.

Because of invariant 1, we require that in every execution in the sequence, all honest parties output the same value. This follows from the fact for every pair of neighboring executions, the party q whose view is identical in the two executions must output the same value in both, and all other parties must output q's value in both executions by consistency. Contradiction follows by arriving at an execution in which the dealer sends no messages; therefore, the output of every party must be independent of the dealer's value.

In order to define a series of executions that satisfy the above properties, the proof defines a "communication graph" that describes all messages sent in an execution. A communication graph is a directed acyclic graph (DAG) which is divided into "levels" such that in each level, every party is represented by a distinct vertex. If in some execution, party A sends a message to party B in round r that B receives round $r+1$, then there is an edge from A's vertex in level r to B's vertex in level $r+1$. (We assume synchronous communication in which all messages sent in round r are always delivered in round $r+1$.)

The proof begins with the full execution graph and defines a recursive procedure by which all messages sent by a party in round r and greater can be removed from the graph, while maintaining the properties required above. Let R be the final round in an execution, and consider two graphs such that the only difference is that a message from party A to party B sent in round $R-1$ and received in R is removed from one of the two graphs. Clearly, all parties except for B have views that are identical between the two executions. Next, consider two graphs such that the only difference between the two is that a message from party A to party B sent in round $r < R-1$ and received in $r+1$ is removed from one of the two graphs, but for which B sends no messages in any round $r' \geq r+1$. Again, all parties except for B have the same view of the execution in the two graphs. Similarly, the reverse operations also preserve invariant 1. Namely, an edge can be "restored" from A to B in either of the two above scenarios.

The proof shows how to remove all edges from the sender while maintaining the above invariants. In order to remove all messages from a party P starting at round r: For every party Q that receive a message that P sends in round r and are delivered in round $r+1$, remove all future edges sent by Q starting in round $r+1$. Then remove the edge from P to Q sent in round r and delivered in round $r+1$. Then restore all edges sent by Q starting in round $r+1$. The proof guarantees that in any execution graph, at most one party is corrupted *per round of the protocol in which messages are dropped*, which maintains that if the protocol requires R rounds, then at most R parties need to be corrupted. The contradiction follows for any protocol requiring fewer than $t_{\mathsf{byz}}+1$ rounds – or in our case, $t_{\mathsf{snd}}+1$ rounds.

For a full treatment, we refer the reader to the very thorough explanation by [6], complete with diagrams.

B Generic Consensus from Broadcast

Here we provide the folklore construction (also discussed by Fitzi [9]) of achieving consensus using a broadcast primitive. Recall that we do not know how to prove an optimal corruption threshold for consensus in our regime for the general form of send corruption; however we provide an upperbound of $n > t_{\mathsf{rcv}} + 2t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$. By this construction, the problem is reduced to finding a broadcast protocol tolerating $n > t_{\mathsf{rcv}} + t_{\mathsf{snd}} + 2t_{\mathsf{byz}}$.

The construction is as follows: given a broadcast protocol Π^{B}, all parties simultaneously broadcast their inputs. Each party counts the number of broadcasts for which it outputs 0, 1, and $\perp$, and it outputs whichever of 0 and 1 appears more. In our model, we require that $n > t_{\mathsf{snd}} + t_{\mathsf{rcv}} + 2t_{\mathsf{byz}}$ to enforce that even if all send-corrupt parties' broadcasts and receive-corrupt parties' broadcasts output $\perp$ (because they fail in their own respective ways), a majority of the remaining parties are honest.

Note that it is still an open problem (with closest attempt coming from Wan et al [22]) to obtain a constant round byzantine broadcast protocol for dishonest majority, with only crash and byzantine faults. Our model is still stronger than theirs, as we consider a strongly adaptive adversary (theirs is weakly adaptive) and we permit send-corruptions.

Lemma 15. *If there exists a* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ *secure broadcast protocol for* $n > t_{\mathsf{snd}} + t_{\mathsf{rcv}} + 2t_{\mathsf{byz}}$ *then there is a* $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$*-secure consensus protocol for* $n > t_{\mathsf{snd}} + t_{\mathsf{rcv}} + 2t_{\mathsf{byz}}$.

Proof. Given a $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$ secure broadcast primitive or protocol Π^{B} for $n > t_{\mathsf{snd}} + t_{\mathsf{rcv}} + 2t_{\mathsf{byz}}$, we construct a corresponding consensus protocol Π^{C} as outlined in Figure 8.

Protocol 8 Consensus from Broadcast $\mathit{\Pi}^{\mathsf{C}}$

Inputs: Each party $p \in \mathcal{P}$ has an input $b \in \{0,1\}$.
*Outputs:*Each party $p \in \mathcal{P}$ outputs $v \in \{0,1\}$.
Protocol: Each party p proceeds as follows:

1. **Broadcast Input:** Each party broadcasts its input to all other parties using $\mathit{\Pi}^{\mathsf{B}}$.
2. **Count Received Bits:** For $u \in \{\perp, 0, 1\}$, let n_u be the number of broadcasts in the previous step for which p output u.
3. **Output:** Output $v \in \{0,1\}$ for which $n_v > n_{1-v}$. If $n_0 = n_1$ then output 1.

Fig. 8. Generic Consensus from Broadcast Construction

We prove that $\mathit{\Pi}^{\mathsf{C}}$ is a $(t_{\mathsf{snd}}, t_{\mathsf{rcv}}, t_{\mathsf{byz}})$-secure protocol. Termination follows from the fact that $\mathit{\Pi}^{\mathsf{B}}$ terminates. Consistency follows from consistency of $\mathit{\Pi}^{\mathsf{B}}$, which requires that if any live party output $m \in \{\perp, 0, 1\}$ from any instance of $\mathit{\Pi}^{\mathsf{B}}$, then all live parties output the same thing. It follows that every live party has the same values of n_0, n_1, and $n_\perp$. Validity follows from the fact that even if all send-corrupt parties' broadcasts output $\perp$ and all receive-corrupt parties become zombies before completing their broadcast protocols (implicitly assuming that a protocol at worst outputs $\perp$ if a receive-corrupt sender does not complete the protocol), then there are still more honest parties who successfully broadcast than byzantine parties. This follows from $n - t_{\mathsf{snd}} - t_{\mathsf{rcv}} > 2t_{\mathsf{byz}}$.