Optimal Consensus in the Presence of Overlapping Faults and Total Omission

Julian Loss[1], Kecheng Shi[1,3], and Gilad Stern[2]

[1] CISPA Helmholtz Center for Information Security
[2] Tel Aviv University
[3] Saarland University

Abstract. Understanding the fault tolerance of Byzantine Agreement protocols is an important question in distributed computing. While the setting of Byzantine faults has been thoroughly explored in the literature, the (arguably more realistic) omission fault setting is far less studied. In this paper, we revisit the recent work of Loss and Stern who gave the first protocol in the mixed fault model tolerating t Byzantine faults, s send faults, and r receive faults, when $2t + r + s < n$ and omission faults do not overlap. We observe that their protocol makes no guarantees when omission faults can overlap, i.e., when parties can simultaneously have send and receive faults. We give the first protocol that overcomes this limitation and tolerates the same number of potentially overlapping faults. We then study, for the first time, the *total omission setting* where all parties can become omission faulty. This setting is motivated by real-world scenarios where every party may experience connectivity issues from time to time, yet agreement should still hold for the parties who manage to output values. We show the first agreement protocol in this setting with parameters $s < n$ and $s+r = n$. On the other hand, we prove that there is no consensus protocol for the total omission setting which tolerates even a single overlapping omission fault, i.e., where $s+r = n+1$ and $s > 2$, or a broadcast protocol for $s + r = n$ and $s > 1$ even without overlapping faults.

1 Introduction

Consensus is a fundamental problem in distributed computing that asks a set of n parties to agree on a common output. Their task is complicated by a subset of faulty parties who interfere with the honest parties' execution of the consensus protocol by sending incorrect messages or simply crashing. Consensus serves as a key building block in various applications such as verifiable secret sharing, MPC, state-machine replication, and more. The overwhelming majority of the literature considers protocols for the fully malicious (a.k.a. Byzantine) setting, where nodes can exhibit arbitrary behaviour. These protocols offer a high degree of security, which may be justified in high-stakes applications such as blockchain systems or reliable databases. On the other hand, the fault tolerance of such protocols is also inherently limited. Indeed, it is well-known that in the plain

model without setup, consensus can be solved if and only if the number t of (maliciously) faulty parties satisfies $t < n/3$. While this can be ameliorated by relying on cryptographic setup such as digital signatures, it can be shown that even given these additional tools, consensus can be achieved if and only if $t < n/2$. A natural question is therefore whether a higher fault tolerance can be achieved if the faulty parties' deviant behaviour is restricted in some manner. One of the most important types of faults one can consider in this context are *omission faults*. Omission faults are parties who run the protocol code honestly, but for which some of the protocol messages may get lost during sending (send omission) or during receiving (receive omission). On one hand, omission faults model a realistic network setting where intermittent failures may occur. On the other hand, consensus can be solved for any number of omission faults $o < n$ (assuming a synchronous network).Moreover, under some conditions, it is even possible to guarantee *uniformity*, meaning that any omission faulty party that does output, does so in agreement with honest parties [21,23,29].In this work, we significantly advance our understanding of the omission fault setting by showing the following results:

- We begin by revisiting the recent work of Loss and Stern [19] who consider a model with mixed types of faults. More precisely, their work gives the first protocol tolerating (simultaneously) t Byzantine faults, r receive faults (for whom the adversary can drop arbitrary incoming messages), and s send faults (for whom the adversary can drop arbitrary outgoing messages), s.t. $2t+r+s < n$. We observe that their protocol does not work when faults can be *overlapping*, i.e., when a party can become both receive and send faulty at the same time. To overcome this limitation, we give a novel consensus protocol which tolerates the same number of faults, but allows to count overlapping faults twice (i.e., as both send- and receive faulty) in the above formula.
- In the second part of our paper, we study consensus in a setting where *every party* can be *either* send faulty *or* receive faulty, i.e., $r + s \leq n$. This setting is motivated by the fact that, from time to time, every party may experience connectivity issues and lose messages. In this case, we would still like to guarantee that for parties who successfully complete the protocol, their output satisfies the usual requirements of a consensus protocol. On the positive side, we give the first protocol in this setting achieving consensus. An interesting feature of our protocol is that it runs in only $O(s)$ many rounds and achieves perfect security. On the negative side, we show that there is no broadcast protocol in this setting when $s > 1$. Additionally, whenever $s > 2$ there is no protocol tolerating even a *single overlapping omission fault*, i.e., where $s + r = n + 1$.

1.1 Our Techniques

We now give a technical overview of our results.

Zombies and Ghosts. Our work builds on the ideas of Zikas, Hauser and Maurer [29] and of Loss and Stern [19]. Zikas *et al.* constructed an information theoretic consensus protocol resilient to t Byzantine corruptions, s send corruptions and r receive corruptions when $n > 3t + r + s$. At the base of their constructions, they utilized the idea of *self-detection*. When a party s sends a message m directly to some party p, p might not receive m for two possible reasons: p might be receive faulty or s might be Byzantine or send faulty (or both). In order to help party p self-detect as receive faulty in the above scenario, s instead relays m to p through every other party in the network. If p is *not* receive faulty, it receives messages from many other parties containing either m or a notification that s did not send a message to p. On the other hand, if p *is* receive faulty, it may receive only very few messages. In the latter scenario, p can detect that it is receive faulty. It then becomes a *zombie*, and stops participating in the protocol in order to make sure it does not harm the rest of network by propagating messages that are inconsistent with honest parties' protocol states.

The work of Loss and Stern generalizes these ideas to the cryptographic setting, assuming $n > 2t + r + s$. The novelty of their construction is a means for send faulty parties to detect themselves, upon which they become *ghosts* and stop participating in the protocol. As before, parties in their protocol send messages to each other through the whole network in order to allow for receive faulty parties to detect themselves as zombies. However, in their protocol, parties also reply to the sender, allowing it to learn whether it succeeded in sending its message or whether it should abort by becoming a ghost. If a sender s receives very few of these abort messages, it knows that its message was delivered to at least one non-faulty party p, who will then be able to propagate it to the rest of the network. These ideas are used in their most basic primitive, a weak multicast protocol (WMC). In WMC, the sender s either successfully delivers its message m to at least one honest party or detects itself as send-faulty.

Additional Challenges with Full-Omission Faults. We notice that the WMC protocol of Loss and Stern (and by extension, all protocols building on top of it) does not cover the most general type of omission fault which cause a party to become simultaneously send- *and* receive faulty, i.e., full-omission faulty. Specifically, when the sender s in their WMC protocol is full-omission faulty, it may not receive the abort messages that honest parties send back to s when they did not receive its message. This prevents s from detecting itself as faulty and becoming a ghost. Dealing with this issue turns out to be very subtle. We devise a novel WMC protocol that leverages additional communication among the receivers, so as to help the sender s detect itself as faulty.

In more detail, parties that do not receive messages from s inform each other of this fact in the form of "abort" messages. Now any party p that is *exclusively* send faulty must have received all abort messages from all honest parties. In the last step of our protocol parties then forward a list of *all* of the abort messages they received back to s. This allows us to modify the conditions under which s turns itself a ghost compared to the protocol of Loss and Stern. Namely, our protocol counts the *total number* of abort messages received directly *or* indirectly

through the above mechanism. In this manner, we force the adversary into the following dilemma. Either, it drops a large number of messages to s in the last step of the protocol. This causes the sender to infer that it must be receive faulty and turn itself a zombie. Alternatively, the adversary delivers the collection of abort messages assembled by at least one party that received all abort messages from all honest parties back to s. This, on the other hand, will make s detect itself as send-faulty and turn itself a ghost. In summary, our WMC allows even a full-omission faulty sender to either send its message to at least one honest party, or turn detect itself as send or receive faulty by the end of the protocol. Our new protocol can be used as a drop in replacement for calls to WMC in the protocols of Loss and Stern. In this manner, we can easily carry over the their consensus protocol to the full-omission setting.

Total Omission Setting. In the second technical part of our work, we initiate the study of a setting in which *all parties* could be omission faulty. This is a very realistic setting that is based on the observation that in practice, it may be very difficult to guarantee permanent connectivity of any of the individual protocol participants. In this case, we would still like a protocol that ensures uniformity, i.e., that parties who output are in consensus with each other. We design a uniform consensus protocol resilient to any s send faults and r receive faults s.t. $s + r = n$ and $s < n$. For completeness, we also show that no such protocol exists if $s = n$. This slighlty strengthens a previous result of Hadzilacos [15]who showed that no broadcast protocol exists when $s = n$. We remark that one could imagine a setting where $s + r > n$ without overlapping faults, in which the adversary has the flexibility to choose the *actual* number of corrupted parties of each kind. We do not address this setting in our paper, but believe that this is a very interesting direction for future work.

An interesting question that arises as a consequence of studying the total omission setting is how it relates to the work of Eldefrawy, Loss, and Turner *et al.* [11]. Their work includes a lower bound showing that $n > 2t + r + s$ is a necessary condition for consensus. We observe that this lower bound is not tight for (at least) the case where $t = 0$. Namely, their bound is stated in a model which allows the adversary to forcibly "zombify" receive faulty parties at the onset of the protocol, upon which they cease any subsequent participation.

This severely limits the generality of protocols covered by this lower bound, since some protocols might have parties send messages even after detecting their own faults. In particular, our protocol for the total omission setting heavily relies on parties who have detected themselves as zombies to continue assisting in the rest of the protocol.

Impossibility Results. We complement our study of the total omission setting by proving two impossibility results. This helps us fill in some of the gaps in our current understanding of the task of consensus.

- Although we construct a uniform consensus protocol with in the total omissions setting, surprisingly, we show that it is impossible to construct a broadcast protocol in this setting whenever $s > 1$. We briefly illustrate why this

is the case. Our consensus protocol relies heavily on the capability of the receive faulty parties to distribute their initial inputs to the non-receive faulty protocol parties. This approach fails completely for a broadcast protocol, since only the sender has input.
– Our second impossibility result shows that it is generally impossible to construct a uniform consensus protocol if $s + r > n$ and $s > 2$. This shows our protocol has the optimal corruption threshold.

1.2 Related Work

The study of consensus protocols, and the related study of broadcast protocols, has a long history [9,12,16,20]. Early results dealt with constructing protocols for systems with a single type of failure. For example, Hadzilacos [15] showed that broadcast is possible in a system of s send faulty parties if and only if $n > s$. Following that, Perry and Toueg [22] shows that broadcast is possible in systems with o general omission faults if $n > o$, while not requiring uniformity, i.e., where omission faulty parties output the correct values. When considering protocols that do require uniformity of outputs, works by Raynal and Parvédy [21,23] showed that consensus is possible if and only if $n > 2o$. Dolev and Strong [9] showed that a similar result holds for Byzantine authenticated broadcast (i.e. with a PKI setup) can be solved in the presence of t Byzantine faults if $n > 2t$. On the other hand, Lamport Shostak and Pease [16] showed that Byzantine consensus can be solved if $n > 3t$ in the unauthenticated setting and $n > 2t$ in the authenticated setting.

Later work also dealt with constructing such protocols in networks of mixed faults. Garay and Perry [13] constructed a consensus protocol resilient to t Byzantine parties and c crash faulty parties that can crash in any point in time and stop participating in the protocol, assuming that $n > 3t + c$. Siu, Chin and Yang [26] strengthened this result and constructed a consensus protocol resilient to t Byzantine parties and k parties with arbitrary non-malicious faults if $n > 3t + k$. Additional more specialized models dealt with t malicious parties, k non-malicious parties and f parties that can act maliciously, but cannot send different messages to different parties. Protocols such as those of Thambidurai and Park [27] and of Lincoln and Rushby [18] are correct as long as $n > 3t + 2f + k$. The more recent work of Hauser, Maurer and Zikas [29] showed that consensus, broadcast and MPC constructions are possible in networks with t Byzantine parties, s send faulty parties and r receive faulty parties, assuming that $n > 3t + r + s$. Recently, Konstantinos and Zikas [3] provided a tight characterization of feasibility for information theoretically secure consensus and MPC with in networks with Byzantine and full-omission faults in the general adversary structure.

Using many of these ideas, the recent work of Eldefrawy, Loss and Terner [11] Abraham, Dolev, Kagan and Stern [1] and Loss and Stern [19] construct mixed-fault protocols in the authenticated setting. Abraham *et al.* [1] construct an authenticated consensus protocol if $n > 2t + c$ Eldefrawy *et al.* construct such a protocol if $n > 2t + 2s + r$, or if $n > 2t + r + s$ and send faulty parties

can either successfully send all messages in a given round, or no messages. The followup work by Loss *et al.* removes this requirement on send faulty parties and achieves consensus if $n > 2t + r + s$. One approach to achieving high resilience is to limit the adversary's actions, and only allow it to act Byzantine in certain parts of the code. This means that in some sense, faulty parties have mixed faults: Byzantine in some code sections, but only non-malicious in others. For example, the works of [5,7,8,17,28] use trusted execution environments (or similar abstractions) to enforce such behaviour from Byzantine parties. Alternatively, some protocols only allow specific parties to be Byzantine [25] or only allow one type of corruption at a time [2].

Similar results have been shown in partially synchronous systems that start as asynchronous networks, and eventually stabilize and become synchronous. For example, the Scrooge [24] protocol is secure if $n > 4t + 2c$. On the other hand, theUpright [6] and SBFT [14] protocols are secure as long if $n > 3t + 2k$ and $n > 3t + 2c$ respectively. Note that like results in synchronous networks, the latter two protocols also "naturally" combine the resilience of protocols for a single faulty type. That is, we know that $n > 3t$ or $n > 2k$ is required for partially synchronous protocols when allowing only Byzantine or non-malicious faults respectively [10].

2 Models, Definitions and Notations

2.1 Network Model

Throughout this work we deal with a fully-connected network of n parties. This means that each pair of parties has a direct channel between them, allowing parties to send messages to each other. The channels are authenticated, meaning that when parties receive a message they know the identity of the sender. In addition, when dealing with Byzantine corruptions, we assume a PKI setup, which allows parties to sign messages and verify each other's messages. We follow the standard approach of modelling the signature scheme as perfectly unforgeable. When replacing these signatures with existentially unforgeable signatures, the guarantees of the protocols hold when considering computationally bounded adversaries. We use the notation $\langle m \rangle_i$ to mean the message m, accompanied by i's signature on the message. The network is assumed to be synchronous. This means that the parties have access to synchronized clocks and run protocol proceeds in well-defined rounds. Parties can send messages in the beginning of a round, and all message that aren't dropped by the adversary (see below) are delivered at the end of the round. Messages can be delivered within each round in whichever order the adversary chooses. Parties can then choose which messages to send in a certain round based on the messages received from the previous round. Our protocols are described in steps of fixed duration (i.e., number of rounds) that are executed in lock-step one after another.

2.2 Adversary Model

Our work aims to design protocols with mixed-fault networks. There are four types of corruption in our work, and we will mention which types are included at the beginning of each section.

- **Send-Omission Faults.** Send faulty parties follow the protocol description. For any message sent from a send faulty party, the adversary can choose to drop that message. We assume there are at most s send omission faults.
- **Receive-Omission Faults.** Receive faulty parties follow the protocol description. For any message sent to a receive faulty party, the adversary can choose to drop that message. We assume that there are at most r send omission faults.
- **Full-Omission Faults.** Full omission faulty parties follow the protocol description and are both send faulty and receive faulty. This means that the adversary can drop any message sent by or to full omission faulty parties.
- **Byzantine/Malicious Faults.** Byzantine/Malicious parties can deviate arbitrarily from the protocol. We assume there are at most t Byzantine parties.

In this work we consider a strongly adaptive adversary that can corrupt parties at any time throughout the protocol, and can drop messages to receive faulty parties, drop messages to send faulty parties and replace messages sent by Byzantine parties in the same round they are corrupted. When corrupting a party, the adversary learns its entire state. On the other hand, the lower bounds of this paper work for a static adversary that chooses which parties to corrupt in the beginning of the protocol.

2.3 Definitions

In this section we define the tasks to be solved in the paper. We follow the ideas and notation presented by Loss *et al.* [19] in the definitions. In their protocols, parties always receive two flags z, g as inputs, in addition to any inputs they receive in the protocol. In all of the protocols, parties may not receive $\perp$ as an input, and indeed never receive that input in our protocols. One could allow such inputs by having distinct $\perp$ values for each protocol. The flags z and g indicate whether the party is already a zombie or a ghost respectively in the beginning of the protocol, and parties store these values in the beginning of each protocol. This allows parties to exclude themselves from protocols if they already know they are zombies or ghosts. A party that is not a zombie or a ghost is said to be *alive*. Parties then output a value x along with two flags z, g. The outputs z and g indicate whether the party is a zombie or a ghost respectively by the end of the protocol. When dealing with mixed faults with the presence of Byzantine faults we keep this notation in order to be consistent with previous work. However, in protocols for the total omission setting, parties might be required to participate even if they detect their own faults. In addition, in this setting our protocols do not allow for parties to detect their send faults and become ghosts. This means that inputting the flags Z, G and outputting the flag g are not meaningful in

this setting, and for simplicity we remove these flags. Standard definitions of the tasks are also provided in Appendix A. The standard definitions are the same as the undead versions, except parties do not become zombies or ghosts. The lower bounds in this paper hold for the standard definitions, which also imply lower bounds for the undead versions.

Undead Weak Multicast An undead weak multicast protocol, defined in the work of Loss et al. [19], allows parties to attempt to send their values to all other parties in the presence of Byzantine, send and receive faults. This is done in such a way that receive faulty parties can detect that they did not receive the message, and send faulty parties can detect that no honest parties received the message. In their work, Loss *et al.*, construct an undead uniform consensus protocol in such a network by first constructing an undead multicast protocol. They then construct a stack of protocols culminating in a consensus protocol. In this work we adapt the undead weak multicast protocol and only very slightly adapt the undead graded multicast protocol. Therefore, an undead weak multicast definition is provided below, and the definition and adaptation of the undead graded multicast protocol is provided in Appendix C.

Definition 1. *Let Π be a protocol executed by parties $1, \ldots, n$, with a designated sender i^* starting with an input $m \neq \perp$. In addition, every party i has two values $z_i, g_i \in \{\mathsf{True}, \mathsf{False}\}$ as input. Every party outputs a triplet (x, z, g) such that x is either a possible message or $\perp$, and z, g are boolean values.*

- ***Validity.*** *Π is* (t, s, r)-valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: If i^* is non-faulty or receive faulty and is alive in the beginning of the protocol, every non-Byzantine party j either outputs (x, z, g) such that $x = m$, or such that $z = \mathsf{True}$. In addition, if i^* is send faulty, no non-Byzantine party outputs (x, z, g) such that $x \notin \{m, \perp\}$.*
- ***Detection.*** *Π is* (t, s, r)-detecting *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: If i^* is send faulty and it is alive at the end of the protocol, at least one non-faulty party output (x, z, g) such that $x = m$.*
- ***Termination.*** *Π is* (t, s, r)-terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: All non-Byzantine parties complete the protocol and output a value.*
- ***No Living Undead.*** *Π is* (t, s, r)-no living undead *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: If a non-Byzantine party j outputs (x, z, g) such that $z = \mathsf{True}$ (resp. $g = \mathsf{True}$), then it is faulty (resp. send faulty).*

If Π is (t, s, r)-valid, (t, s, r)-detecting, (t, s, r)-terminating, *and* (t, s, r)-no living undead *we say that it is a* (t, s, r)-secure *undead weak multicast protocol.*

Very Weak Multicast In order to construct a consensus protocol in the total omission setting, we start by constructing a rudimentary multicast primitive which we call a very weak multicast protocol. This protocol has a designated sender that attempts to multicast its message. Informally, the multicast must succeed if the sender is receive faulty, or if it is send faulty and there are fewer than r receive faulty parties. Parties that do not receive the message due to their own receive faults must become zombies. Formally, the protocol is defined as follows:

Definition 2. *Let Π be a protocol executed by parties $1, \ldots, n$, where i^* is the designated sender starting with an input $m \neq \perp$. Every party $j \in [n]$ outputs (x, Z_j) at the end of the protocol, where x is either a possible message or $\perp$, Z_j is a boolean value.*

- ***Validity.*** *Π is* (s, r)-valid *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: Every party outputs (x, Z) such that $x \in \{m, \perp\}$. In addition, if i^* is honest and there are at most $r - 1$ receive faulty parties or if i^* is a receive faulty party, then every party j either outputs (m, False), or outputs $(\perp, \mathsf{True})$ by the end of the protocol.*
- ***Termination.*** *Π is* (s, r)-terminating *if the following holds whenever at most s parties are send faulty, r parties are receive faulty: All parties terminate and output a value at the end of the protocol.*
- ***No Living Undead.*** *Π is* (s, r)-no living undead *if the following holds whenever at most s parties are send faulty, r parties are receive faulty: If some party j outputs (x, True), then j is receive faulty.*

If Π is (s, r)-valid, (s, r)-terminating *and* (s, r)-no living undead *we say that it is a* (s, r)-secure *very weak multicast protocol.*

Undead Uniform Consensus In a uniform consensus protocol, all parties have an input and they are required to output the same value, or possibly output $\perp$ if they are receive faulty. Importantly, in "normal" consensus protocols, the output of receive faulty parties is not required to be consistent with other parties' outputs. In a uniform consensus protocol, even faulty parties must output the same value as all other values, unless they can detect their own faults and output $\perp$. Similarly to above, parties also output a boolean flag z, indicating whether they detected their own receive faults and became zombies. For a definition of a uniform consensus protocol without the notion of undead parties, see Appendix A.

Definition 3. *Let Π be a protocol executed by parties $1, \ldots, n$, where each party $j \in [n]$ starts with input $m_j \neq \perp$. Every party j outputs (x_j, Z_j) at the end of the protocol.*

- ***Validity.*** *Π is* (s, r)-valid *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: If each party j starts with the same value $m_j = m$, all parties output (m, False) or $(\perp, \mathsf{True})$ at the end of the protocol.*

- ***Consistency.*** *Π is* (s,r)-consistent *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: All non-faulty parties and send faulty parties output $x_j = m$ for the same value m at the end of the protocol. In addition, every receive faulty party either outputs m or $\perp$.*
- ***Termination.*** *Π is* (s,r)-terminating *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: Each party j terminates and outputs (x_j, Z_j) at the end of the protocol.*
- ***No Living Undead.*** *Π is* (s,r)-no living undead *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: If $Z_j = \mathsf{True}$ at the end of the protocol, j must be a receive faulty.*

If Π is (s,r)-valid, (s,r)-consistent, (s,r)-terminating *and* (s,r)-no living undead *we say that it is an* (s,r)-secure *undead uniform consensus protocol.*

Broadcast The task of broadcast is highly related to the task of consensus, and has been shown to be equivalent in some network settings [4,15]. In this task, one designated sender has an input m and all parties output the same value x in the end of the protocol. If the sender does not exhibit faults that prevent it from sending its input (i.e. send or Byzantine faults), all parties should output $x = m$ as well. For a formal definition see Appendix A.

3 Byzantine Agreement with Overlapping Omission Faults

This section deals with the construction of a Byzantine Agreement protocol in the presence of t Byzantine faults, r receive faults and s send faults when $n > 2t + r + s$ and the faults can overlap. As shown in [11], this is the optimal resilience for such protocols. The work of [19] constructs a protocol with such resilience when disallowing overlapping omission faults. In this section we adapt their protocol to the overlapping fault setting. Their protocol is constructed from a stack of four protocols: undead weak multicast, undead graded multicast, undead weak consensus and finally undead consensus.

The most basic protocol in the stack, the undead weak multicast protocol, heavily relies on send faulty parties receiving an indication that others did not hear their message. This allows them to become ghosts and stop participating in the protocol. This mechanism does not work when these parties can also exhibit receive faults. In order to remedy this, parties also send these indications to each other, which are then forwarded back to the faulty sender. If a sender receives enough of these messages it will be able to detect its own send faults and become a ghost. On the other hand, if a party receives too few of these forwarded messages (or messages indicating that no error occurred), they will detect their receive faults and become zombies instead. The protocol is presented in Fig. 1.

The rest of the stack is nearly identical to the original construction. Very slight adaptations to the undead graded multicast definition and protocol are

presented in Appendix C. In the original construction, if a party is receive faulty, it must succeed in sending its message. This property is actually not needed for the rest of the constructions and the proofs, as they only rely on fully non-faulty parties succeeding in sending their messages. In that sense, that property is "too strong" and is used in the original work only because it is possible to achieve. Since the rest of the constructions and proofs remain exactly the same in the mixed-fault setting, Appendix C only contains adaptations to the undead graded multicast protocol.

Protocol Π_{WMC}

Input and Initialization: Each party $j \in [n]$ has inputs Z_j, G_j, the designated sender i^* inputs m additionally. Every party j initializes $m_j = \perp$.

Parties execute the following steps:

1. The sender $i^* \in [n]$ sends $\langle m \rangle_{i^*}$ to all other parties if $Z_{i^*} = \mathsf{False}$ and $G_{i^*} = \mathsf{False}$.
2. Every party $j \neq i^*$ sets $m_j = m'$, upon receiving a message m' with a valid signature from i^* in Step 1. If no message was received from i^*, j sets $m_j = \perp$. If $m_j \neq \perp$, j sends $\langle m_j \rangle_{i^*}$ to all other parties. Otherwise, it sends $\langle \perp \rangle_j$.
3. Every party j that has $m_j = \perp$ and received a forwarded message $\langle m' \rangle_{i^*}$ with a verifying signature from i^* sets $m_j = m'$ (if more than one such message is received, choose one arbitrarily). Every party j who did not receive any message with i^*'s signature (either directly in Step 1 or indirectly in Step 2) does as follows.
   - If j received $\langle \perp \rangle_p$ from at least $n - t - s$ different parties p in Step 2, j sends $\langle \mathsf{Abort} \rangle_j$ to all other parties.
   - Otherwise, if party j received $\langle \perp \rangle_p$ from less than $n - t - s$ many parties p, it sends "$Z_j = \mathsf{True}$" to the sender and sets $Z_j = \mathsf{True}$.
4. Each party j gathers all $\langle \mathsf{Abort} \rangle_j$ messages it received from distinct parties j in Step 3 with verifying signatures into a *report* and sends them to the sender. If j receives no such messages, it sends $\langle \mathsf{NoMsg} \rangle_j$ to the sender instead.

Output Determination: If the sender i^* received $\langle \mathsf{Abort} \rangle_j$ with verifying signatures from at least $t + 1$ distinct parties (either directly in Step 3 or through reports in step 4), it sets $G_{i^*} = \mathsf{True}$. If i^* received messages from less than $n - t - s$ different parties in Step 4, it sets $Z_{i^*} = \mathsf{True}$ and $m_{i^*} = \perp$. Each party j outputs (m_j, Z_j, G_j) and terminates.

Fig. 1. An undead weak multicast protocol

Proofs of the following two claims are provided in Appendix B.1.

Lemma 1. *No non-Byzantine party j will send $\langle \mathsf{Abort} \rangle_j$ in step 3 of Π_{WMC} unless the designated sender has Byzantine or send faults.*

Theorem 1. *Protocol* Π_{WMC} *is a* (t, s, r)-secure *undead weak multicast protocol resilient to overlapping faults if* $n > 2t + s + r$.

4 Undead Uniform Consensus in the Total Omission Setting

This section deals with constructing a uniform consensus protocol in the total omission setting. That is, we construct an (s, r)-*secure* uniform consensus protocol that is resilient to s send faults and r receive faults when $s + r = n$. Constructing this protocol fills in a gap left by the lower bound presented in [11]. Their lower bound showed that in a setting where Byzantine failures are also allowed, $n > 2t + r + s$ must hold in order to solve the task. Setting $t = 0$, this would seem to imply that $n > r + s$ is required in the total omission setting. However, their lower bound assumes that the adversary is also allowed to actively "zombify" receive faulty parties throughout the protocol. This means that it can force them to stop participating in the protocol. While in some protocols [19,29] parties do stop participating if they detect their own faults, this assumption limits the generality of the result. In the protocol presented in this section, parties do detect their own faults in order to be able to output $\perp$ when required, but do not necessarily stop participating in the protocol. In fact, since this model does not consider mixed faults, a party can act upon finding out that it is receive faulty. By that we mean that such a party knows that its message will arrive at all parties that aren't receive faulty, and thus it can use that power to help push forward consensus. This work extends ideas of [22,23] to the total omission model, while using the syntax of [29,19] regarding zombification.

4.1 Very Weak Multicast

We first construct an (s, r)-*secure* very weak multicast protocol in the synchronous setting resilient to s send faults and r receive faults, where $s < n$ and $s + r = n$. In the protocol, the sender sends its message to all parties. Parties then forward the received message, or $\perp$ if no message was received. Finally, every party that received a large enough number of messages received a small number of messages (fewer than $n - s$) becomes a zombie. Every other party outputs the message received or forwarded from i^* if such a message exists, and $\perp$ otherwise. This protocol is described fully in Fig. 2.

A proof of the following claim is provided in Appendix B.2.

Lemma 2. *Protocol* Π_{VWMC} *is an* (s, r)-secure *very weak multicast protocol for any* s, r *such that* $s < n, s + r \leq n$ *without overlapping faults.*

4.2 Optimal Uniform Consensus

Now we construct an (s, r)-*secure* undead uniform consensus protocol Π_{TOC} in the total omission setting, i.e. with $n = s + r$. The protocol proceeds in $s +$

Protocol Π_{VWMC}

Input and Initialization: Each party $j \in [n]$ sets $Z_j := \mathsf{False}$, the designated sender i^* has input m. Every party j sets $m_j = \perp$.

Parties execute the following steps:

1. The sender $i^* \in [n]$ sends its input m to all other parties.
2. Every party j that has received a message m in the first step sets $m_j = m$. Party j sends m_j to all parties.

Output Determination: If party j has received messages from less than $n - s$ parties (in total) in Step 1 and 2, it sets $Z_j = \mathsf{True}$. If party j receives a message $m \neq \perp$ in either round 1 or round 2 and $Z_j = \mathsf{False}$, it sets $m_j = m$. Party j outputs (m_j, Z_j).

Fig. 2. A Very Weak Multicast Protocol

1 rounds. Each round has a designated sender, which is rotated in a round robin fashion. Each sender invokes the very weak multicast protocol with its current value and zombie flags one by one. Parties simply adopt any non-$\perp$ value they receive and continue propagating it in the next rounds. Intuitively, having $s+1$ such rounds guarantees that at least one of the rounds has a leader that is not send faulty. Every party will either receive that leader's message or become a zombie. Considering the latest such leader, all following leaders are only send faulty, and thus receive its message. This means that they will continue propagating its message in the following rounds, and thus its message will be all parties' output from the protocol. The protocol is provided in Fig. 3.

Protocol Π_{TOC}

Input and Initialization: Each party $j \in [n]$ inputs v_j, Z_j and sets $Z_j := \mathsf{False}, I_j = v_j$.

Parties execute the following steps, for every $i \in [s+1]$:

– Party i becomes the leader and invokes Π_{VWMC} on the input (I_i, Z_i). Let (m_j^i, Z_j^i) denote the output of party j in the i'th invocation of Π_{VWMC}. If $Z_j^i = \mathsf{True}$, j sets $Z_j = \mathsf{True}$. Then, if $Z_j = \mathsf{False}$ and $m_j^i \neq \perp$, j sets $I_j = m_j^i$.

Output Determination: If $Z_j = \mathsf{True}$, j terminates with output $(\perp, Z_j)$. Otherwise, it terminates with output (I_j, Z_j).

Fig. 3. A consensus protocol for $s + r \leq n$.

A proof of the following theorem is provided in Appendix B.3.

Theorem 2. *Protocol Π_{TOC} is an (s, r)-*secure *undead uniform consensus protocol for any s, r such that $s < n, s + r \leq n$ without overlapping faults.*

5 Lower Bounds

This section provide several new lower bounds that show the optimality of the presented uniform consensus protocol. We also show the impossibility of broadcast in the total omission conditions, despite being able to construct a uniform consensus protocol in this setting.

5.1 Total Send Corruption

The protocol in Section 4 works when $s + r \leq n$, as long as $s < n$. We start by showing that $s < n$ is a necessary condition in order to construct a uniform consensus protocol. Intuitively, we partition the parties into 2 groups and only allow parties to communicate within the groups. If one group receives one input and the other receives another, they would be forced to output different values and the protocol would not remain consistent. A proof of the following theorem is provided in Appendix D.1

Theorem 3. *There does not exist an* $(n, 0)$*-secure uniform consensus protocol.*

5.2 Total Omission Broadcast

One might expect that since uniform consensus can be solved in the total omission setting, broadcast would be solvable as well. Note that the uniform consensus construction in this paper uses the fact the receive faulty parties can push their inputs to all other parties, which can then be used to achieve consensus. However, in a broadcast protocol only one party's input is taken into consideration. If that party is send faulty, it will not be able to successfully push its value to all other parties, making this approach fail.

We formalize this by constructing a broadcast lower bound in the total omission setting. In this lower bound, we have the designated sender i^* communicate with a set A of $s - 1$ parties freely. The rest of the parties, denoted by a set B, do not hear anything from i^* or from A, either due to them being receive faulty, or due to the rest being send faulty. Parties in B must output some value, even without hearing anything from i^*, while the rest of the parties hear all sent messages and must be consistent with i^*'s input because the parties in B might simply be receive faulty. Since this can be made to take place even when parties in B are non faulty the rest are send faulty, we immediately break the consistency of the protocol. A proof of the following theorem is provided in Appendix D.2.

Theorem 4. *There is no* (s, r)-secure *broadcast protocol resilient to* s *send faults and* r *receive faults for any* s, r *such that* $s \geq 1, s + r = n$ *without overlapping faults.*

5.3 Consensus with Overlapping Faults

In this section, we prove that the threshold $s + r \leq n$ is necessary to achieve uniform consensus as long as $s > 2$. This shows the optimal corruption tolerance

of our uniform consensus protocol. The basic idea in the proof of Theorem 4 relies on the fact that hearing one party's messages (in that proof, the sender) is not reliable, because it could be send faulty and thus only a subset of the parties hear those messages. In the following proof, we use this idea to show that one could "switch" one party's input without changing the outputs of all parties. After switching all of the parties' inputs, we finally find either a validity violation or a consistency violation.

Below we prove the lower bound for the minimal case in which $s > 2$ and $s \neq n$, i.e. $n = 4$, $s = 3$ and $r = 2$. A proof of the general case is provided in Appendix D.3. In the lower bound for the minimal case, we have 4 parties, R_1, R_2, S_1, S_2. In all of the executions S_1, S_2 are send faulty and R_1, R_2 are receive faulty. R_1 and R_2 sometimes also exhibit overlapping send faults. In addition, R_1, R_2 hear nothing in all executions, and S_1, S_2 only hear messages from R_1, R_2, but not necessarily all of these message. We start off with all four parties having the input 1, and use the previous intuition to show that one could switch the inputs of R_1, R_2 gradually by making them send faulty as well. This is done in a series of executions, with each pair of consecutive executions having at least one party with the same view, forcing the same value to be output. Finally, after switching the inputs of R_1, R_2, it is easy to switch the two final inputs and reach either a consistency violation or a validity violation. Fig. 4 illustrates the executions used in the lower bound. The proof in Appendix D.3 uses the same strategy, simply switching all receive faulty parties' inputs one-by-one, and then switching the send faulty parties' inputs.

Theorem 5. *There is no* (s, r)-secure *uniform consensus protocol resilient to overlapping faults for any* s, r *s.t.* $s > 2$ *and* $s + r > n$.

Proof. We prove the lower bound holds for $n = 4, s = 3, r = 2$, and a proof for the general case is provided in Appendix D.3. Note that when $r < 2$, all parties can be send faulty, which was already proven impossible in Theorem 3. Assume there are four parties, R_1, R_2, S_1 and S_2. R_1 and R_2 are receive faulty in all executions, and have overlapping send faults in some of the executions. S_1 and S_2 are send faulty in all executions. The adversary drops all messages sent to R_1, R_2 and all messages sent from S_1, S_2 in all executions. This means that in all executions R_1 and R_2 hear no messages, and S_1, S_2 hear only messages from R_1 and R_2, but might not hear some in executions where they have overlapping send faults as well. In all of the following descriptions, parties S_1, S_2 hear all messages sent by R_1, R_2, unless explicitly stated otherwise.

1. In the first execution, all parties start with input 1 and no parties has overlapping faults.
2. In the second execution, R_1 has overlapping send faults. All parties start with input 1. The adversary drops all message sent from R_1 to S_2, but delivers messages from R_1 to S_1.
3. In the third execution, R_1 has overlapping send faults. R_1 starts with input 0 and all other parties have the input 1. The adversary drops all message sent from R_1 to S_2, but delivers messages from R_1 to S_1.

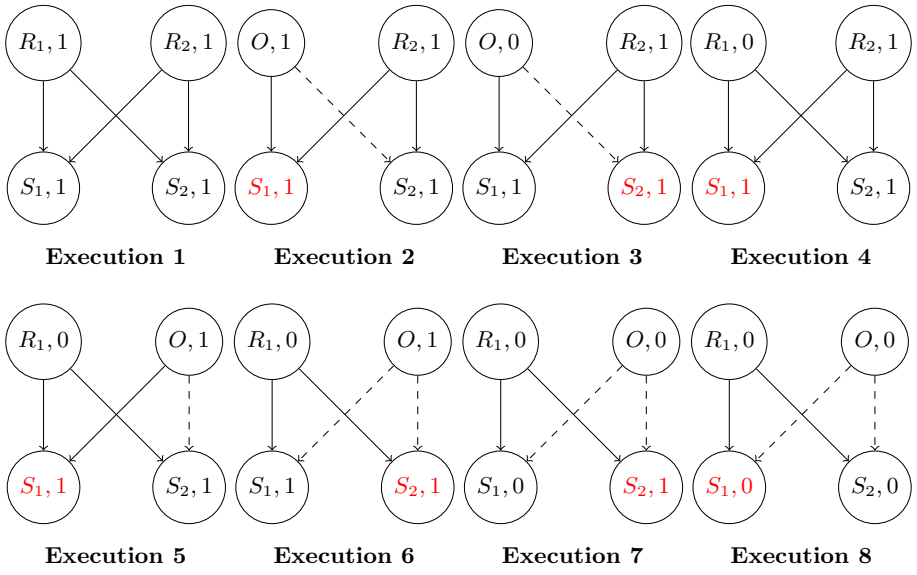

Fig. 4. Lower Bound Executions. A node with the text P, b indicates that party P has input b. Parties named R_i are receive faulty, parties named S_i are send faulty, and parties named O have overlapping faults. Arrows indicate messages are successfully sent, dashed arrows indicate that message are dropped. Parties whose names are colored red have views indistinguishable from the previous execution.

4. In the forth execution, R_1 starts with input 0 and all other parties have input 1. All messages from R_1, R_2 are delivered to S_1, S_2.
5. In the fifth execution, R_2 has overlapping send faults. All messages sent from R_2 to S_2 are dropped, but messages from R_2 to S_1 are delivered. R_1 starts with input 0, and all other parties start with input 1.
6. In the sixth execution, R_2 has overlapping send faults. All messages sent from R_2 are dropped by the adversary. R_1 and R_2 start with input 0, while S_1 and S_2 starts with input 1.
7. In the seventh execution, R_2 has overlapping send faults. All messages sent from R_2 are dropped by the adversary. S_2 starts with input 1 and all other parties start with input 0.
8. In the eighth execution, R_2 has overlapping send faults. All messages sent from R_2 are dropped by the adversary. All parties start with input 0.

By the validity requirements, parties S_1 and S_2 must output 1 in execution 1. S_1 has indistinguishable views in executions 1 and 2, so it outputs 1 in execution 2. Since the protocol is consistent, S_2 does so as well. On the other hand, S_2 has indistinguishable views in executions 2 and 3 since it hears nothing from R_1 in both. Therefore, it outputs 1 in execution 3 and therefore S_1 does so too. S_1's view is identical in executions 3 and 4 and thus it outputs 1 in execution 4, and S_2

outputs 1 as well due to consistency. Executions 4 and 5 are indistinguishable from S_1's point of view, so it outputs 1 in execution 5. From consistency, S_2 outputs 1 too in execution 5. S_2 has identical views in executions 5 and 6, so it must output 1, and so does S_1 from consistency. S_2's views in execution 7 is indistinguishable from its view in execution 6, so it outputs 1, and so does S_1. Finally, S_1's views in executions 7 and 8 are indistinguishable, and thus it outputs 1 in both. However, all parties have the input 0 in execution 8. This means that from validity, S_1 must output 0 in execution 8, reaching a contradiction.

References

1. Abraham, I., Dolev, D., Kagan, A., Stern, G.: Brief announcement: Authenticated consensus in synchronous systems with mixed faults. In: Scheideler, C. (ed.) 36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA. LIPIcs, vol. 246, pp. 38:1–38:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.DISC.2022.38, https://doi.org/10.4230/LIPIcs.DISC.2022.38
2. Bessani, A.N., Sousa, J., Alchieri, E.A.P.: State machine replication for the masses with BFT-SMART. In: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014. pp. 355–362. IEEE Computer Society (2014). https://doi.org/10.1109/DSN.2014.43, https://doi.org/10.1109/DSN.2014.43
3. Brazitikos, K., Zikas, V.: General adversary structures in byzantine agreement and multi-party computation with active and omission corruption. Cryptology ePrint Archive (2024)
4. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (mar 1996). https://doi.org/10.1145/226643.226647, https://doi.org/10.1145/226643.226647
5. Chun, B., Maniatis, P., Shenker, S., Kubiatowicz, J.: Attested append-only memory: making adversaries stick to their word. In: Bressoud, T.C., Kaashoek, M.F. (eds.) Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007. pp. 189–204. ACM (2007). https://doi.org/10.1145/1294261.1294280, https://doi.org/10.1145/1294261.1294280
6. Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., Riché, T.: Upright cluster services. In: Matthews, J.N., Anderson, T.E. (eds.) Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009. pp. 277–290. ACM (2009). https://doi.org/10.1145/1629575.1629602, https://doi.org/10.1145/1629575.1629602
7. Correia, M., Lung, L.C., Neves, N.F., Veríssimo, P.: Efficient byzantine-resilient reliable multicast on a hybrid failure model. In: 21st Symposium on Reliable Distributed Systems (SRDS 2002), 13-16 October 2002, Osaka, Japan. pp. 2–11. IEEE Computer Society (2002). https://doi.org/10.1109/RELDIS.2002.1180168, https://doi.org/10.1109/RELDIS.2002.1180168
8. Correia, M., Neves, N.F., Veríssimo, P.: How to tolerate half less one byzantine nodes in practical distributed systems. In: 23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil. pp. 174–183. IEEE Computer Society (2004). https://doi.org/10.1109/RELDIS.2004.1353018, https://doi.org/10.1109/RELDIS.2004.1353018

9. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983). https://doi.org/10.1137/0212045, https://doi.org/10.1137/0212045
10. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)
11. Eldefrawy, K., Loss, J., Terner, B.: How byzantine is a send corruption? In: Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings. p. 684–704. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-09234-3_34, https://doi.org/10.1007/978-3-031-09234-3_34
12. Fitzi, M., Maurer, U.: Efficient byzantine agreement secure against general adversaries. In: Distributed Computing: 12th International Symposium, DISC'98 Andros, Greece, September 24–26, 1998 Proceedings 12. pp. 134–148. Springer (1998)
13. Garay, J.A., Perry, K.J.: A continuum of failure models for distributed computing. In: Segall, A., Zaks, S. (eds.) Distributed Algorithms, 6th International Workshop, WDAG '92, Haifa, Israel, November 2-4, 1992, Proceedings. Lecture Notes in Computer Science, vol. 647, pp. 153–165. Springer (1992). https://doi.org/10.1007/3-540-56188-9_11, https://doi.org/10.1007/3-540-56188-9_11
14. Golan-Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M.K., Seredinschi, D., Tamir, O., Tomescu, A.: SBFT: A scalable and decentralized trust infrastructure. In: 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019. pp. 568–580. IEEE (2019). https://doi.org/10.1109/DSN.2019.00063, https://doi.org/10.1109/DSN.2019.00063
15. Hadzilacos, V.: Issues of fault tolerance in concurrent computations (databases, reliability, transactions, agreement protocols, distributed computing). Ph.D. thesis, Harvard University (1985)
16. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176, https://doi.org/10.1145/357172.357176
17. Levin, D., Douceur, J.R., Lorch, J.R., Moscibroda, T.: Trinc: Small trusted hardware for large distributed systems. In: Rexford, J., Sirer, E.G. (eds.) Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA. pp. 1–14. USENIX Association (2009), http://www.usenix.org/events/nsdi09/tech/full_papers/levin/levin.pdf
18. Lincoln, P., Rushby, J.M.: A formally verified algorithm for interactive consistency under a hybrid fault model. In: Digest of Papers: FTCS-23, The Twenty-Third Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22-24, 1993. pp. 402–411. IEEE Computer Society (1993). https://doi.org/10.1109/FTCS.1993.627343, https://doi.org/10.1109/FTCS.1993.627343
19. Loss, J., Stern, G.: Zombies and ghosts: Optimal byzantine agreement in the presence of omission faults. In: Theory of Cryptography Conference. pp. 395–421. Springer (2023)
20. Micali, S., Rogaway, P.: Secure computation. Springer (1992)
21. Parvédy, P.R., Raynal, M.: Uniform agreement despite process omission failures. In: 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings. p. 212. IEEE Computer Society (2003). https://doi.org/10.1109/IPDPS.2003.1213388, https://doi.org/10.1109/IPDPS.2003.1213388

22. Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. IEEE Trans. Software Eng. **12**(3), 477–482 (1986). https://doi.org/10.1109/TSE.1986.6312888, https://doi.org/10.1109/TSE.1986.6312888
23. Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: 2002 Pacific Rim International Symposium on Dependable Computing, 2002. Proceedings. pp. 221–228. IEEE (2002)
24. Serafini, M., Bokor, P., Dobre, D., Majuntke, M., Suri, N.: Scrooge: Reducing the costs of fast byzantine replication in presence of unresponsive replicas. In: Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010, Chicago, IL, USA, June 28 - July 1 2010. pp. 353–362. IEEE Computer Society (2010). https://doi.org/10.1109/DSN.2010.5544295, https://doi.org/10.1109/DSN.2010.5544295
25. Serafini, M., Suri, N.: The fail-heterogeneous architectural model. In: 26th IEEE Symposium on Reliable Distributed Systems (SRDS 2007), Beijing, China, October 10-12, 2007. pp. 103–113. IEEE Computer Society (2007). https://doi.org/10.1109/SRDS.2007.33, https://doi.org/10.1109/SRDS.2007.33
26. Siu, H.S., Chin, Y.H., Yang, W.P.: Byzantine agreement in the presence of mixed faults on processors and links. IEEE Transactions on Parallel and Distributed Systems **9**(4), 335–345 (1998). https://doi.org/10.1109/71.667895
27. Thambidurai, P.M., Park, Y.: Interactive consistency with multiple failure modes. In: Seventh Symposium on Reliable Distributed Systems, SRDS 1988, Columbus, Ohio, USA, October 10-12, 1988, Proceedings. pp. 93–100. IEEE Computer Society (1988). https://doi.org/10.1109/RELDIS.1988.25784, https://doi.org/10.1109/RELDIS.1988.25784
28. Veronese, G.S., Correia, M., Bessani, A.N., Lung, L.C., Veríssimo, P.: Efficient byzantine fault-tolerance. IEEE Trans. Computers **62**(1), 16–30 (2013). https://doi.org/10.1109/TC.2011.221, https://doi.org/10.1109/TC.2011.221
29. Zikas, V., Hauser, S., Maurer, U.M.: Realistic failures in secure multi-party computation. In: Reingold, O. (ed.) Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5444, pp. 274–293. Springer (2009). https://doi.org/10.1007/978-3-642-00457-5_17, https://doi.org/10.1007/978-3-642-00457-5_17

A Standard Task Definitions

A.1 Uniform Consensus

Definition 4. *Let Π be a protocol executed by parties $1, \ldots, n$, where each party $j \in [n]$ starts with input $m_j \neq \perp$. Every party j outputs x_j at the end of the protocol.*

- ***Validity.*** *Π is* (s, r)-valid *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: If each party j starts with the same value $m_j = m$, all parties that are not receive faulty output m, and receive faulty parties output m or $\perp$ at the end of the protocol.*
- ***Consistency.*** *Π is* (s, r)-consistent *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: All non-faulty parties and send faulty parties output $x_j = m$ for the same value m at the end of the protocol. In addition, every receive faulty party either outputs m or $\perp$.*
- ***Termination.*** *Π is* (s, r)-terminating *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: Each party j terminates and outputs x_j at the end of the protocol.*

If Π is (s, r)-valid, (s, r)-consistent *and* (s, r)-terminating *we say that it is an* (s, r)-secure *uniform consensus protocol.*

A.2 Broadcast

Definition 5. *Let Π be a protocol executed by parties $1, \ldots, n$, where i^* is the designated sender starting with an input $m \neq \perp$. Every party j outputs x_j at the end of the protocol.*

- ***Validity.*** *Π is* (s, r)-valid *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: If the sender is non-faulty or receive faulty, every party that isn't receive faulty outputs m and receive faulty parties output m or $\perp$ at the end of the protocol.*
- ***Consistency.*** *Π is* (s, r)-consistent *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: Every non-faulty or send faulty party j outputs $x_j = v$ for the same value $v \in \{m, \perp\}$ at the end of the protocol. In addition, every receive faulty party outputs $x_j = v$ or $x_j = \perp$.*
- ***Termination.*** *Π is* (s, r)-terminating *if the following holds whenever at most s parties are send faulty and r parties are receive faulty: Each party j terminates and outputs x_j at the end of the protocol.*

If Π is (s, r)-valid, (s, r)-consistent *and* (s,r)-terminating *we say that it is an* (s, r)-secure *broadcast protocol.*

B Proofs of Protocols

B.1 Proofs for Undead Weak Multicast

Lemma 1. *No non-Byzantine party j will send $\langle \mathsf{Abort} \rangle_j$ in step 3 of Π_{WMC} unless the designated sender has Byzantine or send faults.*

Proof. Assume there is a non-Byzantine party j that sends $\langle \mathsf{Abort} \rangle_j$. It must have received $\langle \perp \rangle$ from at least $n-t-s \geq t+r+1$ different parties, and at least one of those messages is from a party without receive omission or Byzantine faults. That party would have received the message if the designated sender is neither send faulty nor Byzantine.

Theorem 1. *Protocol Π_{WMC} is a (t,s,r)-*secure *undead weak multicast protocol resilient to overlapping faults if $n > 2t+s+r$.*

Proof. **Validity.** If i^* is non-faulty or is receive faulty and alive in the beginning of the protocol, it sends $\langle m \rangle_{i^*}$ to every party and all non-faulty parties receive it and forward it. Every party that isn't receive faulty receives those messages. Since i^* is non-Byzantine, it only sends one verifying signature and thus this is the only received value, which will then be output. In addition, if a receive faulty party j does not output $Z_j = \mathsf{True}$, then it receives messages from at least $n-t-s \geq 2t+s+r+1-t-s = t+r+1$ different parties, then at least one of those messages was sent by a party that is neither Byzantine nor receive faulty party. Those parties receive the message m from i^* and forward it, and thus every $j \neq i^*$ that hasn't become a zombie, outputs m. In addition, if $Z_{i^*} \neq \mathsf{True}$ by the end of the protocol, it outputs (m, Z_{i^*}, G_{i^*}) as well. Finally, if i^* is send faulty, it only signs the message m. This means that no non-faulty party receives another signed message from i^*, and thus output either m or $\perp$.

Detection. If $Z_{i^*} = \mathsf{False}$ and $G_{i^*} = \mathsf{False}$ at the end of the protocol, i^* receives messages from at least $n-t-s$ parties in step 4 with and receives Abort messages from at most t parties. Since $n-t-s \geq t+r+1$, i^* received the report sent by at least one party that is neither Byzantine nor receive faulty. This party heard all Abort messages sent by all non-faulty parties. In other words, i^* either directly or indirectly heard all Abort messages sent by non-faulty parties. Since in total it received Abort messages from at most t parties, at least one non-faulty party did not send such a message. This party received i^*'s message and will output it in the end of the protocol.

Termination. All parties terminate after exactly 4 rounds.

No Living Undead. In the end of the protocol, every party j outputs (m_j, Z_j, G_j). This means that we need to argue that only receive faulty parties set $Z_j = \mathsf{True}$ and only send faulty parties set $G_j = \mathsf{True}$.

- NLU of the designated sender: If the sender is non-faulty, by Lemma 1, at most t parties p send $\langle \mathsf{Abort} \rangle_p$ messages, and thus i^* does not set $G_{i^*} = \mathsf{True}$. In addition, it will receive at least $n-t-s$ many messages in the step 4, so it won't set $Z_{i^*} = \mathsf{True}$ either.

– NLU of all other parties: Each non-receive-faulty party j must receive at least $n-t-s$ messages in the step 3, so they won't set $Z_j = \mathsf{True}$ and they don't set $G_j = \mathsf{True}$ anywhere in the protocol.

B.2 Proofs for Very Weak Multicast

Lemma 2. *Protocol* Π_{VWMC} *is an* (s, r)-secure *very weak multicast protocol for any* s, r *such that* $s < n, s + r \leq n$ *without overlapping faults.*

Proof. **Validity.** For the first part of the property, note that parties only output a value other than $\perp$ if they received i^*'s message containing m, or another party's message forwarding m. In either case, parties only send i^*'s input. For the second part of the property, we start by considering a non-faulty sender i^* and assuming that there are fewer than r receive faulty parties. If some party j output (x, Z_j) with $Z_j = \mathsf{False}$, it must have received messages from at least $n - s > r - 1$ many parties. By assumption, at most $r - 1$ parties are receive faulty, so at least one of the messages comes was received from a party that is not receive faulty. Those receive i^*'s message in round 1 and forward it, and thus in both cases j receives the sender's input, sets $m_j = m$ and outputs this value.

Now assume the sender is receive faulty. If some party j output (x, Z_j) with $Z_j = \mathsf{False}$, it must have received messages from at least $n - s \geq r$ many parties. If it received messages from all receive faulty parties, then it received one from i^* as well and set $m_j = m$. Otherwise, it received at least one message from a send-faulty party. That party does not exhibit any receive faults, so it received m from i^* and forwarded that message to j. In either case j sets $m_j = m$ and outputs that value.

Termination. The protocol Π_{VWMC} terminates in exactly 2 rounds.

No Living Undead. Assume that some party j outputs $Z_j = \mathsf{True}$. It only sets Z_j to True if it receives messages from a total of less than $n - s$ many different parties in Steps 2 and 3 of Π_{VWMC}. This means that j did not receive a message from at least one party that is not send faulty. This implies that j must be a receive faulty party, as required.

B.3 Proofs for Total Omission Consensus

Theorem 2. *Protocol* Π_{TOC} *is an* (s, r)-secure *undead uniform consensus protocol for any* s, r *such that* $s < n, s + r \leq n$ *without overlapping faults.*

Proof. **Validity.** Assume all parties start with the same input m. This means that they all set $I_j = m$ in the beginning of the protocol. We will show that if all parties have $I_j = m$ in the beginning of a round, this will continue to hold in the end of the round. Following a simple inductive argument, this means that all parties have $I_j = m$ in the end of round $s + 1$, and thus either output (I_j, False) if $Z_j = \mathsf{False}$ or $(\perp, \mathsf{True})$ otherwise. Assume that all parties have $I_j = m$ in the beginning of round r. This means that the round's leader sends (m, Z_r) in the Π_{VWMC} protocol. From the validity property of the protocol, all parties receive

(m', Z) such that $m' \in \{m, \perp\}$. This means that each j either updates I_j to m if it output $m' = m$, or does not update I_j at all otherwise. Therefore all parties have $I_j = m$ in the end of the round as well.

Consistency. First assume that at least one of the first $s + 1$ leaders is receive faulty, and let l be the maximal such party. If that is not the case, then there are no receive faulty parties among the first $s + 1$ leaders. Since there are at most s send faulty parties, at least one of these parties is non-faulty. In addition, if at least $s + 1$ parties are not receive faulty, then there are at most $n - (s + 1) \leq r - 1$ receive faulty parties. Let l be the nonfaulty leader with the maximal index such that $l \leq s + 1$. In either case, let m' be the message sent by the leader l. From the validity property of Π_{VWMC}, every party j either receives the message m' and sets $I_j = m'$ or sets $Z_j = \mathsf{True}$ at the end of round l.

We will show that in every subsequent round (if such a round exists), every party is either receive faulty with $Z_j = \mathsf{True}$ or has $I_j = m'$. Note that in both of the above cases, there are no receive faulty leaders after round l. That is, parties $l + 1, \ldots, s + 1$ are not receive faulty. First, since parties $l + 1, \ldots, s + 1$ are not receive faulty, from the Validity and No Living Undead properties of Π_{VWMC}, each such j outputs m', False from the protocol in round l and updates $I_j = m'$. Therefore, in the beginning of round $l + 1$, the leader sends the value $I_{l+1} = m'$ in the Π_{VWMC} protocol and thus from the Validity property, every party either outputs m' or $\perp$. This means that every party that updates its I_j variable in round $l+1$ updates it to m'. Following identical logic, the same holds for rounds $l + 2, \ldots, s + 1$, and thus parties don't update I_j to any value other than m' following round l. Finally, every party that has $Z_j = \mathsf{True}$ outputs $(\perp, \mathsf{True})$ from the protocol. Every other party updated $I_j = m'$ in round l, and possibly in following rounds as well, and finally output (m', False).

Termination. The protocol terminates after $s + 1$ rounds and each party j will output (m_j, Z_j).

No Living Undead. Parties only set $Z_j = \mathsf{True}$ in the protocol if they output (x, True) in one of the invocations of Π_{VWMC}. From no living undead property of Π_{VWMC}, only receive faulty parties do so, as required.

C Undead Graded Multicast

C.1 definition

An undead graded multicast protocol has a designated sender i^* with input m. Every party i also has two flags $z_i, g_i \in \{\mathsf{True}, \mathsf{False}\}$ as input indicating whether it is already a zombie or a ghost when starting the protocol. Every party outputs a value, as well as grade $y \in \{0, 1, 2\}$, indicating whether it thinks that the sender succeeded in propagating its message. If some party thinks that a non Byzantine sender definitely succeeded in sending its message (outputs $y = 2$), then all parties actually received that message or became zombies. The protocol is formally defined bellow.

Definition 6. *Let Π be a protocol executed by parties $1, \ldots, n$, with a designated sender i^* starting with an input $m \neq \perp$. In addition, every party i has two values $z_i, g_i \in \{\mathsf{True}, \mathsf{False}\}$ as input. Every party outputs a tuple (x, y, z, g) such that x is either a possible message or $\perp$, $y \in \{0, 1, 2\}$ and z, g are boolean values.*

- ***Validity.*** *Π is* (t, s, r)-valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: If i^* non-faulty, every non-Byzantine party j outputs (x, y, z, g) such that either $x = m, y = 2$, or such that $z = \mathsf{True}$. In addition, if i^* is send faulty, no non-Byzantine party outputs (x, y, z, g) such that $x \notin \{m, \perp\}$.*
- ***Detection.*** *Π is* (t, s, r)-detecting *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: If i^* is send faulty and it is alive at the end of the protocol, every non-faulty party output (x, y, z, g) such that $x = m$ and $y \geq 1$.*
- ***Consistency.*** *Π is* (t, s, r)-consistent *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: If i^* is non-Byzantine, for every two non-Byzantine parties j, k that output (x_j, y_j, z_j, g_j) and (x_k, y_k, z_k, g_k) respectively, either $|y_j - y_k| \leq 1$, or at least one of z_j, z_k equals True. In addition, either $x_j = \perp$ and $y_j = 0$, or $x_j = m$.*
- ***Termination.*** *Π is* (t, s, r)-terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: All parties complete the protocol and output a value.*
- ***No Living Undead.*** *Π is* (t, s, r)-no living undead *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: If a non-Byzantine party j outputs (x, y, z, g) such that $z = \mathsf{True}$ (resp. $g = \mathsf{True}$), then it is receive faulty (resp. send faulty).*

If Π is (t, s, r)-valid, (t, s, r)-detecting, (t, s, r)-consistent, (t, s, r)-terminating, *and* (t, s, r)-no living undead *we say that it is a* (t, s, r)-secure *undead graded multicast protocol.*

The only difference from the original definition, provided in [19], is in the validity property. In the original property, validity was required to hold when i^* is either receive faulty or non-faulty. This is much harder to achieve if overlapping faults are allowed. That is because originally, a zombie could send a message and know that it will arrive even after finding out it is receive faulty. However, a party with overlapping omission faults might become a zombie instead of becoming a ghost and thus won't know whether its future messages will arrive at their destination. Therefore, in this definition we only require the validity to hold when the sender is non-faulty. Thankfully this is enough, as the proofs of [19] only rely on non-faulty parties successfully sending their messages.

C.2 Construction

In the protocol, the sender starts by sending its input to all parties in an undead weak multicast protocol. Following that, every party that is still alive forwards

the received message in an undead weak multicast protocol as well. If a party received a message from i^* in both rounds it knows that it did not become a ghost, and thus at least one non-faulty party received its message in the first round. This means that it can output $m, 2$ and be assured that all other parties will receive the message from that non-faulty party and be able to output m with a grade of at least 1, or become zombies.

Protocol Π_{GMC}

Input and Initialization: Each party $j \in [n]$ has inputs z_j, g_j, the designated sender i^* also has input m. Every party j initializes $m_j = \perp$, $Z_j = z_j$ and $G_j = g_j$.

Parties execute the following steps:

1. The sender i^* sends $\langle m \rangle_{i^*}$ to all other parties using Π_{WMC} if $Z_{i^*} = \mathsf{False}$ and $G_{i^*} = \mathsf{False}$.
2. Let x_j, Z'_j, G'_j be j's output in the previous round. If $Z'_j = \mathsf{True}$, j sets $Z_j = \mathsf{True}$, and if $G'_j = \mathsf{True}$, j sets $G_j = \mathsf{True}$. Every party that receives $x = \langle m' \rangle_{i^*}$ (i.e. a message m' with a verifying signature from i^*) sets $m_j = \langle m' \rangle_{i^*}$. Every party j that has $Z_j = G_j = \mathsf{False}$ sends m_j using Π_{WMC}.

Output Determination: For every pair j, k, define $x_{j,k}, Z_{j,k}, G_{j,k}$ to be j's output in the Π_{WMC} protocol with k as sender. If there exists a k such that $Z_{j,k} = \mathsf{True}$, j sets $Z_j = \mathsf{True}$. In addition, if there exists a k such that $G_{j,k} = \mathsf{True}$, j sets $G_j = \mathsf{True}$. If $Z_j = \mathsf{False}$, $m_j = \langle m' \rangle_{i^*} \neq \perp$ and $x_{j,i^*} = m_j$, j outputs $(m', 2, Z_j, G_j)$. Otherwise, if $Z_j = \mathsf{False}$ and for some k, $x_{j,k} = \langle m' \rangle_{i^*}$ with a verifying signature from i^*, j outputs $m', 1, Z_j, G_j$ (choose one arbitrarily if more than one exists). Finally, if j did not output a value in either of the previous cases, it outputs $\perp, 0, Z_j, G_j$.

Fig. 5. An undead graded multicast protocol

C.3 Security Proof

Theorem 6. *Protocol Π_{GMC} is a (t, s, r)-*secure *undead graded multicast protocol resilient to overlapping faults if $n > 2t + s + r$.*

Proof. **Validity.** If i^* is non-faulty, then from the Validity of the Π_{WMC} protocol, every non-Byzantine party j outputs $\langle m \rangle_{i^*}$ in the weak multicast instances with i^* as sender in both rounds or outputs either $Z'_j = \mathsf{True}$ or $Z_{j,i^*} = \mathsf{True}$. In the first case, j outputs $m, 2, z, g$ in the end of the protocol, and in the second it outputs $\perp, 0, \mathsf{True}, g$. Note that from the no living undead property of Π_{WMC}, i^* has $Z'_{i^*} = G'_{i^*} = \mathsf{False}$ so it does send $\langle m \rangle_{i^*}$ in the second round as well.

Detection. Assume i^* is send faulty and is alive in the end of the protocol. This means that it has $Z_{i^*} = G_{i^*} = \mathsf{False}$ in the end of the protocol and outputs

these flags. It did not set $Z_{i^*} = \mathsf{True}$ or $G_{i^*}\,\mathsf{True}$ in the beginning of round 2, so it output $Z'_{i^*} = G'_{i^*} = \mathsf{False}$. From the detection and no living undead properties of the Π_{WMC} protocol, some non-faulty party k output $\langle m \rangle_{i^*}, \mathsf{False}, \mathsf{False}$ from the first call to Π_{WMC}. Party j then sent that message in the Π_{WMC} protocol, and from the validity and detection properties of the protocol every non-Byzantine party outputs $x_{j,k}, Z_{j,k}, G_{j,k}$ such that either $x_{j,k} = \langle m \rangle_{i^*}$ or has $Z_{j,k} = \mathsf{True}$. If $Z_{j,k}$, then j sets $Z_j = \mathsf{True}$ and outputs $\perp, 0, \mathsf{True}, G_j$. From the no living undead property, this only takes place if j is receive faulty. On the other hand, if $x_{j,k} = \langle m \rangle_{i^*}$, then j received m with a valid signature from i^*. Since i^* only signs one message, m is the only message j will receive with a valid signature, and thus j outputs m, y, z, g with $y = 1$ if it hasn't done so with $y = 2$ already.

Consistency. Assume i^* is not Byzantine and let j, k be two non-Byzantine parties that output (x_j, y_j, z_j, g_j) and (x_k, y_k, z_k, g_k) respectively. If either $z_j = \mathsf{True}$ or $z_k = \mathsf{True}$, then the first part of the property holds. The first part of the property also holds if neither party outputs a grade of 2, i.e. if $y_j \neq 2$ and $y_k \neq 2$, In order to prove the final case, assume w.l.o.g that $y_j = 2$. If that is the case, it received the same message $\langle m' \rangle_{i^*}$ from i^* in both calls to Π_{WMC}. Since i^* is not Byzantine and from the validity of the Π_{WMC} protocol, i^* sent the message m' in both of these calls. If i^* is either non-faulty or receive faulty, from the validity of the Π_{WMC} protocol, every nonfaulty party received that message in the first call to Π_{WMC}. If i^* is send faulty, from the detection of the protocol, at least one non-faulty party l receive the message. Following that, l sent the message in the second round. By assumption, k outputs $z_k = \mathsf{False}$. This means that it output $Z_{k,l} = \mathsf{False}$ as well, because otherwise it would have become a zombie and output $z_k = \mathsf{True}$. By validity, this means that it also output $x_{k,l} = \langle m' \rangle_{i^*}$. Finally, during the output determination, if k doesn't output $(m', 2, z_k, g_k)$, it will reach the second condition and output $(m', 1, z_k, z_k)$ instead. For the second part of the property, if j receives some message $\langle m' \rangle_{i^*}$ with a verifying signature by i^*, it outputs $x_j = m'$. Since i^* is not Byzantine it only signs its input m, so $x_j = m$. If j does not receive such a message, then it output $\perp, 0, Z_j, G_j$, as required.

Termination. The protocol after exactly two calls to Π_{WMC}.

No Living Undead. Party j outputs the flags Z_j, G_j in the end of the protocol. It sets $Z_j = \mathsf{True}$ if $Z'_j = \mathsf{True}$ or $Z_{j,k} = \mathsf{True}$ for some k. Similarly, it only sets $G_j = \mathsf{True}$ if $G'_j = \mathsf{True}$ or $G_{j,k} = \mathsf{True}$ for some k. The values Z'_j, G'_j as well as $Z_{j,k}, G_{j,k}$ are the flags output from the Π_{WMC} protocol. From the no living undead property of Π_{WMC}, Z'_j or $Z_{j,k}$ only equal True if j is receive faulty. Similarly, G'_j or $G_{j,k}$ only equal True if j is send faulty, as required.

D Lower Bound Proofs

D.1 Total Send Corruption Uniform Consensus

Theorem 3. *There does not exist an $(n, 0)$-secure uniform consensus protocol.*

Proof. Assume there exists an $(n, 0)$-secure uniform consensus protocol. Let $A, B \subseteq [n]$ be an arbitrary partition of $[n]$ to non-empty sets. That is $A \cup B = [n]$, $A \cap B = \emptyset$ and neither A nor B are empty. We build three executions as follows.

1. In the first execution, all parties start with the input 1. The parties in A are non-faulty and the parties in B are send faulty. The adversary drops all messages sent from parties in group B to parties in group A. By the validity requirements, all parties in group A must output 1.
2. In the first execution, all parties start with the input 0. The parties in A are send faulty and the parties in B are non-faulty. The adversary drops all messages sent from parties in group A to parties in group B.
3. In the third execution, all parties are send faulty. Parties in A start with input 1 and parties in group B start with input 0. The adversary drop all messages sent between parties in group A and in group B, but delivers all messages within each of the groups.

All parties in group A have indistinguishable views in execution 1 and in execution 3, and thus must output 1 in both. On the other hand, all parties in group B have indistinguishable views in execution 2 and execution 3 and thus must output 0 in both. Parties in execution 3 output different values, and thus the protocol does not have the consistency property, reaching a contradiction.

D.2 Total Omission Broadcast

Theorem 4. *There is no* (s, r)-secure *broadcast protocol resilient to* s *send faults and* r *receive faults for any* s, r *such that* $s \geq 1, s + r = n$ *without overlapping faults.*

Proof. Assume there exists an (s, r)-*secure* broadcast protocol for $s \geq 1, s + r = n$. Let i^* be the index of the designated sender, and let A, B be an arbitrary partition of $[n] \setminus \{i^*\}$ such that $|A| = s - 1$ and $|B| = r$. For example, let A be the $s - 1$ minimal indices that are not i^*, and let $B = [n] \setminus (A \cup \{i^*\})$. In all of the following executions i^* and parties in A successfully communicate with each other and within A, and hear messages sent by parties in B. On the other hand, parties in B hear nothing from i^* or from parties in A, but communicate successfully among themselves. We construct four executions as follows.

1. In the first execution, i^* has the input 0. Parties in A and i^* are send faulty, but parties in B are non faulty. Parties in A and i^* communicate among themselves, but due to their send faults all messages to B are dropped. In addition, parties in B communicate among themselves, and any messages they send arrive at A and i^*.
2. In the second execution, i^* has the input 1. Parties in A and i^* are send faulty, but parties in B are non faulty. Parties in A and i^* communicate among themselves, but due to their send faults all messages to B are dropped. In addition, parties in B communicate among themselves, and any messages they send arrive at A and i^*.

3. In the third execution, i^* has the input 0. Parties in A and i^* are non-faulty, but parties in B are receive faulty. Parties in A and i^* communicate among themselves, but all messages to B are dropped due to their receive faults. In addition, parties in B communicate among themselves, and any messages they send arrive at A and i^*.
4. In the fourth execution, i^* has the input 1. Parties in A and i^* are non-faulty, but parties in B are receive faulty. Parties in A and i^* communicate among themselves, but all messages to B are dropped due to their receive faults. In addition, parties in B communicate among themselves, and any messages they send arrive at A and i^*.

Note that in the third and fourth executions, i^* and parties in A are non-faulty. From validity they must all output 0 and 1 in executions 3 and 4 respectively. In addition, the view of these parties in executions 1 and 3 are indistinguishable, as well as their views in executions 2 and 4. This means that they must also output 0 and 1 in executions 1 and 2 respectively. On the other hand, parties in B have identical views in all 4 executions, and thus act the same in all of them. These parties are non-faulty in executions 1 and 2 and thus must output consistent values. If parties in B output a value $v \neq 0$[4] in all executions, then this leads to a consistency violation in execution 1. Similarly, if they output a value $v \neq 1$ in all executions, then this leads to a consistency violation in execution 2.

D.3 Generalizing Theorem 5

The technique in Theorem 5 showed that it is possible to take one receive faulty party and switch its input, but all parties must still output the same value. In order to prove the general result, one could start with all parties having the input 1, and then switching their inputs one-by-one until all of them have the input 0. Finally, if the send faulty parties' messages are all dropped, their inputs can also be switched without other parties noticing. Finally, we will find that all parties have the input 0, but output the value 1, which will contradict the validity of the protocol. A formal proof of the general case of Theorem 5 is provided below.

Theorem 5. *There is no* (s, r)-secure *uniform consensus protocol resilient to overlapping faults for any* s, r *s.t.* $s > 2$ *and* $s + r > n$.

Proof. Assume there exists an (s, r)-*secure* uniform consensus protocol resilient to overlapping faults for some s, r such that $s > 2$ and $s+r > n$. For every $i \in [r]$ let $R_i = i$ and for every $j \in [s]$ let $S_j = r + j$. In all of the executions, parties $R_1, \ldots, R_r$ will be receive faulty (and one possibly send faulty as well) and parties $S_1, \ldots, S_s$ will be send faulty. In all of the executions Parties $R_1, \ldots, R_r$ will receive no message due to their receive faults, and the messages sent by $S_1, \ldots, S_s$ will be dropped. This means that the S_j parties will hear from every party

[4] Technically, in probabilistic protocols there might not be only one possible output in this case. Since this proof deals with perfectly secure protocols, it is enough that there is a positive probability they output $v \neq 0$.

R_i, except for possibly those that are also send faulty in particular executions. Messages sent from every R_i in all executions must be identical since all receive faulty parties hear nothing in all executions. We will show inductively on $i \in \{0, \ldots, r\}$ that in the following conditions all send faulty parties must output 1:

- parties $R_1, \ldots, R_r$ are receive faulty, parties $S_1, \ldots, S_s$ are send faulty, and no party has full omission faults,
- parties $R_1, \ldots, R_i$ have the input 0 and all other parties have the input 1, and
- parties $R_1, \ldots, R_r$ receive no messages and parties $S_1, \ldots, S_s$ receive all messages from parties $R_1, \ldots, R_r$ but no messages from each other.

For $i = 0$, this immediately holds because all parties have input 1 and thus the send faulty parties must output 1 due to validity. We assume this is the case for $i < r$ and will show that this holds for $i + 1$ as well. We will do so by constructing 5 executions:

1. $R_1, \ldots, R_i$ have input 0 and all other parties have input 1. $R_1, \ldots R_r$ are receive faulty and receive no messages, $S_1, \ldots, S_s$ are send faulty and only receive messages sent by $R_1, \ldots, R_r$.
2. $R_1, \ldots, R_i$ have input 0 and all other parties have input 1. $R_1, \ldots R_r$ are receive faulty and $S_1, \ldots, S_s$ are send faulty. R_{i+1} also has send omissions, i.e. it has full omission faults. $R_1, \ldots, R_r$ hear no messages. R_{i+1}'s messages are not delivered to S_1. $S_1, \ldots, S_s$ hear all other messages from $R_1, \ldots, R_r$ and hear no message sent by each other.
3. $R_1, \ldots, R_i$ have input 0 and all other parties have input 1. $R_1, \ldots R_r$ are receive faulty and parties $S_1, \ldots, S_s$ are send faulty. R_{i+1} also has send omissions, i.e. it has full omission faults. $R_1, \ldots, R_r$ hear no messages. R_i's messages are not delivered to $S_1, \ldots, S_s$. $S_1, \ldots, S_s$ hear all other messages from $R_1, \ldots, R_r$ and hear no message sent by each other.
4. $R_1, \ldots, R_{i+1}$ have input 0 and all other parties have input 1. $R_1, \ldots R_r$ are receive faulty and parties $S_1, \ldots, S_s$ are send faulty. R_i also has send omissions, i.e. it has full omission faults. $R_1, \ldots, R_r$ hear no messages. R_i's messages are not delivered to $S_1, \ldots, S_s$. $S_1, \ldots, S_s$ hear all other messages from $R_1, \ldots, R_r$ and hear no message sent by each other.
5. $R_1, \ldots, R_{i+1}$ have input 0 and all other parties have input 1. $R_1, \ldots R_r$ are receive faulty and receive no messages, $S_1, \ldots, S_s$ are send faulty and only receive messages sent by $R_1, \ldots, R_r$.

The first execution is the one described in the induction hypothesis, and thus $S_1, \ldots, S_s$ output 1. $S_2, \ldots, S_n$'s views in executions 1 and 2 are indistinguishable and thus they output 1 in execution 2 as well[5]. From the consistency of the protocol, S_1 outputs 1 as well. S_1 has an identical view in executions 2 and 3, so it outputs 1 in execution 3. From the consistency of the protocol, $S_2, \ldots, S_s$ outputs 1 as well. $S_2, \ldots, S_s$'s views in executions 3 and 4 are indistinguishable, so they output 1 in execution 4. From the consistency of the protocol, S_1 outputs

[5] The assumption that $s > 2$ is used here to guarantee that S_2 exists.

1 as well. Finally, S_1's has identical views in executions 4 and 5 and thus it outputs 1 in execution 5, and so do $S_2, \ldots, S_s$. Note that execution 5 is one in which $R_1, \ldots, R_r$ are receive faulty, $S_1, \ldots, S_s$ are send faulty. In addition, parties $R_1, \ldots, R_{i+1}$ have the input 0 and the rest of the parties have the input 1. In other words, we proved the claim for $i + 1$.

Applying this claim to $i = r$, we find that if $R_1, \ldots, R_r$ have the input 0 and $S_1, \ldots, S_s$ have the input 1, and the only message delivered are those from $R_1, \ldots, R_r$ to $S_1, \ldots, S_s$, all parties output 1. We now construct three final executions to show that all parties output 1 even when they all have the input 0, which would contradict the validity of the protocol.

1. $R_1, \ldots, R_r$ have input 0 and $S_1, \ldots, S_s$ have input 1. $R_1, \ldots R_r$ are receive faulty and receive no messages, $S_1, \ldots, S_s$ are send faulty and only receive messages sent by $R_1, \ldots, R_r$.
2. $R_1, \ldots, R_r, S_1$ have input 0 and $S_2, \ldots, S_s$ have input 1. $R_1, \ldots R_r$ are receive faulty and receive no messages, $S_1, \ldots, S_s$ are send faulty and only receive messages sent by $R_1, \ldots, R_r$.
3. All parties have the input 0. $R_1, \ldots R_r$ are receive faulty and receive no messages, $S_1, \ldots, S_s$ are send faulty and only receive messages sent by $R_1, \ldots, R_r$.

Execution 1 is the execution described in the induction above with $i = r$. As shown, all parties output 1. $S_2, \ldots, S_s$ have indistinguishable views in executions 1 and 2 and thus they output 1 in execution 2 as well. From the consistency of the protocol, S_1 does so as well. S_1's views in executions 2 and 3 are indistinguishable, so it outputs 1 in execution 3. From consistency, $S_2, \ldots, S_s$ also output 1. However, all parties have the input 0, so this violates the validity of the protocol, reaching a contradiction.

Note that in the definitions used in this paper, receive faulty parties are allowed to output $\perp$, but send faulty parties must output a correct value. This proof can be adjusted to show that in execution 3 parties $S_1, \ldots, S_s$ must output non-$\perp$ values even if we allow send faulty parties to output $\perp$. In order to show that S_i must output non-$\perp$ values in execution 3 above, we can construct another execution which is identical to execution 3, except S_i is non-faulty, but receives no messages from the rest of the send faulty parties. S_i's view is identical in execution 3 and this new execution, and thus it must output non-$\perp$ values in both since it is non-faulty in one of them. Since this is true for an arbitrary S_i, this is true for all of them.