Theoretical Computer Science 412 (2011) 5602–5630

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Synchronous consensus under hybrid process and link failures☆

Martin Biely*, Ulrich Schmid, Bettina Weiss

Technische Universität Wien, Institut für Technische Informatik, Embedded Computing Systems Group E182/2, Treitlstraße 1–3, A-1040 Vienna, Austria

ARTICLE INFO

Keywords:
Fault-tolerant distributed systems
Byzantine agreement
Hybrid failure models
Link failures
Uniform consensus
Authentication

ABSTRACT

We introduce a comprehensive hybrid failure model for synchronous distributed systems, which extends a conventional hybrid process failure model by adding communication failures: Every process in the system is allowed to commit up to f_ℓ^s send link failures and experience up to f_ℓ^r receive link failures per round here, without being considered faulty; up to some $f_\ell^{sa} \leq f_\ell^s$ and $f_\ell^{ra} \leq f_\ell^r$ among those may even cause erroneous messages rather than just omissions. In a companion paper (Schmid et al. (2009) [14]), devoted to a complete suite of related impossibility results and lower bounds, we proved that this model surpasses all existing link failure modeling approaches in terms of the assumption coverage in a simple probabilistic setting.

In this paper, we show that several well-known synchronous consensus algorithms can be adapted to work under our failure model, provided that the number of processes required for tolerating process failures is increased by small integer multiples of $f_\ell^s, f_\ell^r, f_\ell^{sa}$, f_ℓ^{ra}. This is somewhat surprising, given that consensus in the presence of unrestricted link failures and mobile (moving) process omission failures is impossible. We provide detailed formulas for the required number of processes and rounds, which reveal that the lower bounds established in our companion paper are tight. We also explore the power and limitations of authentication in our setting, and consider uniform consensus algorithms, which guarantee their properties also for benign faulty processes.

1. Overview

Fault-tolerant synchronous consensus is a classic topic in distributed computing, see [1] for an in-depth treatment and [2] for a concise overview. Consensus is the problem of reaching agreement on an output value, based on input values provided locally by the processes of a distributed system. Synchronous processes allow the algorithm to (conceptually[1]) execute in a sequence of lock-step rounds, each of which comprises exchanging messages and taking a single computing step, simultaneously at every process, for processing the received messages and sending out the messages for the next round.

It is well-known that synchronous consensus is solvable under a wide range of different process failure semantics. For example, $n \geq f + 1$ processes are required for achieving consensus among the correct processes, within $f + 1$ rounds, if at most f processes can fail by crash or omission failures [3]. By contrast, $n \geq 3f + 1$ processes are needed in order to tolerate f Byzantine faulty processes [4]. Consensus has also been studied under several hybrid failure models [5–9], which distinguish different classes of process failures. In all these models, communication is assumed to be reliable, however.

☆ This research has been supported by the Austrian START programme Y41-MAT, the FWF project P18264 (SPAWN), and the FIT-IT project 812205 (TRAFT).

* Corresponding author.
E-mail addresses: biely@ecs.tuwien.ac.at (M. Biely), s@ecs.tuwien.ac.at (U. Schmid), bw@ecs.tuwien.ac.at (B. Weiss).

[1] In fact, lock-step rounds can be simulated in any system with bounded computing step times and message transmission delays.

0304-3975
doi:10.1016/j.tcs.2010.09.032

When communication failures enter the picture, the situation is quite different: Without (severe) restrictions, synchronous consensus is impossible to achieve [10]. Nevertheless, due to the high reliability of modern processors, communication-related failures like receiver overruns (run out of buffers), unrecognized packets (synchronization errors), and CRC errors (data reception problems) in high-speed wireline and, in particular, all sorts of wireless networks are increasingly dominating process failures. Such *link failures* occur on the communication channel or at the network interface, and are typically transient and mobile, in the sense that they typically hit different messages to/from different processes over time.

Perfect communications abstractions, as provided by retransmission schemes in data-link layer protocols [11–13], are hence usually resorted to. However, apart from increasing the protocol's complexity, such solutions considerably impair an algorithm's real-time properties. This is particularly true for synchronous systems, where the duration of the synchronous rounds must be enlarged appropriately to cover multiple retransmissions: The overall execution time of a synchronous algorithm designed for up to k retransmissions per round is the k-fold of the execution time of the link failure intolerant algorithm — even if there is no link failure in the entire execution at all.

In order to avoid the need of a perfect communications abstraction in synchronous systems, link failures must be handled explicitly somehow. A straightforward approach is to map link failures to process failures: A message hit by a link failure is usually detected by CRC checking and discarded, which can be viewed as an omission process failure. In rare cases, such message alterations might go unnoticed, which can be interpreted as a Byzantine process failure. Unfortunately, however, declaring the sender or receiver of every faulty message as faulty quickly exhausts any reasonable maximum number f of faulty processes; see [14] for a simple probabilistic analysis. Note that declaring the sender or receiver process as faulty would also be overly conservative in terms of the failure semantics, since a communication failure does not usually imply a process failure: After all, the – fault-tolerant – algorithm is executed correctly by such "communication-faulty" processes.

Consequently, link failures should be a category of their own in a more realistic failure model. Unfortunately, however, there is a discouraging impossibility result for deterministic synchronous consensus in the presence of link failures, which goes back to Gray's 1978 paper [15] on atomic commitment in distributed databases:

Theorem 1.1 (*Gray's Impossibility [1, Thm. 5.1]*)**.** *There is no deterministic algorithm that solves the coordinated attack problem in a synchronous 2-process system with lossy links.*

This result was strengthened by Santoro and Widmayer [10,16], who introduced mobile transmission failures: A different set of links can be faulty in every round here. This model also covers the scenario where, in every round, a different process experiences transmission failures on all its outgoing links and hence appears send omissive (termed omissive for short). Unfortunately, even a single mobile omissive process failure was shown to render consensus unsolvable.

Perhaps influenced by these results, almost all deterministic algorithms for consensus developed during the past 20+ years considered process failures only.[2] In fact, only a few algorithms [17–20] have been proposed in the past that can also deal with a small number of link failures. For substantial link failure rates, this is not sufficient.

As a remedy, one could ask whether some of the pivotal assumptions of the above impossibility results can be circumvented — without sacrificing the mobile nature of link failures, of course. The synchronous consensus algorithms developed for the *perception-based hybrid failure model* introduced in this paper[3] show that this is indeed possible. Our model essentially augments a comprehensive hybrid process failure model by several classes of link failures. More specifically, every process in the system is allowed to commit up to f_ℓ^s send link failures and up to f_ℓ^r receive link failures, in a mobile, round-by-round fashion [10,25], without being considered (process-)faulty. Optionally, $0 \le f_\ell^{sa} \le f_\ell^s$ and $0 \le f_\ell^{ra} \le f_\ell^r$ links may produce erroneous messages – rather than just omissions – as well. We coined the term perception-based for our model, since the failure semantics is solely defined in terms of how processes perceive the messages received in a round. It thus shares many features with the Byzantine variant [26] of the Heard-of Model by Charron-Bost and Schiper [27]. As discussed in some detail in [14], it is possible to express our link failure restrictions as simple communication predicates that involve only single rounds, for example.

In the companion paper [14], we provided a comprehensive theoretical study of consensus under our link failure model. Using bivalency and "easy impossibility proof" techniques, we developed a complete suite of related impossibility results and lower bounds for the required number of processes and rounds in presence of link failures. They are primarily based on a generalization of Theorem 1.1, which stresses the importance of unimpaired *bidirectional* communication for solving consensus. Our results also allowed us to precisely characterize what makes a process failure actually Byzantine resp. omissive. Finally, we conducted a detailed analysis of the assumption coverage in a simple probabilistic setting, which revealed that our link failure model is the only one with a coverage that approaches 1 (rather than 0) for large n. In [28], it was shown that a decent coverage can also be established in case of small n, even in case of substantial link failure rates.

The present paper provides a suite of consensus algorithms for our perception-based failure model. More specifically, it will turn out that many existing synchronous consensus algorithms [4,8,29–31] work also, with small modifications, under our failure model, provided that the number of processes required for tolerating process failures is increased by

[2] Just compare Chapter 5 ("Distributed Consensus With Link Failures") in Lynch's book [1] with Chapter 6 ("Distributed Consensus With Process Failures") …

[3] Part of our work has already been presented in [21–23]. A formal verification of some of our proofs can be found in the SRI technical report [24].

$C_r f_\ell^r + C_s f_\ell^s$, for some small integers C_r, C_s that depend on the particular algorithm. Detailed formulas for the required number of processes and rounds will be provided for every algorithm; our results will prove/reconfirm that the lower bounds established in [14] are tight. Our results for authenticated algorithms will also reveal that authentication is beneficial not only for process failures but also for link failures.

The remaining sections of our paper are organized as follows:

- *Section 2:* Brief overview of related work.
- *Section 3:* Definition of our hybrid perception-based failure model.
- *Section 4:* Analysis of the *Hybrid Oral Messages* algorithm OMH (Section 4.1), and its uniform variant OMHU (Section 4.2).
- *Section 5:* Introduction of authentication issues for *Hybrid Written Messages* (Section 5.1) and algorithms OMHA, ZA and ZAr (Section 5.2).
- *Section 6:* Analysis of the polynomial consensus algorithms Phase Queen (Section 6.1), Phase King (Section 6.2), and variants ST1 and ST2 of Srikanth & Toueg's algorithm (Section 6.3).
- *Section 7:* Summary and discussion of our accomplishments.

Some conclusions in Section 8 eventually complete our paper.

2. Related work

There are a number of *hybrid failure models* in the literature [5–9,20,32–38], which differ primarily in the classes of process failures considered: Early models like [6] distinguish only manifest and Byzantine failures, whereas fully-fledged models like the ones of [9] and [38] provide a reasonably complete classification of all conceivable failure modes. Hybrid failure models are interesting, in particular for small-sized systems, since they exploit the fact that less severe failures can usually be handled with fewer processes than more severe ones. However, none of the existing hybrid failure models that are applicable to consensus algorithms also covers link failures explicitly.

In fact, there are only a few failure models for synchronous systems in the literature that deal with link failures at all. We provided a very detailed discussion of the related work in [14]. Hence, we will only summarize some related approaches here.

One obvious approach to model link failures is to simply map link failures to process failures, as in [30,3]. However, for significant link failure rates, this quickly leads to the exhaustion of the maximum number f of tolerable process failures [14].

Another class of models [18–20] considers a small number of link failures explicitly: Those papers assume that at most $\mathcal{O}(n)$ links may be faulty system-wide during the entire execution of a consensus algorithm. Obviously, such models can only be applied in case of very small link failure rates. In their landmark paper [10,16], Santoro and Widmayer showed that consensus cannot be solved in the presence of $n-1$ omission resp. $\lfloor n/2 \rfloor$ Byzantine link failures per round, in particular, if those link failures hit the same sender process. As a consequence, consensus cannot be solved in the presence of just a single *mobile* omission or Byzantine faulty process. On the other hand, if the number of moving link failures is restricted to less than $n-1$ omission resp. $\lfloor n/2 \rfloor$ Byzantine link failures per round, consensus can be solved in a constant number of rounds [39,40].

The consensus algorithm proposed by Reischuk in [17] is the first algorithm we are aware of which was proved to tolerate more than $O(n)$ link failures per round, in addition to f Byzantine process failures: In a system with $n \geq 20f + 1$ processes, at most f of which may be Byzantine faulty in a round, the algorithm tolerates $l = n/20$ link failures at every process.

An even better link failure resilience (actually, comparable to our results) is provided by the suite of synchronous atomic broadcast protocols introduced in [41]. Although reliable/atomic broadcasting is usually investigated in a more communications-oriented context [42], it obviously allows one to solve Byzantine agreement as well. The three algorithms of [41] tolerate an arbitrary number of processes with omission, timing, or Byzantine failures (if authentication is available) and work on general communication graphs subject to link failures. Instead of making the number of link failures explicit, however, it is just assumed that any two processes in the system are always connected via a path of non-faulty links. Since our failure model respects the requirements of [41] (see [14, Thm. 9]), we can compare our results in Section 5.2.

A similar connectivity-related requirement is used within the "majority groups" in the Paxos-related paper [43]. A theoretical study of connectivity requirements for consensus in case of bidirectional links can be found in [44].

Unlike in the deterministic setting, link failures are easily tolerated by *randomized* consensus algorithms like the one of [45]. Such algorithms circumvent the impossibility result of Theorem 1.1 by adding non-determinism (coin tossing) to the computations. Still, there is a lower bound $1/(R+1)$ for the probability of disagreement after R rounds [1, Thm. 5.5]. It is interesting to note, however, that our approach of modeling link failures also allows one to circumvent this lower bound: For a suitably adapted variant of some well-known randomized algorithm, we established a probability of disagreement of only $(1/2)^R$ [46].

We finally note that it is relatively easy to handle omissive link failures in asynchronous systems with "fair (lossy) links": If sending an infinite number of messages over a link causes an infinite number of messages to be received, a perfect link can be simulated by suitable retransmission-based protocols [11–13]. Such time redundancy techniques cannot reasonably be used in synchronous systems, however: Since the duration of the rounds is fixed, it must be chosen so as to accommodate some specified number of retransmissions. Clearly, this choice is overly conservative in the typically vast majority of rounds

where no message loss occurs. In sharp contrast, our approach uses resource redundancy only (additional processes) and therefore allows one to chose a round duration that does not incorporate retransmission times. A detailed survey of link failures in partially synchronous and asynchronous systems may be found in [47].

3. System model

We consider a system of n distributed *processes*, each identified by a unique id $p \in \Pi = \{1, \ldots, n\}$. The processes are fully connected by a point-to-point network made up of unidirectional *links*. Every pair of processes p and $q \neq p$ is hence connected by a pair of links (p, q), from sender process p to receiver process q, and (q, p), from sender process q to receiver process p, which are considered independent of each other. To simplify our presentation, we also assume that there is a link (p, p) from every process $p \in \Pi$ to itself. Our system hence contains n^2 unidirectional links. Links may reorder messages, that is, are not assumed to be FIFO.

Note that we will define our system model for a single process per processor for simplicity. If a distributed algorithm consists of several processes per processor, the process failure model must be applied at the processor-level, in conjunction with the assumption that all processes executed by a single processor may commit failures at most as severe as the failure mode of the processor allows. By contrast, the link failure budgets are always on a per process basis.

3.1. Computing model

For our computing model, we employ the standard lock-step round model as used in [10,14]. Every process p is modeled as a deterministic state machine – acting on some local state $state_p \in State_p$ – that can send and receive messages from some (possibly infinite) alphabet $\mathcal{M}$. The initial state of process p is drawn from a set of initial states $Init_p \subseteq State_p$. All processes execute, in perfect synchrony, a sequence of atomic computing steps forming a sequence of lock-step rounds $k \in K = \{1, 2, \ldots\}$: In round k, process p performs the following steps:

1. Initialize the *received messages vector* Rm_p to $\forall q \in \Pi : Rm_p[q] := \emptyset$, where $\emptyset$ represents "no message", and sends at most one message $msg_p^k = Msg_p(state_p, k)$ to every process $q \in \Pi$ (including itself); $Msg_p : State_p \times K \to \mathcal{M} \cup \{\emptyset\}$ denotes the *message sending function* of the algorithm executed by p.
2. Wait for some time while receiving messages into Rm_p. This time must be sufficiently long to allow delivery of all correct round k messages. We assume that no messages arrive after this waiting period is over; practically, if late messages arrive, they are discarded.
3. Perform a state transition from $state_p$ to $state'_p = Trans_p(state_p, rm_p, k)$, where $Trans_p : State_p \times Rm_p \times K \to State_p$ denotes the *state transition function* of the algorithm executed by p.

Note that the round number k can be viewed as global time in this model, and is typically part of $state_p$.

The distributed algorithm executed by the processes is hence specified by the pairs of message sending functions and message transition functions $\{(Msg_p, Trans_p) | p \in \Pi\}$. The system-wide $n \times n$ *received messages matrix* $\mathcal{R}^k$ for round k is the column vector $(rm_1^k, \ldots, rm_n^k)^T$ of the content of all processes' received messages vectors Rm_p in round k, i.e.,

$$\mathcal{R}^k = \begin{pmatrix} rm_1^k[1] & rm_1^k[2] & \ldots & rm_1^k[n] \\ rm_2^k[1] & rm_2^k[2] & \ldots & rm_2^k[n] \\ \vdots & \vdots & \vdots & \vdots \\ rm_n^k[1] & rm_n^k[2] & \ldots & rm_n^k[n] \end{pmatrix}, \tag{1}$$

with entry $rm_p^k[q]$ denoting the message that process p received from process q via its incoming link in round k, or $\emptyset$ if no such message (or a detectably bad one) was received.

3.2. Failure model

Consider the system-wide $n \times n$ *messages matrix* $\mathcal{M}^k$ for round k, which consists of n rows containing the messages

$$\mathcal{M}^k = \begin{pmatrix} msg_1^k & msg_2^k & \ldots & msg_n^k \\ msg_1^k & msg_2^k & \ldots & msg_n^k \\ \vdots & \vdots & \vdots & \vdots \\ msg_1^k & msg_2^k & \ldots & msg_n^k \end{pmatrix}, \tag{2}$$

with $msg_p^k = Msg_p(state_p, k)$ denoting the message process p should – according to the algorithm – send to every[4] process q via its outgoing link in round k.

[4] For simplicity, we assume that every process sends the same message to all processes in a round — generalizing the model to the case where different messages can be sent to different receivers in a round is quite straightforward.

Consider further the $n \times n$ *sent messages matrix* $\mathcal{S}^k$ for round k, which consists of n rows containing the messages actually sent by every process in round k, i.e.,

$$\mathcal{S}^k = \begin{pmatrix} sm_1^k[1] & sm_2^k[1] & \dots & sm_n^k[1] \\ sm_1^k[2] & sm_2^k[2] & \dots & sm_n^k[2] \\ \vdots & \vdots & \vdots & \vdots \\ sm_1^k[n] & sm_2^k[n] & \dots & sm_n^k[n] \end{pmatrix}, \tag{3}$$

with $sm_p^k[q]$ denoting the message process p sends to process q via its outgoing link in round k, or $\emptyset$ if no message is sent.

Clearly, in case of no failures, $\mathcal{M}^k = \mathcal{S}^k = \mathcal{R}^k$ for all rounds k. Our failure model distinguishes several classes of (static) process failures as well as (dynamic, on a per-round basis) link failures. These failures are characterized by how they affect the matrices $\mathcal{S}^k$ and $\mathcal{R}^k$.

We distinguish the following classes of process behavior:

Correct: A process p that faithfully follows its algorithm for the whole execution, such that for every round k, it holds that $sm_p^k[q] = msg_p^k \ \forall q \in \Pi$.

Omission: A process p which at least once fails to send its value to a subset of its receivers, that is, for every round k, $sm_p^k[q] \in \{msg_p^k, \emptyset\} \ \forall q \in \Pi$, and $\exists k_0$ and $q \in \Pi$ such that $msg_p^{k_0} \neq sm_p^{k_0}[q] = \emptyset$.

Manifest: A process p that at least once fails to send a message to any receiver or sends a detectably bad value (which can be recognized as erroneous by some local checks) to all receivers, that is, for every round k and $\forall q \in \Pi$, we have either $sm_p^k[q] = msg_p^k$ or else $sm_p^k[q] = \emptyset$ or $sm_p^k[q]$ detectably bad, and $\exists k_0$ where $sm_p^{k_0}[q] \neq msg_p^{k_0}$ for all $q \in \Pi$.

Symmetric: A process p, which at least once sends the same (generally not detectably) bad value to all receivers, that is, for every round k, $sm_p^k[q] = m_p^k \ \forall q \in \Pi$ for some $m_p^k \in \mathcal{M}$, and $\exists k_0$ where $msg_p^{k_0} \neq m_p^{k_0} \neq \emptyset$.

Arbitrary: A process p which has no restrictions on the messages it sends.

In our system model, during any execution, there may be at most f_m manifest, f_o omission, f_s symmetric, and f_a arbitrary faulty processes. Let

$$f_{p\_all} = f_a + f_s + f_o + f_m \tag{4}$$

abbreviate the total number of process failures. Once a process exhibits a failure, it is considered faulty for the remaining execution. Processes exhibiting more than one type of failure must be counted according to their most severe behavior, with arbitrary $\supseteq$ {symmetric, omissive} $\supseteq$ manifest. Thus, a symmetric faulty processes with omissive behavior must be counted as arbitrary faulty.

In the message matrices introduced above, process failures are such that, during all rounds k, at most the fixed columns corresponding to the same f_m manifest, f_o omission, f_s symmetric, and f_a arbitrary faulty processes in the sent messages matrix $\mathcal{S}^k$ may deviate from the columns in the messages matrix $\mathcal{M}^k$. Note that this modeling is based on a sender-centric view of process failures. In particular, we do not incorporate process receive omission failures [3]. Such failures can be incorporated via link failures, however.

More specifically, apart from process failures, our model also incorporates communication failures: A *link failure* hitting the directed link (p, q) results in $rm_q^k[p] \neq sm_p^k[q]$. As in [10,14], we distinguish the following types of link failures of (p, q) in a single round k:

Correct link: $rm_q^k[p] = sm_p^k[q]$

Lossy link: $\emptyset = rm_q^k[p] \neq sm_p^k[q]$

Erroneous link (corruption): $\emptyset \neq rm_q^k[p] \neq sm_p^k[q] \neq \emptyset$

Erroneous link (spurious): $\emptyset \neq rm_q^k[p] \neq sm_p^k = \emptyset$

For some round k, a lossy link is called *omission faulty*, an erroneous link (corrupted or spurious) is termed *arbitrary faulty*. A link producing either type of failure is termed *faulty*.

Our link failure model is such that, system-wide, up to $c \cdot n^2$ links, for some $c < 1$, may be faulty in any round. We cannot allow *any* set of $c \cdot n^2$ links to be hit by link failures, however, but require some restriction on the allowed link failure patterns: Let $\mathcal{F}^k$ be the $n \times n$ *link failure pattern matrix* with entries

$$f_q^k[p] = \begin{cases} ok & \text{if } rm_q^k[p] = sm_p^k[q], \\ om & \text{if } \emptyset = rm_q^k[p] \neq sm_p^k[q], \\ arb & \text{otherwise,} \end{cases}$$

which is just the difference of $\mathcal{R}^k$ and $\mathcal{S}^k$ interpreted as *ok*, *om*, or *arb* on a per entry basis, depending on the corresponding link behavior. The *feasible* pattern of system-wide link failures must be such that for every process p and every round k,

(A1^r) p's row $(f_p[1], \ldots, f_p[n])$ in $\mathcal{F}^k$ contains at most f_ℓ^r entries $\neq ok$, with at most $f_\ell^{ra} \leq f_\ell^r$ of those equal to *arb*. Since row p corresponds to p acting as a receiver process, we say that p may perceive at most f_ℓ^r *receive link failures* (on its incoming links), with up to f_ℓ^{ra} arbitrary ones among those.

(A1^s) p's column $(f_1[p], \ldots, f_n[p])^T$ in $\mathcal{F}^k$ contains at most f_ℓ^s entries $\neq ok$, with at most $f_\ell^{sa} \leq f_\ell^s$ of those equal to *arb*. Since column p corresponds to p acting as a sender process, we say that p may experience at most f_ℓ^s *send link failures* (on its outgoing links), with up to f_ℓ^{sa} arbitrary ones among those.

Note that *every* process in the system may experience up to f_ℓ^s send link failures (f_ℓ^{sa} of them arbitrary), and up to f_ℓ^r receive link failures (f_ℓ^{ra} of them arbitrary) in every round. Properties (A1^s) and (A1^r) are in fact considered independent of each other, with the obvious restriction that $f_\ell^r = 0 \Leftrightarrow f_\ell^s = 0$. Clearly, the particular links hit by a link failure may be different in different rounds.

Viewed from a receiver's perspective, it is sometimes too restrictive to assume that only non-faulty processes get their inputs and faithfully execute the particular algorithm. We will hence introduce the notion of an *obedient* process p, which gets its inputs and faithfully executes its algorithm like a non-faulty process, but may fail in specific ways to communicate its messages to the outside world, i.e., allows $sm_p^k[q] \neq msg_p^k$. Somewhat arbitrarily, we will consider all processes except arbitrary and symmetric faulty ones as obedient: Since these processes are (also) allowed to convey the correct value to their receivers, they must know this value internally. Although we need not care how this is actually accomplished, it is nevertheless true that the only way to ensure this in practice is to assume that benign faulty processes are obedient.

Definition 3.1. A process is

- *benign faulty* if it is either manifest or omission faulty,
- *obedient* if it is either correct or benign faulty,
- *consistent* if it is either correct, manifest or symmetric faulty.

Note carefully that a generalization from non-faulty to obedient processes does not always work, cp. algorithms OMH vs. its uniform variant OMHU in Section 4 for an example.

Remarks.
1. Our system model considers process and link failures to be independent. Therefore, even a manifest or omission faulty process's broadcast could generate erroneous messages at f_ℓ^{sa} receivers, for example.
2. Since our failure model allows link failures to be transient or permanent, it can also be applied in networks which are not fully connected, cp. [48,44,20,14].
3. Specified in a round-by-round fashion, our failure model does not contain a direct equivalent for standard *crash failures*, where a process can die once and forever even during its broadcast. Standard crash failures must hence be counted as omission failures in our model: Considering a single round only, crash failures are in fact equivalent to omission failures. Viewed throughout an execution, however, both manifest and omission failures are more severe than crashes in that (manifest/omission) faulty processes do not have to remain silent from some round onwards but may resume correct operation later on. Note that the latter behavior is not equivalent to the crash-recovery model of [13], since our processes may not lose state but must continuously follow the algorithm.
4. Manifest failures are less severe than omission failures since they are perceived symmetrically at all receivers. They could be produced by a faulty component that, e.g., lost round synchronization and hence sends correct messages in the wrong round. Note that our manifest failures differ from the system wide detectable ones of [49] and the "benign failures" of [38,9] by also including symmetric omissions, where no receiver gets a non-$\emptyset$ value. We provide only *strong* manifest failures, however, where all receivers, including the sender itself, get the same value. Some related work like [24] considers *weak* manifest failures, where the sender is allowed to deliver the correct value instead of $\emptyset$, thereby causing some asymmetry in the reception. Of course, weak manifest failures can always be incorporated as omission failures in our model.
5. Arbitrary faulty processes need not adhere to the particular algorithm. Unlike a symmetric or benign faulty process, an arbitrary faulty process could even send multiple messages in a single round; a receiver may deliver any of those or $\emptyset$ in this case.

4. Hybrid oral messages algorithms

In this section, we will show that the *Hybrid Oral Messages* algorithm OMH derived from [4] in [8], as well as its uniform variant OMHU, solves Byzantine agreement under the system model of Section 3. In this and the following sections, we will assume that f_a, f_s, f_o, and f_m specify the maximum number of faulty processes of the appropriate type during the entire execution. Moreover, we assume that link failures do not hit the message sent by a process to itself. Still, strong manifest faulty processes deliver $\emptyset$ also to themselves, as may omission faulty processes.

In Byzantine Agreement, the value $v \in V$ (chosen from a finite set of values V) of a dedicated *transmitter* is to be disseminated to the remaining $n-1$ *receivers*; the transmitter already knows its value. Eventually, every non-faulty process p – including the transmitter – must *deliver* a value v_p ascribed to the transmitter that satisfies the following properties:

(B1) *Agreement*: If processes p and q are both non-faulty, then both deliver the same $v_p = v_q$.

(B2) *Validity*: If process p is non-faulty, the value v_p delivered by p is
- v, if the transmitter is non-faulty,
- $\emptyset$, if the transmitter is manifest faulty,
- v or $\emptyset$, if the transmitter is omission faulty,
- the value actually sent, if the transmitter is symmetric faulty,
- unspecified, if the transmitter is arbitrary faulty.

A fully-fledged n-process consensus algorithm is obtained by using a separate instance of a Byzantine agreement algorithm (with $n-1$ receivers) for disseminating any process's local value, and using a suitable choice function (majority) for the consensus result. Conversely, such a Byzantine agreement algorithm is obtained by just sending the transmitter's value directly to $n-1$ receivers, and feeding the received values into a $n-1$-process consensus algorithm. As a consequence, the round complexity lower bounds $R(Byz, n)$ resp. $R(Cons, n)$ for n-process Byzantine agreement resp. consensus in any model are obviously related by $R(Cons, n) \leq R(Byz, n) \leq 1 + R(Cons, n-1)$ (and are known to be the same for standard process failures, cp. [4]). A similar relation $F(Cons, n) \geq F(Byz, n) \geq F(Cons, n-1)$ holds for the resilience. Bear this in mind when we compare our Byzantine agreement algorithms to the consensus lower bounds established in [14].

Properties (B1) and (B2) require agreement and validity to hold only for non-faulty processes. Since there are applications where the behavior of benign faulty processes also matters, cp. [22, Sec. 4], one could ask whether it is possible to extend those properties to obedient processes. We will hence also consider the following *uniform* properties [42]:

(UB1) *Uniform Agreement*: If processes p and q are both obedient, then both deliver the same $v_p = v_q$.
(UB2) *Uniform Validity*: If process p is obedient, the value v_p delivered by p is
- v, if the transmitter is non-faulty,
- $\emptyset$, if the transmitter is manifest faulty,
- v or $\emptyset$, if the transmitter is omission faulty,
- the value actually sent, if the transmitter is symmetric faulty,
- unspecified, if the transmitter is arbitrary faulty.

Note that this definition does not extend (B1) and (B2) to symmetric and arbitrary faulty processes, since these classes of failures do not constrain the internal behavior of faulty processes and can hence be arbitrary. (This issue is also discussed in [50].) Nevertheless, if symmetric and arbitrary faulty processes would faithfully execute the algorithm, i.e., would behave erroneous only when emitting messages, then our uniform Byzantine agreement algorithm OMHU would achieve the uniform agreement property for them as well.

4.1. Byzantine agreement with the algorithm OMH

OMH is a "Byzantine generals" algorithm, where the value v of a dedicated transmitter is to be disseminated to the remaining $n-1$ receivers such that the Agreement (B1) and Validity (B2) properties hold. The algorithm OMH as specified in Definition 4.1 below uses two primitives:

- The *wrapper function* $R(v)$ encodes a statement "I am reporting v" as a unique value. Reporting is undone by means of the inverse function $R^{-1}(v)$, which must guarantee $R^{-1}(R(v)) = v$. The purpose of the wrapper function is to make erroneous values ($\emptyset$) originating in different rounds distinguishable, so that they cannot accumulate over multiple rounds and eventually overwhelm legitimate values $v \in V$. Therefore, strictly speaking, only $\emptyset$, $R(\emptyset)$, $R(R(\emptyset))$, $R(R(R(\emptyset)))$, ... must actually be distinguishable and different from any $v \in V$ here; for each legitimate value v, we can allow $R(v) = R^{-1}(v) = v$. Consider Remark 3 below for a simple way to define a suitable wrapper function $R(.)$.
- The *hybrid-majority* of a set $\mathcal{V}$ of values provides the majority of all non-$\emptyset$ values in $\mathcal{V}$. If no majority exists, the default value $R(\emptyset)$ is returned. Note that this particular default value is required for securing validity in the presence of an omission faulty transmitter, see the proof of Lemma 4.3.

Consult [8] for a detailed discussion of the above primitives and the operation of OMH in general.

Definition 4.1 (*Algorithm OMH [8]*)**.** The Hybrid Oral Message algorithm OMH is defined recursively as follows (the parameter m gives the number of rounds the algorithm is allowed to run, which is in turn related to the maximum number of failures; see Theorem 4.4 for details):
OMH(0):

1. The transmitter t sends its value v to every receiver and delivers $v_t = v$.
2. Every receiver p delivers the value v_p received from the transmitter, or the value $\emptyset$ if a missing or manifestly erroneous value was received.

OMH(m), $m > 0$**:**

1. The transmitter t sends its value v to every receiver and delivers $v_t = v$.
2. For every p, let w_p be the value receiver p receives from the transmitter, or $\emptyset$ if no value or a manifestly bad value was received.
   Every receiver p acts as the transmitter in algorithm OMH($m-1$) to communicate the value $R(w_p)$ to all receivers.

3. For every p and $q \neq p$, let w_p^q be the value receiver p delivers as the result of OMH$(m-1)$ initiated by receiver q in step 2 above. Every receiver p calculates the hybrid-majority value of all values w_p^q and its own value $w_p^p = R(w_p)$, and applies R^{-1} to that value. This result is delivered as v_p.

Remarks. 1. There are $n-1$ receivers in the first instance OMH(m) of the algorithm; the transmitter does not participate in any way in further recursive instances. Our n-process, $m+1$-round Byzantine agreement algorithm OMH(m) can hence be viewed as an initial broadcast of the transmitter's value to all receivers combined with an $(n-1)$-process, m-round consensus algorithm.
2. In our failure model, the transmitter's delivery $v_t = v$ can be considered as the result of sending its value to itself. Hence, there is no need to treat the case $q \neq p$ and $q = p$ differently in step 3 in OMH(1). We will use this fact in order to immediately apply Lemma 4.2 below in OMH's analysis.
3. During the recursive execution of OMH, multiple (in fact, exponentially many) instances of OMH are concurrently active in each round. In OMH's description, we did not explicitly address the question of how received messages are assigned to the unique recursive instance they belong to. Practically, the unique *id* of the transmitter process is appended to each message for this purpose, which produces a string of ids $p_1 p_2 \dots p_k$ that uniquely determines the particular recursive instance (and, by its complement, the set of participating receivers). Note carefully that this string must be reconstructed upon reception of a $\emptyset$-value and included in the $R(\emptyset)$-message prior to submitting it to further recursive instances, which also results in a simple implementation of the wrapper function $R(.)$.
4. By expanding OMH's recursive description, it is easy to see that the execution of OMH(m) consists of $m+1$ rounds where messages are wrapped and forwarded only, followed by the backwards-recursive computation (hybrid-majority + unwrapping) in step 3 of all instances, cp. the EIG-tree representation in [51, p. 105].

By adopting and extending the analysis of [8] to our perception-based failure model, we will now show that OMH satisfies (B1) and (B2).

Before we start our analysis, we present a generally applicable technical Lemma 4.2, which shows that a common decision value is reached among all obedient processes if sufficiently many processes have sent the same value. More specifically, let us assume that all processes $p \in \mathcal{S}$ of some set of processes $\mathcal{S}$ *convey* their local values w_p to an arbitrary obedient receiver q (possibly $q \in \mathcal{S}$) in some round of the execution of a certain (multi-round) algorithm. Conveying means that all $p \in \mathcal{S}$ send $R(w_p)$ to q, which computes R^{-1} of the hybrid-majority among the set of received values as its result v_q. Conveying occurs in steps 2 and 3 of OMH(1), for example. Note carefully that also a manifest faulty sender process p_m could send $R(w_{p_m})$ in the conveying round, since manifest failures need not occur in every round.

Lemma 4.2. *For any $f, f_a, f_o, f_s, f_m, f_\ell^s, f_\ell^r, f_\ell^{ra} \geq 0$, let some set of processes $\mathcal{S}$ be given that convey their values to an obedient receiver q in some round of the execution of a distributed algorithm.*

(a) *If $|\mathcal{S}| > 2f + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + 2f_o + f_m$ and all but at most f of the non-faulty or manifest faulty processes $p \in \mathcal{S}$*
  (1) *convey the same value $w_p = v$, then q computes $v_q = v$,*
  (2) *convey values taken from a set $\mathcal{W}$ with $|\mathcal{W}| \geq 2$, then $v_q \in \mathcal{W} \cup \{\emptyset\}$.*

(b) *If $|\mathcal{S}| > 2f + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m$ and all but at most f of the obedient processes $p \in \mathcal{S}$*
  (1) *convey the same value $w_p = v$, then q computes $v_q = v$,*
  (2) *convey values taken from a set $\mathcal{W}$ with $|\mathcal{W}| \geq 2$, then $v_q \in \mathcal{W} \cup \{\emptyset\}$.*

Proof. Let $n' = |\mathcal{S}|$, $\mathcal{S}_f \subseteq \mathcal{S}$ be the set of the at most f exempted processes, and f_o' resp. f_m' be the actual number[5] of omission resp. manifest failures in $\mathcal{S}$ that affect our receiver q in the conveying round by a $\emptyset$-value (or, in case of an additional arbitrary link failure, even by a faulty value).

In addition, let $f_{\ell n}^{r\,\prime} \leq f_\ell^r$ be the number of actual receive link failures at q that hit non-faulty senders $\in \mathcal{S} \setminus \mathcal{S}_f$, $f_{\ell m}^{ra\,\prime}$ resp. $f_{\ell o}^{ra\prime}$ be the actual number of arbitrary receive link failures at q hitting manifest resp. omission faulty senders, and $f_\ell^{ro\prime\prime}$ be the number of actual receive link omission failures at q hitting all but actually manifest or omission faulty senders. Clearly, we have $f_\ell^{ra} + f_\ell^{ro\prime\prime} \geq f_{\ell n}^{r\,\prime} + f_{\ell o}^{ra\prime} + f_{\ell m}^{ra\,\prime}$, since we can split $f_{\ell n}^{r\,\prime}$ into its arbitrary and omissive components $f_{\ell n}^{r\,\prime} = f_{\ell n}^{ra\prime} + f_{\ell n}^{ro\prime}$ and $f_\ell^{ra} \geq f_{\ell n}^{ra\prime} + f_{\ell o}^{ra\prime} + f_{\ell m}^{ra\,\prime}$ and $f_\ell^{ro\prime\prime} \geq f_{\ell n}^{ro\prime}$.

We start with proving item (a.1) of the statements of our lemma. Since all non-faulty and manifest faulty processes $\in \mathcal{S} \setminus \mathcal{S}_f$ convey the same value $R(v)$, our obedient receiver q will obtain at least

$$n_q' = n' - f - f_a - f_s - f_o - f_m' - f_{\ell n}^{r\,\prime}$$

identical values $R(v)$. It will also get at most

$$n_q'' = n' - f_o' - f_m' - f_\ell^{ro\prime\prime} + f_{\ell o}^{ra\prime} + f_{\ell m}^{ra\,\prime}$$

[5] In this and subsequent proofs, we often need the actual number of failures of a given type. It will be denoted by priming the bound on the maximum number of failures: $0 \leq f_o' \leq f_o$ etc.

non-$\emptyset$ values. Since $n' > 2f + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + 2f_o + f_m$ here, we have

$$\begin{aligned} 2n'_q - n''_q &= n' - 2f - 2(f_a + f_s) - 2f_o - f'_m - 2f_{\ell n}^{r\,\prime} + f'_o + f_\ell^{ro\prime\prime} - f_{\ell o}^{ra\prime} - f_{\ell m}^{ra\,\prime} \\ &> (f_m - f'_m) + f'_o + (f_\ell^r - f_{\ell n}^{r\,\prime}) + (f_\ell^{ra} + f_\ell^{ro\prime\prime} - f_{\ell n}^{r\,\prime} - f_{\ell o}^{ra\prime} - f_{\ell m}^{ra\,\prime}) \\ &\geq 0. \end{aligned} \tag{5}$$

Therefore, $R(v)$ will indeed win the hybrid majority at process q.

In case (b.1), where all obedient processes $\in \mathcal{S}\backslash\mathcal{S}_f$ convey the same value $R(v)$, q will obtain at least

$$\overline{n}'_q = n' - f - f_a - f_s - f'_o - f'_m - f_{\ell n}^{r\,\prime}$$

identical values $R(v)$ and will again get at most

$$\overline{n}''_q = n' - f'_o - f'_m - f_\ell^{ro\prime\prime} + f_{\ell o}^{ra\prime} + f_{\ell m}^{ra\,\prime}$$

non-$\emptyset$ values. Since $n' > 2f + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m$ here, we get

$$\begin{aligned} 2\overline{n}'_q - \overline{n}''_q &= n' - 2f - 2f_a - 2f_s - f'_o - f'_m - 2f_{\ell n}^{r\,\prime} + f_\ell^{ro\prime\prime} - f_{\ell o}^{ra\prime} - f_{\ell m}^{ra\,\prime} \\ &> (f_o - f'_o) + (f_m - f'_m) + (f_\ell^r - f_{\ell n}^{r\,\prime}) + (f_\ell^{ra} + f_\ell^{ro\prime\prime} - f_{\ell n}^{r\,\prime} - f_{\ell o}^{ra\prime} - f_{\ell m}^{ra\,\prime}) \\ &\geq 0, \end{aligned} \tag{6}$$

by the same reasoning as above, so $R(v)$ will again win the hybrid-majority at q.

As far as (a.2) [resp. (b.2)] of the statements of Lemma 4.2 is concerned, if the appropriate processes $\in \mathcal{S}\backslash\mathcal{S}_f$ did not all send the same value but rather values from a set $\mathcal{W}$, then we cannot always guarantee that receiver q obtains a majority for any of these values (although this could happen). In this case, however, since (5) [resp. (6)] holds if we conceptually replace each value from $\mathcal{W}$ by a single legitimate value X, it turns out that there cannot be a majority for any value $\notin \mathcal{W}$. Therefore, only the default value $R(\emptyset)$ can be returned by the hybrid-majority primitive here, which leads to the alternative return value $v_q = \emptyset$. □

Now we are ready for proving that OMH satisfies the validity property (B2). Note that Lemma 4.3 below is void in case of an arbitrary faulty transmitter, since (B2) does not say anything about the value a receiver ascribes to the transmitter in this case.

Lemma 4.3 (*Validity OMH*). *For any $m \geq \min\{1, f_\ell^s\}$ and any $f_a, f_s, f_o, f_m, f_\ell^s, f_\ell^r, f_\ell^{ra} \geq 0$, the algorithm OMH(m) satisfies the validity property* (B2) *if there are strictly more than $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m$ participating processes.*

Proof. Proving (B2) amounts to showing that any obedient[6] process p, including the transmitter, delivers

(1) $v_p = v = v$, if the transmitter is non-faulty,
(2) $v_p = v = \emptyset$, if the transmitter is manifest faulty,
(3) $v_p = v$ or $v_p = \emptyset$, if the transmitter is omission faulty,
(4) $v_p = v$, if the transmitter is symmetric faulty.

The actual proof is by induction on m.

If there are no link failures, i.e., if $f_\ell^s = 0$, induction will start at $m = 0$: In OMH(0), the transmitter sends its value v ($v = v$ if the transmitter is non-faulty) to all receivers and itself; the received values are simply delivered as v_p. Properties (1)–(4) follow immediately from the definition of process failures in Section 3.

If $f_\ell^s > 0$, however, induction must start at $m = 1$ as the base case: According to the definition of OMH(1), every receiver q of step 1 of OMH(1) uses OMH(0) to disseminate its w_q to all receivers p (including link failure free transmission to itself, according to our additional assumption regarding self-transmission), which in turn compute the majority of the non-$\emptyset$ values delivered in OMH(0) to obtain the result v_p of OMH(1). Let us first consider the cases where the transmitter is consistent, i.e., (1), (2) and (4) above: Abbreviating the number of initially participating receivers by

$$n' \geq 2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2f_a + 2f_s + f_o + f_m + m,$$

all but at most f_ℓ^s of the obedient receivers p of step 1 of OMH(1) get the same $w_p = v$ (we can simply assume $v = \emptyset$ in case of a manifest faulty transmitter) due to the transmitter's at most f_ℓ^s link failures according to ($A1^s$). Therefore, we can apply item (b.1) of Lemma 4.2 with $f = f_\ell^s$ to conclude that every obedient receiver will deliver the same value $w_p = v$ as the result of OMH(1). In the remaining case (3), where the transmitter is omission faulty, all obedient receivers get either $w_p = v$ or $w_p = \emptyset$. In this case, item (b.2) of Lemma 4.2 with $f = f_\ell^s$ and $\mathcal{W} = \{v, \emptyset\}$ shows that every obedient receiver q must deliver either v or $\emptyset$ as required.

[6] In order to show (B2), it would be sufficient to replace obedient by non-faulty. However, our proof shows an even stronger result: OMH actually satisfies uniform validity (UB2).

Assuming now that the lemma is already true for $m-1 \geq \min\{1, f_\ell^s\}$, we will show that it is also true for m: For a consistent transmitter, we have at least $n' - f_\ell^s - f_a - f_s$ obedient receivers q of step 1 of OMH(m) that apply OMH($m-1$) to disseminate their $R(w_q) = R(v)$. Since both m and the number of participants decreased by one, we can apply the induction hypothesis to OMH($m-1$) to conclude that every obedient receiver p actually delivers $R(v)$ in this step for every non-faulty transmitter q. Consequently, any obedient receiver p must have at least

$$\overline{n}'_p = n' - f_\ell^s - f_a - f_s - f'_o - f'_m$$

values equal to $R(v)$ among the at most $\overline{n}''_p = n' - f'_a - f'_o - f'_m$ non-$\emptyset$ values it may have got at all. Herein, $f'_m \leq f_m, f'_o \leq f_o$, and $f'_a \leq f_a$ gives the number of $\emptyset$-values delivered to receiver p by OMH($m-1$) for manifest, omission, and arbitrary faulty transmitters q, respectively. Since $m > 0$,

$$\begin{aligned} 2\overline{n}'_p - \overline{n}''_p &= n' - 2f_\ell^s - 2f_a + f'_a - 2f_s - f'_o - f'_m \\ &\geq f_\ell^r + f_\ell^{ra} + f'_a + (f_o - f'_o) + (f_m - f'_m) + m \\ &> 0, \end{aligned} \tag{7}$$

so $R(v)$ wins the hybrid-majority at any obedient process p and the final delivery value $v_p = v = R^{-1}(R(v))$ follows.

For the remaining case (3), exactly the same reasoning as in the proof of item (a.2) and (b.2) of Lemma 4.2 reveals that only the default value $R(\emptyset)$ can be returned by hybrid-majority if no majority of either $R(v)$ or $R(\emptyset)$ exists. This eventually completes the proof of Lemma 4.3. □

With the help of Lemma 4.3, it is not too difficult to show the major Theorem 4.4 for OMH. Comparison with the tight consensus lower bound $n > f_\ell^s + f_\ell^{sa} + f_\ell^r + f_\ell^{ra}$ established in [14] reveals that the algorithm has close to optimal link failure resilience (as well as optimal resilience w.r.t. Byzantine process failures). The required number of rounds also matches the tight consensus lower bound $f_a + f_o + 2$ of [14] (which is not true for the simple algorithm used for proving tightness of the above lower bound in [14]).

Theorem 4.4 (*Agreement & Validity OMH*). *For any $m \geq f_a + f_o + \min\{1, f_\ell^s\}$ and $f_a, f_o, f_s, f_m, f_\ell^s, f_\ell^r, f_\ell^{ra} \geq 0$, the algorithm OMH($m$) satisfies agreement* (B1) *and validity* (B2) *if there are strictly more than $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m$ participating processes.*

Proof. Since Lemma 4.3 is applicable under the conditions of Theorem 4.4, validity (B2) is evident. Hence, we need to show agreement (B1) only.

The proof is by induction on m and is an extension of the one of [8]. In the base case $m = \min\{1, f_\ell^s\}$, we must have $f_a = f_o = 0$ since $m \geq f_a + f_o + \min\{1, f_\ell^s\}$ by assumption. Hence, the transmitter must not be arbitrary or omission faulty here, such that Lemma 4.3 also implies (B1).

We can therefore assume that (B1) is satisfied by OMH($m-1$) with $m - 1 \geq \min\{1, f_\ell^s\}$, and have to prove this for OMH(m). Again, it suffices to consider the case where the transmitter is arbitrary or omission faulty, since Lemma 4.3 also implies (B1) in the other cases. Since we have at most $f_a + f_o \leq m - \min\{1, f_\ell^s\} \geq 1$ arbitrary or omission faulty processes and the transmitter is one of those, either (1) at most $f_a - 1$ arbitrary faulty processes or (2) at most $f_o - 1$ omission faulty ones remain among the strictly more than $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m - 1$ processes. Since obviously

$$2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m - 1 > 2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2([f_a - 1] + f_s) + f_o + f_m + [m - 1]$$

as well as

$$2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m - 1 > 2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + [f_o - 1] + f_m + [m - 1],$$

we can apply the induction hypothesis and conclude that any OMH($m-1$) satisfies (B1). Hence, for any q, any two non-faulty processes deliver the same value for w_p^q in step (3) of the algorithm. Note carefully that this actually follows from (B2) if one of the two receivers is process q, and from (B1) otherwise. Consequently, any two non-faulty receivers get the same vector of values and hence the same hybrid-majority, which eventually proves (B1) for OMH(m). □

Remarks. 1. Note that the proof of agreement uses only the validity property and the fact that f_a and f_o are added to the number of rounds required by validity. Hence, every consensus algorithm that uses forwarding like OMH that achieves validity for a given m will also achieve agreement for $m' = m + f_a + f_o$.

2. Lemma 4.3 and Theorem 4.4 and their proofs are also valid if the at most f_m manifest faults were not strong but rather weak manifest faults, cf. Remark 4 in Section 3.2. Since a weak manifest fault allows the manifest faulty sender process to receive the correct value instead of $\emptyset$, it is usually harder to tolerate than a strong manifest one. This is not the case here, however, since properties (B1) and (B2) need only hold for non-faulty but not for manifest faulty processes — self-delivery at non-faulty processes is always correct since by assumption it may not be hit by a link failure.

3. The proof of Lemma 4.3 revealed that OMH satisfies *uniform* validity (UB2): Uniform validity holds not only at non-faulty but also at (alive) manifest and omission faulty processes. By using this fact in the proof of Theorem 4.4, it is easy to show that OMH provides agreement not only at non-faulty but also at strong manifest faulty processes: The induction step (last paragraph in the proof of Theorem 4.4) is valid when one of the two receivers is the manifest faulty transmitter process q here as well.

4. It is apparent from Theorem 4.4 that $2f_o$ processes are required by OMH to tolerate f_o omission faulty processes, which is definitely sub-optimal: Algorithms like the one of [3] require only $f_o < n - 1$. This is not due to an overly conservative analysis, but rather the price paid for OMH's ability to mask additional symmetric and arbitrary failures. We do not know whether this price could possibly be avoided by using a more clever algorithm.
5. From the proof of Lemma 4.3, it is apparent that $(A1^r)$ – and one additional round – is only required to eventually rule out the inconsistencies caused by $(A1^s)$. This is solely done in the base case $m = 1$ of the induction, which implies that limiting the number of link failures for a single receiver by f_ℓ^r according to $(A1^r)$ is only required in the last round, where in turn $(A1^s)$ is not explicitly used. This supports our approach of considering both types of failures independently from each other, recall Section 3.2.
6. For OMH, receive link failures $(A1^r)$ resulting in an omission are easier to tolerate than those that produce a value failure, cf. Theorem 4.4, since $f_\ell^r + f_\ell^{ra} = f_\ell^{ro} + 2f_\ell^{ra}$ with f_ℓ^{ro} bounding the number of "pure" omission failures. This is not the case for send link failures $(A1^s)$, since an omission failure appears like a value failure in all but the last round (where send link failures do not matter, recall Remark 5 above) due to wrapping all $\emptyset$-values with $R()$.

4.2. Uniform Byzantine agreement with the algorithm OMHU

In this subsection, we will consider uniform Byzantine agreement with the properties Uniform Agreement (UB1) and Uniform Validity (UB2), see Section 4.

Originally, we (erroneously) conjectured that the proofs of Lemma 4.3 and Theorem 4.4 carry over literally to uniform validity (UB2) and uniform agreement (UB1). Formal verification in [24] showed that this is indeed true for (UB2), and our proof of Lemma 4.3 actually does prove (UB2). But the proof for (B1) cannot be carried over to (UB1): OMH does not satisfy (UB1) due to the "asymmetry" of transmitter and receivers in conjunction with the ambiguity of (UB2) for omission faulty transmitters. A simple counterexample is an omission faulty transmitter that sends v to itself but $\emptyset$ to all other processes. The receivers will all agree on $\emptyset$, but the transmitter will deliver v. This problem causes the induction step in the proof of Theorem 4.4 (last paragraph) to fail in the case where one of the two receivers is the transmitter process q, since (B2) does not guarantee agreement if q is omission faulty. To guarantee (UB1), OMH must hence be modified.

In [24], a symmetric version of OMH was proposed, where the transmitter participates in recursive instances as well. This algorithm guarantees uniform validity and agreement for weak manifest faulty processes, but not for omission faulty processes.

A straightforward algorithm, which guarantees (UB1) and (UB2) without restrictions, could be built atop of OMH by letting all receivers convey their delivered value back to the transmitter of each recursive instance; the transmitter would then deliver the unwrapped result of the hybrid majority among the received values and its own value. However, this variant would almost double the number of rounds and messages required.

This overhead is avoided by the following simple uniform algorithm OMHU, which employs just a single additional round with a full message exchange: In the final round, all processes (including the transmitter) convey the wrapped value delivered locally by the original OMH to each other. The final decision value is then computed by unwrapping the result of a hybrid majority vote applied to the received values. Lemma 4.6 and Theorem 4.7 below show that OMHU satisfy uniform agreement (UB1) and uniform validity (UB2).

Definition 4.5 (*Algorithm OMHU*)**.** The Uniform Hybrid Oral Message algorithm OMHU is defined as follows:
OMHU(m), $m > 1$**:**

1. Execute OMH(m). Let w_p be the value OMH delivers to process p.
2. Every process p sends the value $R(w_p)$ to all other processes.
3. For every p and $q \neq p$, let w_p^q be the value receiver p receives from process q, with $w_p^q = \emptyset$ if no or a manifestly erroneous value was received. Every process p calculates the hybrid-majority value of all values w_p^q and its own value $w_p^p = R(w_p)$, and applies R^{-1} to that value. This result is delivered as v_p.

Lemma 4.6 (*Validity OMHU*)**.** *For any $m \geq \min\{1, f_\ell^s\}$ and any $f_a, f_o, f_s, f_m, f_\ell^s, f_\ell^r, f_\ell^{ra} \geq 0$, the uniform algorithm OMHU(m) satisfies uniform validity (UB2) if there are strictly more than $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m$ participating processes.*

Proof. From the proof of Lemma 4.3, we know that, in case of a consistent transmitter, all obedient processes in OMH deliver the same value v that is conveyed to all processes in the additional round of OMHU. Since at least $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2f_a + 2f_s + f_o + f_m + \min\{1, f_\ell^s\} + 1$ processes participate here, we can apply item (a.1) of Lemma 4.2 with $f = 0$ to conclude that every obedient process will receive enough values to eventually deliver v.

In case of an omission faulty transmitter, the obedient processes in OMH need not agree upon the same value at the end of OMH but may deliver either v or $\emptyset$. However, item (a.2) of Lemma 4.2 with $f = 0$ and $\mathcal{W} = \{v, \emptyset\}$ applies here. Since $\mathcal{W} \cup \{\emptyset\} = \mathcal{W}$, any obedient process can deliver v or $\emptyset$ only. □

Theorem 4.7 (*Validity & Agreement OMHU*)**.** *For any $m \geq f_a + f_o + \min\{1, f_\ell^s\}$ and any $f_a, f_o, f_s, f_m, f_\ell^s, f_\ell^r, f_\ell^{ra} \geq 0$, the uniform algorithm OMHU(m) satisfies uniform agreement (UB1) and uniform validity (UB2) if there are strictly more than $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m$ participating processes.*

Proof. Due to Lemma 4.6, it only remains to prove (UB1). First of all, as noted in Remark 3 on Theorem 4.4, the original OMH satisfies agreement (B1) not only at non-faulty, but also at (strong) manifest faulty processes. Hence, both non-faulty and (alive) manifest faulty processes in OMH deliver the same value v. Since there are at least $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 3f_a + 2f_s + 2f_o + f_m + \min\{1, f_\ell^s\} + 1$ participating processes here, item (a.1) of Lemma 4.2 for $f = 0$ amply ensures that all obedient processes will deliver v as the result of the full message exchange in the final round. □

Comparison with OMH shows that OMHU achieves uniform agreement and validity with the same number of processes but one additional round, in which $n(n-1)$ messages are exchanged.

5. Hybrid written messages algorithms

5.1. Authentication

Written messages [4] extend oral messages by assuming that (1) it is impossible to alter a message without being detected at the receiver, and (2) that the originator of a message's content can always be determined, even if the message comes from an intermediate process. It is generally agreed that electronic signatures can be used to achieve these goals (although there are some pitfalls [30]). With electronic signatures, each process p uses its private *signature* σ_p to sign some piece of information x, thereby generating the corresponding *signed information* $\sigma_p(x)$. Without knowing σ_p, it is computationally infeasible (ideally impossible) to compute $\sigma_p(x)$ from x. Recovering the content x of some valid signed information $\sigma_p(x)$ is easily done, however, by applying the (usually public) *inverse signature* σ_p^{-1}.

We will call the process that generated some signed information its *originator*. A *signed message* denotes a message that contains some signed information. Note that the sender of a signed message may be different from the originator of the signed information it contains.

Definition 5.1 (*Authentication Properties*)**.** We place the following specific assumptions on the signature scheme:

(SA1) Only process p can be the originator of $\sigma_p(x)$.
(SA2) Only process p can change the content x of $\sigma_p(x)$.
(SA3) The content of any signed information can be extracted at any process in the system. More specifically, we assume that any process can compute $\sigma_p^{-1}(SI) = x$ iff $SI = \sigma_p(x)$, whereas $\sigma_p^{-1}(SI) =$ ERROR for some distinguished value ERROR otherwise.

Note carefully that (SA1) implicitly requires countermeasures against replay attacks, since an eavesdropping process could otherwise inject prerecorded messages from an earlier execution run. This can be avoided by incorporating unique execution sequence numbers in messages, for example. (SA2) implies that forwarding processes cannot manipulate signed messages without being detected at the receiver. (SA3) secures that authentication does not introduce new errors (interpreting a valid signature for an invalid one) or mask present ones (generating an apparently valid x out of an incorrect SI), cf. [30].

Clearly, the above requirements hold only for *unbroken* signatures. If process p's signature has been disclosed (deliberately or not), (SA1) and (SA2) do not hold any longer. More specifically, some other process $q \neq p$ could then manufacture signed messages $\sigma_p(x)$ as if they originated from process p. Also note that assumptions (SA1) and (SA2) restrict the power of Byzantine faulty processes and links, such that these faults cannot generate a valid signed message for process p while p's signature remains unbroken.

In the subsequent sections, we will investigate authenticated Byzantine agreement algorithms that disseminate the transmitter t's value $v \in V$ by forwarding signed messages via paths of distinct processes, one hop per round, as does the non-authenticated algorithm OMH of Section 4.1: Its recursive execution can be unrolled into a sequence of m forwarding-only rounds followed by a final forwarding round $m+1$, at the end of which all the recursive instances of hybrid-majority are computed, recall Remark 4 on Definition 4.1.

In round k of every such round-by-round forwarding scheme, a value v that is sent from the transmitter $p_1 = t$ along a chain of distinct processes $p_2 \dots p_k$ arrives at some receiver q as a multiply signed message $M = \sigma_{p_k} \cdots \sigma_{p_1}(v)$. Due to failures, however, it may happen that some process $p_{\ell+1} \in \{p_2, \dots, p_k, p_{k+1}\}$ with $p_{k+1} = q$ does not receive a valid round ℓ message $\sigma_{p_\ell} \cdots \sigma_{p_1}(v)$ but rather $\emptyset$. The algorithm ZAr (see Section 5.2) simply stops forwarding in this case, so that no message M will arrive at q in this case. Alternative algorithms like OMHA and ZA cause $p_{\ell+1}$ to generate a signed message $\sigma_{p_{\ell+1}}(\emptyset_R)$ containing some specific value $\emptyset_R$ meaning "I am reporting $\emptyset$" (like the $R(\emptyset)$ of Section 4.1) in this case, which is forwarded along the remaining path as usual. Consequently, the final receiver q could get a signed message $\sigma_{p_k} \cdots \sigma_{p_{\ell+1}}(\emptyset_R)$ (or $\emptyset$ in case of an omission failure) instead of $M = \sigma_{p_k} \cdots \sigma_{p_1}(v)$ here as well.

All our algorithms need the string of forwarding process id's $p_k, \dots, p_1$ for any round k message, both for assigning an arriving message to the appropriate concurrent instance of the algorithm (recall Remark 3 on Definition 4.1) and for applying the appropriate inverse signature when successively recovering the transmitter's value v. Consequently, in a real implementation, $p_k, \dots, p_1$ must somehow be incorporated in signed messages. Moreover, any process $p_{\ell+1}$ generating a message containing $\emptyset_R$ must encode $p_\ell, \dots, p_1$ in $\emptyset_R = \emptyset_R^{p_\ell, \dots, p_1}$. (We will usually use the shorthand notation $\emptyset_R$, however, to keep the notation simple.)

The following Definition 5.2 summarizes the properties of valid information:

Definition 5.2 (*Valid Information*)**.** For $k \geq 1$, let $p_1, \ldots, p_k$ with $p_1 = t$ be a sequence of different processes. Some information M is *valid information* w.r.t. path $p_1, \ldots, p_k$ iff it satisfies the following properties:

(1) $M = \sigma_{p_k} \cdots \sigma_{p_{\ell+1}}(x)$ for some ℓ with $k > \ell \geq 0$, where all signatures in M are valid.
(2) Non-$\emptyset_R$ information: If $\ell = 0$, i.e., $M = \sigma_{p_k} \cdots \sigma_{p_1}(x)$, then $x = v \in V$ is a legitimate transmitter value.
(3) $\emptyset_R$ information: If $k > \ell > 0$, then $x = \emptyset_R^{p_\ell, \ldots, p_1}$ (only allowed in some forwarding schemes).

A received message containing valid information is called a valid message; a message containing valid non-$\emptyset_R$ resp. $\emptyset_R$ information is a valid non-$\emptyset_R$ resp. $\emptyset_R$ message. Received messages that do not contain valid information are locally detectable and hence manifest faulty. More specifically, in all our algorithms, every process q that receives a forwarded message containing $M = \sigma_{p_k} \ldots \sigma_{p_{\ell+1}}(x)$ from process p_k checks whether the message is manifest faulty, and delivers $\emptyset$ in this case. For simplicity, we assume that this check also includes the validity of the last signature σ_{p_k} in M; in reality, checking signatures will usually be deferred to the final round $m + 1$.

Although faulty processes and links can of course generate erroneous messages, their capabilities to generate *valid* messages are considerably limited due to authentication. To prove this, we first summarize the effects of authentication on our system/failure model:

- Every obedient process adheres to the particular algorithm at all times; it may hence sign and broadcast at most one (valid) message in every round.
- A symmetric faulty process may in principle perform arbitrarily internally and may sign and broadcast even a faulty message to all receivers consistently. However, it may neither disclose its signature nor sign and forward (or disseminate otherwise) multiple messages in the same round.
- An arbitrary faulty process may perform arbitrarily (except that it cannot generate non-disclosed signatures of other nodes). In particular, it may generate, sign and broadcast multiple messages in the same round, collude with other arbitrary faulty processes, disclose its signature, etc.
- Links are incapable of generating valid signatures, recall (SA1) and (SA2). However, an arbitrary faulty link could inject signed messages generated elsewhere in the system.

The following Lemma 5.3 establishes that valid forwarding information w.r.t. a path ending in a not arbitrary faulty process is unique.

Lemma 5.3 (*Information Uniqueness*)**.** *Consider round-by-round forwarding as outlined above, starting at the transmitter $p_1 = t$, and let q be a process that is not arbitrary faulty. Then, for any sequence $p_1, \ldots, p_k$ of $k \geq 0$ different processes, there is at most one valid information M w.r.t. path $p_1, \ldots, p_k$, q in the system at all times; M is generated by q in round $k + 1$.*

Proof. Assuming the contrary, there are times t and t' where $M = \sigma_q \sigma_{p_k} \cdots \sigma_{p_{\ell+1}}(x)$ and $M' = \sigma_q \sigma_{p_k} \cdots \sigma_{p_{\ell'+1}}(x')$ with $M \neq M'$, i.e., $\ell \neq \ell'$ or $x \neq x'$, or both, exist in the system.

Since both M and M' contain a valid signature $\sigma_q(.)$, both M and M' must originate from q by (SA1) and (SA2). Let $M = \sigma_q(L)$ with $L = \sigma_{p_k} \cdots \sigma_{p_{\ell+1}}(x)$ and $M' = \sigma_q(L')$ with $L' = \sigma_{p_k} \cdots \sigma_{p_{\ell'+1}}(x') \neq L$. There are only two possibilities:

(a) Both L and L' is valid information w.r.t. path $p_1, \ldots, p_k$, i.e., $\ell + 1 \leq k$ and $\ell' + 1 \leq k$ according to Definition 5.2. In this case, both M and M' must be generated by the same process q in round $k + 1$, which is not possible since q is not arbitrary faulty.
(b) L is valid information w.r.t. path $p_1, \ldots, p_k$, but $L' = \emptyset$, i.e., $\ell + 1 \leq k$ and $\ell' + 1 > k$, due to a receive omission or a manifest failure. As before, M is generated by q in round $k + 1$. In order for M' to be valid information w.r.t. path $p_1, \ldots, p_k, q$ as well, $M' = \sigma_q(\emptyset_R^{p_k, \ldots, p_1})$, i.e., $L' = \emptyset_R^{p_k, \ldots, p_1}$, must hold. This is only possible if M' is also generated by q in round $k + 1$, which leads to the same contradiction as before. □

As an immediate corollary, it follows that the out-bound links of a process q that is not arbitrary faulty with unbroken signature cannot suffer from arbitrary link failures: Since there is at most one valid forwarded message M from q, every other message generated by a link must be invalid (i.e., manifest faulty). And arbitrary link failures hitting the out-bound links of an arbitrary faulty process do not increase the process' adverse capabilities anyway. We hence obtain:

Corollary 5.4. *Consider round-by-round forwarding as outlined above. Then, arbitrary link failures need only be counted as omissive, i.e., one can set $f_\ell^{sa} = f_\ell^{ra} = 0$.*

As another almost immediate corollary of Lemma 5.3, the following Lemma 5.5 reveals that information forwarded via paths with a common prefix that includes at least one not arbitrary faulty process with an unbroken signature is unique.

Lemma 5.5 (*Value Uniqueness*)**.** *Consider round-by-round forwarding as outlined above, starting at the transmitter $p_1 = t$. Let $M = \sigma_{p_k} \ldots \sigma_{p_1}(v)$ resp. $M' = \sigma_{p'_{k'}} \ldots \sigma_{p'_1}(v')$ be two valid messages containing v resp. v', which arrive at two obedient processes p resp. p' (possibly $p = p'$) along forwarding paths of length $k \geq c$ resp. $k' \geq c$ with a common prefix of length $c \geq 1$, i.e., $p_1 = p'_1 = t, p_2 = p'_2, \ldots, p_c = p'_c$. If at least one of those c processes is not arbitrary faulty and has an unbroken signature, then $v = v'$.*

Proof. Since the message $M_c = \sigma_{p_c} \dots \sigma_{p_1}(v)$ forwarded by c is unique and $M = \sigma_{p_k} \dots \sigma_{p_{c+1}}(M_c)$ and $M' = \sigma_{p'_{k'}} \dots \sigma_{p'_{c+1}}(M_c)$, we obviously have $v = v'$. □

Lemma 5.5 immediately implies that, if the common prefix of a forwarding path is long enough, i.e., $k \geq f_a+1$, all receivers that obtain a valid non-$\emptyset_R$ message agree on the transmitter's value. We can prove an even stronger result, however:

Lemma 5.6 (*Identity*). *Consider a Byzantine agreement algorithm with $n > f_\ell^s + f_\ell^r + f_a + f_s + f_o + f_m + 1$ processes, $f_\ell^s, f_\ell^r, f_a, f_s, f_o, f_m \geq 0$, which forwards the value of the transmitter $p_1 = t$ along all possible paths of different processes as outlined above. If an obedient process p receives some value v in a valid message $M = \sigma_{p_k} \dots \sigma_{p_1}(v)$, where at least one of the processes in the set $\{p_1, \dots, p_{k-\min\{1,f_\ell^s\}}\}$ is consistent with unbroken signature, then every other non-faulty process q also gets a valid message with at most k signatures that contains v.*

Proof. Assume that p_x, $1 \leq x \leq k-\min\{1, f_\ell^s\}$ minimal, is consistent and thus sends $M_x = \sigma_{p_x} \dots \sigma_{p_1}(v)$. Note that p_x cannot act manifest faulty when broadcasting M_x, since M would not carry σ_{p_x} in this case. Since p_x must have received v correctly from itself in this case, as must have all non-faulty processes along the forwarding chain $p_1, \dots, p_{x-1}$, we can restrict our attention to obedient processes q which are not on the path $p_1, \dots, p_x$. If there are no link failures, i.e., $f_\ell^s = f_\ell^r = 0$, all those obedient processes get $M' = M_x$ and we are done. If $f_\ell^s > 0$, at least $n' \geq n - 1 - f_{p\_all} - f_\ell^s > f_\ell^r + 1$ (with the abbreviation f_{p_all} as defined in (4)) non-faulty processes get M_x and forward it. Every obedient process q thus receives at least one of those forwarded messages, despite of the at most f_ℓ^r receive link failures it might experience. □

Note carefully that Lemma 5.6 cannot be extended to obedient instead of non-faulty processes q, i.e., our lemma does not ensure uniform agreement on non-$\emptyset$ values. More specifically, it cannot be extended to omission faulty processes: If there is some omission faulty process p_o in M's path prefix $p_1, \dots, p_{x-1}$, M_x and hence v is not forwarded to p_o by p_x. Unlike in case of a consistent p_o, it cannot be guaranteed that the value contained in M_x has already been self-delivered to p_o either, since $\emptyset$ might have been received instead. Hence, p_o could fail to get some value v present at other obedient processes.

It is instructive to also consider the question of how to model broken signatures. More specifically, assume that some arbitrary faulty processes, in addition to their own signatures, also know the signatures of at most f_b not arbitrary faulty processes. Knowing the signature of some process p allows malicious processes to generate signed messages on behalf of p, which makes p to appear arbitrary faulty. However, Lemmas 5.5 and 5.6 reveal that malicious processes cannot do more. Consequently, the problem of incorporating f_b additional broken signatures can be solved by increasing f_a to $f_a + f_b$.

Since the power of faulty processes is considerably restricted when using written messages, authenticated Byzantine agreement algorithms should have much better fault-tolerance capabilities than non-authenticated algorithms. For example, it follows immediately from Lemma 5.6 that all non-faulty processes get the same set of values $\notin \{\emptyset, \emptyset_R\}$ if the transmitter's value is forwarded via all paths of length $k > f_a + f_o + \min\{1, f_\ell^s\}$. This reveals, for example, that the simple authenticated algorithm SMH of [8], which just forwards the transmitter's value for sufficiently many rounds and then takes the hybrid-majority of all received values, achieves agreement and validity with as few as $n > f_\ell^s + f_\ell^r + f_a + f_s + f_o + f_m + 1$ processes.

In the following subsection, we will introduce and analyze a suite of Hybrid Written Messages Algorithms derived from the non-authenticated OMH. Starting from a "naive" application of authentication in algorithm OMHA, the effects and benefits of authentication will be gradually explored and exploited for improving certain aspects, including link failure resilience, process failure resilience and message complexity. The final algorithm ZAr has optimal resilience and polynomial message complexity, but depends critically upon unbroken signatures.

5.2. Algorithm ZAr

We start our treatment of authenticated algorithms by informally[7] introducing an authenticated variant of OMH, derived from the algorithm OMHA developed in [30], under the system model of Sections 3 and 5.1: It is the same as OMH, except that every message sent in OMHA(m) for every $m \geq 0$ is signed. Moreover, the signature is used as the wrapper function R, and both missing and manifest faulty messages are reported by a distinguished non-empty value $\emptyset_R \neq \emptyset$. Note that $\emptyset_R$ is also the default value returned by hybrid-majority (as opposed to the $R(\emptyset)$ value in Section 4.1) if no majority exists.

The correctness proof for OMH in Section 4.1 applies literally for OMHA, although for different parameter settings ($f_\ell^{sa} = f_\ell^{ra} = 0$). Unfortunately, the performance of OMHA turns out to be no better than that of OMH − except that link failure resilience is slightly improved:

Theorem 5.7 (*Validity & Agreement OMHA [52, Lem. 5.8]*). *For any $m \geq f_a + f_o + \min\{1, f_\ell^s\}$ and any $f_a, f_s, f_o, f_m, f_\ell^s, f_\ell^r \geq 0$, algorithm OMHA($m$) satisfies agreement* (B1) *and validity* (B2) *if there are strictly more than $2f_\ell^s + f_\ell^r + 2(f_a + f_s) + f_o + f_m + m$ participating processes.*

Whereas adding authentication to OMH in OMHA only improves the resilience to link failures, migrating the authenticated algorithm ZA of [30] to the failure model of Section 3 also improves the resilience to process failures. Informally, ZA just uses the unique empty value $\emptyset_R = \emptyset$ (which is also the default value returned by hybrid-majority

[7] To save space, the detailed description and analysis of the algorithms OMHA and ZA have been relegated to a research report [52].

in the absence of a majority), unlike OMHA, which used a distinguished reporting value $\emptyset_R$. As a consequence, the only values that may occur during the execution of ZA are the value(s) sent by the transmitter and $\emptyset$. There is hence no need for an "overwhelming" number of non-faulty values in order to compensate $\emptyset_R$ values; even a single non-$\emptyset$-value is sufficient here.

However, the correctness of the resulting algorithm ZA depends critically upon unbroken signatures of not arbitrary faulty processes (but recall the end of Section 5.1 for how to deal with additional broken signatures).

The following Theorem 5.8 shows that ZA satisfies validity (B2) and agreement (B1) with optimal resilience. Comparison with the tight consensus lower bound $n > f_\ell^s + f_\ell^r$ established in [14] reveals that the algorithm has optimal link failure resilience (as well as optimal resilience w.r.t. Byzantine process failures). The required number of rounds also matches the tight consensus lower bound $f_a + f_o + 2$ of [14].

Theorem 5.8 (*Agreement and Validity ZA [52, Thm. 5.9]*)**.** *For any $m \geq f_a + f_o + \min\{1, f_\ell^s\}$ and any $f_a, f_s, f_o, f_m, f_\ell^s, f_\ell^r \geq 0$, algorithm ZA(m) satisfies agreement* (B1) *and validity* (B2) *if there are strictly more than $f_\ell^s + f_\ell^r + f_a + f_s + f_o + f_m + 1$ participating processes.*

Although ZA proceeds like OMH, the information flow develops quite differently: According to Lemma 5.6, every process gets sufficient information to achieve validity within the first two rounds of ZA in case of a not arbitrary faulty transmitter, regardless of the number of rounds actually employed. This observation finally leads to the optimal algorithm ZAr introduced below.

In fact, even though ZA has a much better resilience than OMHA, it does not fully exploit the benefits of authenticated messages: We know from Lemma 5.6 that every non-faulty receiver obtains the same set of distinct (unsigned) values $\neq \emptyset$ at the end of ZA(0). There is no need, however, to receive (and hence forward) multiple signed messages containing the same value v. Moreover, there is no use in forwarding $\emptyset$-values, since they are discarded at the end anyway. Note that item (3) in Definition 5.2 is hence void here.

In the algorithm ZAr given in Definition 5.9 below, every process p maintains a set w_p of legitimate values seen so far, and forwards only new legitimate values that are added to w_p in a round. ZAr dramatically reduces both message and computational complexity of ZA, from exponential to polynomial. Interestingly, ZAr turned out to be essentially the same as the authenticated algorithm for atomic broadcasting under Byzantine failures proposed in [53,41].

Definition 5.9 (*Algorithm ZAr*)**.** Let $\mathcal{W}_p$, initially $\mathcal{W}_p = \{\}$, be the set of legitimate unsigned values already seen by process p during the execution. Moreover, let $val(M)$ be the value v contained in a valid signed message after removing all signatures, and $val(M) = \emptyset$ otherwise.

ZAr(0):

1. The transmitter t sends the signed message $w_t = \sigma_t(v)$ to every receiver, and adds $val(w_t)$ to $\mathcal{W}_t$ if $val(w_t) \neq \emptyset$.
2. For every receiver p, if p has received a valid message w_p containing a value $\emptyset \neq val(w_p) \notin \mathcal{W}_p$, then p adds $val(w_p)$ to $\mathcal{W}_p$.

ZAr(*k*), $m \geq k > 0$**:**

1. The transmitter t sends the signed message $w_t = \sigma_t(v)$ to every receiver, and adds $val(w_t)$ to $\mathcal{W}_t$ if $val(w_t) \neq \emptyset$.
2. For every receiver p, if p has received a valid message w_p that satisfies $\emptyset \neq val(w_p) \notin \mathcal{W}_p$, then p acts as the transmitter in algorithm ZAr($k-1$) to disseminate w_p to all receivers.
3. End of ZAr(m) only: For every process q, if $\mathcal{W}_q$ contains a single legitimate value v_q, then q delivers v_q, otherwise it delivers $\emptyset$.

Using Lemma 5.6, it is not difficult to show that ZAr satisfies validity and agreement:

Lemma 5.10 (*Validity ZAr*)**.** *For any $m \geq \min\{1, f_\ell^s\}$ and any $f_a, f_s, f_o, f_m, f_\ell^s, f_\ell^r \geq 0$, algorithm ZAr(m) satisfies validity* (B2) *if there are strictly more than $f_\ell^s + f_\ell^r + f_a + f_s + f_o + f_m + 1$ participating processes.*

Proof. By Lemma 5.6, we know that $\min\{1, f_\ell^s\} + 1 \leq 2$ rounds suffice to ensure that every obedient process gets the transmitter's value v if the transmitter is non-faulty or symmetric faulty. If the transmitter is manifest faulty, no receiver ever gets a non-$\emptyset$ value. In case of an omission faulty transmitter, Lemma 5.5 ensures that $\emptyset$ and v are the only possible values. Hence, validity is always fulfilled. □

Theorem 5.11 (*Validity & Agreement ZAr*)**.** *For any $m \geq f_a + f_o + \min\{1, f_\ell^s\}$ and any $f_a, f_s, f_o, f_m, f_\ell^s, f_\ell^r \geq 0$, algorithm ZAr(m) satisfies validity* (B2) *and agreement* (B1) *if there are strictly more than $f_\ell^s + f_\ell^r + f_a + f_s + f_o + f_m + 1$ participating processes.*

Proof. Due to Lemma 5.10, it only remains to show agreement. In our proof, we use the fact that every non-faulty process obtains the same set of values at the end of round $m + 1$ by Lemma 5.6. We only have to look at values that "survive" forwarding until ZAr(0), because forwarding stops only if a particular value is already contained in some process's $\mathcal{W}_p$ — but in that case, the earlier message must have already been forwarded. Note carefully that it does not matter here that any two signed message containing the same v are forwarded along different paths: Although the sets of receivers are different, they differ only in the processes on the already taken paths. All non-faulty ones among those, however, must have received v by self-reception and, consequently, do not need forwarding.

The signature chains in the final round have length $m+1 \geq f_a + f_o + 1 + \min\{1, f_\ell^s\}$. Consequently, there must be at least one consistent process p_x in each such chain $p_1, \ldots, p_{m+1}$ with $1 \leq x \leq m+1-\min\{1, f_\ell^s\}$. If p_x is non-faulty or symmetric faulty, Lemma 5.6 guarantees agreement. If p_x is manifest faulty, no signed message with $m+1$ signatures incorporating σ_{p_x} can be received at any process. Hence, every non-faulty process obtains the same set of non-$\emptyset$ values. □

Remarks. 1. During the first two rounds, ZAr sends the same number of non-$\emptyset$ messages as ZA. In subsequent rounds, however, ZAr is clearly superior with respect to communication complexity: From Definition 5.9, it is obvious that every value v sent by the transmitter is forwarded by any process at most once to all $n-1$ receivers. Hence, every v causes at most $(n-1)^2$ messages during forwarding, so the worst case occurs if the transmitter sends a different value to each of its $n-1$ receivers; recall that no other value except $\emptyset$ can be generated in the system. Consequently, at most $n-1+(n-1)^3$ messages can be sent system-wide.

2. If the initial transmitter is arbitrary faulty and sends multiple values, then ZAr will always deliver $\emptyset$. However, ZA might deliver one of the sent values $v \neq \emptyset$ in this case, if v has been received by a majority of processes.

We will conclude this section by briefly comparing ZAr to the authenticated algorithm for atomic broadcasting under Byzantine failures proposed in [41]. The latter was analyzed in a more abstract failure model, where the only requirement is that the removal of faulty processes and links does not partition the network. For example, non-faulty processes might even be connected in a chain there, in which case the atomic broadcast algorithm would need $n-1$ rounds. By contrast, we showed in [14, Thm. 9] that our link failure model ensures that any two non-faulty processes are connected to each other by at least one non-faulty path of length $\min\{1, f_\ell^s\} + 1 \leq 2$, which explains the comparatively small number of rounds required by ZAr.

Generally, our approach has the advantage over [41] that it models link failures explicitly on a per-process-basis, rather than by their effect on the communication graph. In addition, whereas [41] provides a suite of algorithms each targeted to a specific class of failures, our algorithms have been developed for a comprehensive hybrid failure model. As already noted in Remark 4 on Theorem 4.4, however, this comes at the price of a suboptimal resilience with respect to benign failures: ZAr needs $n > 2f_o + 1$ instead of the $n > f_o + 1$ processes of the omission-tolerant algorithm in [41] for tolerating f_o omission failures.

6. Polynomial algorithms

In the previous section, we investigated Byzantine agreement algorithms with (almost) optimal resilience and round complexity. However, except for the authenticated algorithm ZAr, they all exchange an exponential number of messages, or messages of exponential size. In this section, we will focus on non-authenticated consensus and Byzantine agreement algorithms with polynomial communication complexity: We will show that (1) existing algorithms can be employed in our hybrid perception-based failure model with minor modifications, and (2) that algorithms with sub-optimal resilience with respect to some failure semantics may have good or even optimal resilience for other failure types.

The algorithms analyzed in the following two subsections solve *binary consensus*, that is, assuming that every process p is provided with some *initial value* $x_p \in V = \{0, 1\}$, the consensus algorithm computes a *decision value* v_q at every process q that satisfies the following properties:

(C1) *(Uniform) agreement*: If processes p and q are both obedient, then both compute the same $v_p = v_q$.
(C2) *Validity*: If all obedient processes start with the same initial value $\forall p : x_p = v$, then any non-faulty process q computes $v_q = v$.

Before going into the details of the polynomial protocols (which we will present in the following subsections), we first elaborate on some consequences of restricting the initial values to be either 0 or 1. When we say that in some round a *full message exchange* is performed, then every (obedient) process sends a message to every process (including itself). When every message has to contain one of at most two values, then the following lemma can be used to bound the difference between the number of times two processes see one of these values:

Lemma 6.1 (*Difference in Perceptions*). *Let $C_r[v]$ and $C_q[v]$ be the number of messages containing v received at two obedient processes r and q in a full message exchange of a system complying with the model of Section 3. Then, $|C_q[v] - C_r[v]| \leq f_a + f_o + f_\ell^r + f_\ell^{ra}$.*

Proof. Assuming w.l.o.g. $C_q[v] > C_r[v]$, at most f_a arbitrary faulty senders could have sent v to process q and $x \neq v$ to process r, and at most f_o processes could have sent v to q but no message to r. Process failures hence contribute at most $f_a + f_o$ to $|C_q[v] - C_r[v]|$. The remaining terms originate from link failures: Process q could have received at most f_ℓ^{ra} messages containing v from processes that actually sent $x \neq v$, and at most f_ℓ^r messages containing v could have been lost in transit to process r. □

When using a majority test after a full message exchange (e.g., as in line 7 in Fig. 1), inconsistent reception could produce a majority for the value v at some process q but a majority for $x \neq v$ at some process $r \neq q$. The question arises how big the lead of v must be at process q to be able to guarantee that r will compute the same majority? This is answered by the following Lemma 6.2.

```
1:  for k := 1 to f_a + f_s + f_o + f_m + 2 do
2:    /* Round 1: full message exchange */
3:    send(v) to all nodes [including itself]
4:    receive(v_q) from all nodes
5:    C[0] := |{v_q : v_q = 0}|
6:    C[1] := |{v_q : v_q = 1}|
7:    if C[1] > C[0] then v := 1 else v := 0
8:    /* Round 2: queen's broadcast */
9:    if k = p then
10:     send(v) to all nodes [including itself]
11:   receive(v_queen)
12:   if v_queen = ∅ then v_queen := 0
13:   if C[v] ≤ C[1 − v] + 2f_a + f_o + 2f_ℓ^r + 2f_ℓ^ra then
14:     v := v_queen
```

Fig. 1. Hybrid Phase Queen algorithm, code for process p.

Lemma 6.2 (*Deviation of Differences*)**.** *Let $C_r[v]$ and $C_q[v]$ be the number of messages containing $v \in \{0, 1\}$ received at two obedient processes r and q in a binary full message exchange of a system complying with Section 3. For $\Delta_q[v] = C_q[v] - C_q[1-v]$ and $\Delta_r[v] = C_r[v] - C_r[1-v]$, we obtain $|\Delta_q[v] - \Delta_r[v]| \leq 2f_a + f_o + 2f_\ell^r + 2f_\ell^{ra}$.*

Proof. We denote by Δ the difference $\Delta_q[v] - \Delta_r[v]$ and we assume, w.l.o.g., that it is non-negative. Clearly, ignoring the effect of link failures for the moment, the broadcasts of processes with consistent behavior (i.e., correct, symmetric faulty and manifest faulty) are received in the same way by all processes, and thus have no influence on Δ (they do, of course, influence $\Delta_q[v]$).

When an inconsistently faulty process sends v to q and $1 - v$ to r (instead of sending v to both), then (as r receives one more message containing $1 - v$ and one less with v) $\Delta_r[v]$ can become smaller than $\Delta_q[v]$ by two. Thus, Δ is increased by two by this failure. An omission faulty process can either fail to send $1 - v$ to q or v to r, but not both. In any case, Δ is increased by at most one here.

Turning to the worst case deviation caused by link failures, we observe that for each link failure that q experiences the same arguments as above apply, i.e., each of the f_ℓ^{ra} arbitrary receive link failures contributes two to Δ, while each omissive one contributes one to Δ. The same is true for the reception of r, possibly for a different set of affected in-bound links. Thus, in the worst case, the $2f_\ell^{ra}$ arbitrary link failures at q and r contribute up to $4f_\ell^{ra}$ to Δ, while the $2(f_\ell^r - f_\ell^{ra})$ link omissions at q and r contribute up to $2(f_\ell^r - f_\ell^{ra})$. Adding up these numbers, we arrive at a total of $2f_a + f_o + 2f_\ell^r + 2f_\ell^{ra}$ as asserted. □

6.1. Phase queen algorithm

The *Phase Queen Protocol* introduced in [31] is a simple binary consensus algorithm for systems with $n > 4f$ processes, at most f of which may be arbitrary faulty, and no link failures. It assumes unique process identifiers $\in \{1, \ldots, n\}$, uses constant size (1-bit) messages and takes $f + 1$ phases, each consisting of 2 rounds. Our contribution is a modified version of the original algorithm that can cope with hybrid process and link failures according to the system model of Section 3.

Fig. 1 shows the pseudocode of our hybrid algorithm, which consists of $f_{p\_all} + 2 = f_a + f_s + f_o + f_m + 2$ *phases* made up of two rounds each. In the first round of every phase, every process p broadcasts its current preference value v_p to all processes in the system (including itself) and collects the resulting messages from its peers (i.e., a full message exchange). Process p then counts how many processes sent preferences for 0 and 1, respectively, and updates its v_p accordingly. In the second round of each phase, only one specific process (the *Phase Queen*, whose identifier is equal to the current phase number) broadcasts its new preference. This value is used by all processes in the system to break ties.

Note that we made only two non-trivial changes to the original algorithm: First, the bound in the decision when to use the queen's broadcast (line 13 in Fig. 1) had to be adapted to our hybrid failure model. Second, in order to improve the resilience with respect to non-arbitrary failures, the original condition $C[1] > n/2$ in the update of the preference value (line 7) had to be replaced by the simple majority test $C[1] > C[0]$. Note carefully that a message omission leads to a value $\emptyset \notin \{0, 1\}$, i.e., is neither counted in $C[1]$ nor in $C[0]$.

In the following, we call a process p which does not emit any messages from round k onwards (i.e., $sm_p^{k'}[q]$ for all q and all $k' \geq k$) to be *actually dead*, and let d^k denote the number of actually dead processors among the manifest or omission faulty ones by the beginning of phase k; obviously $f_m + f_o \geq d^i \geq d^j$ must hold[8] for all $i \geq j$. We will need this value to discriminate between obedient processes and benign faulty processes that are dead.

[8] This allows clean as well as unclean crash failures. Whereas a clean crash can be accounted for in f_m, an unclean one is an asymmetric omission in one round followed by symmetric omissions in all later round, and must thus be accounted for in f_o.

As a first step in showing the correctness of the Phase Queen protocol, we look at what happens when a certain amount of processes all share the same value. More specifically, we show that once sufficiently many processes have the same preference value, it will not change any more. Lemma 6.3 will subsequently be used to show the validity property (C2).

Lemma 6.3 (*Persistence of Agreement*). *If at least $n - f_a - f_s - d^k - f_\ell^s$ obedient processes prefer $v \in \{0, 1\}$ at the beginning of phase $k \in \{1, \ldots, f_{p\_all} + 2\}$ in a system with $n > 4f_a + 2f_s + 2f_o + f_m + 2f_\ell^s + 3f_\ell^r + 3f_\ell^{ra}$, then all obedient processes prefer v at the end of phase k.*

Proof. Since $n - f_a - f_s - d^k - f_\ell^s$ obedient processes prefer v at the beginning of phase k, every obedient process q receives $C_q[v] \geq n - f_{p\_all} - f_\ell^s - f_\ell^r$ preferences for v in the first round of phase k. Moreover, since the preference of $f_a + f_s + f_\ell^s$ alive processes was left unspecified in our assumptions and at most f_ℓ^{ra} links may cause erroneous values, process q could also get $C_q[1 - v] \leq f_a + f_s + f_\ell^s + f_\ell^{ra}$ preferences for $1 - v$. This implies

$$\begin{aligned} C_q[v] &\geq n - f_{p\_all} - f_\ell^s - f_\ell^r \\ &> 3f_a + f_s + f_o + f_\ell^s + 2f_\ell^r + 3f_\ell^{ra} \\ &\geq C_q[1 - v] + 2f_a + f_o + 2f_\ell^r + 2f_\ell^{ra}, \end{aligned}$$

so every obedient process q updates its preference v_q to v in line 7 in Fig. 1 and ignores the queen's broadcast in line 13. □

The second major lemma is a prerequisite for showing that agreement can be reached by means of a non-faulty queen breaking tie.

Lemma 6.4 (*Agreement*). *Let g be a phase whose queen is non-faulty. Then, at least $n - f_a - f_s - d^{g+1} - f_\ell^s$ obedient processes finish phase g with the same preference.*

Proof. Since the queen g is non-faulty, it sends the same v_g to all receivers. Assume first that all obedient processes use the value v_{queen} received from g as their new preference in line 14 of Fig. 1. According to ($A1^s$), at most f_ℓ^s of those could have received a queen's preference $v_{queen} \neq v_g$,[9] such that at least $n - f_a - f_s - d^{g+1} - f_\ell^s$ obedient processes have set their preference to v_g by the end of phase g.

If, on the other hand, some obedient process p ignores the queen's broadcast and uses its own majority value v as its new preference, $C_p[v] - C_p[1 - v] > 2f_a + f_o + 2f_\ell^r + 2f_\ell^{ra}$ according to line 13 in Fig. 1. Hence, for every other obedient process q (including the non-faulty queen g), $C_q[v] - C_q[1 - v] \geq C_p[v] - C_p[1 - v] - 2f_a - f_o - 2f_\ell^r - 2f_\ell^{ra} > 0$ by Lemma 6.2, so g must have set its preference to v in line 7 as well. Process p can therefore safely use its majority value instead of the queen's in this case, since they are the same. □

By means of Lemmas 6.3 and 6.4, it is not difficult to prove the following major Theorem 6.5. Comparison with the tight lower bound $n > f_\ell^s + f_\ell^{sa} + f_\ell^r + f_\ell^{ra}$ from [14] reveals that the algorithm has sub-optimal link failure resilience (as well as sub-optimal resilience w.r.t. Byzantine process failures). The required number of rounds is also more than twice the tight lower bound $f_a + f_o + 2$ of [14].

Theorem 6.5 (*Phase Queen*). *Under the system model of Section 3 with $f_a, f_s, f_o, f_m, f_\ell^r, f_\ell^{ra}, f_\ell^s \geq 0$ and $n > 4f_a + 2f_s + 2f_o + f_m + 2f_\ell^s + 3f_\ell^r + 3f_\ell^{ra}$, the Phase Queen algorithm of Fig. 1 solves binary consensus in $2(f_a + f_s + f_o + f_m + 2)$ rounds with a total of $(f_a + f_s + f_o + f_m + 2)(n + 1)$ 1-bit message broadcasts from obedient processes.*

Proof. Lemma 6.3 already implies the validity property (C2): If all obedient processes start out with the same value v, they will continue to prefer v until the algorithm terminates after $f_{p\_all} + 2$ phases, since there are $n - f_a - f_s - d^k - f_\ell^s \geq n - f_{p\_all} - f_\ell^s$ non-faulty processes, in any phase k.

To show agreement (C1), we note that all but at most f_ℓ^s obedient processes will have the same preference at the end of a non-faulty queen's phase g by Lemma 6.4. The bound on the number of faulty processes ensures that at least one phase $g \in \{1, \ldots, f_{p\_all} + 1\}$ has a non-faulty queen. Since the algorithm takes $f_{p\_all} + 2$ phases, Lemma 6.3 assures that all obedient processes will have the same (persistent) preference by the end of phase $g + 1$, no matter how many phases with faulty queens follow.

To justify the claimed time and communication complexity of our algorithm, we note that there is one full message exchange consisting of n broadcasts in the first round and one additional queen's broadcast in the second round of every phase. Hence, at most $(f_{p\_all} + 2)(n + 1)$ 1-bit message broadcasts are performed during the whole execution by processes that faithfully[10] follow the algorithm in Fig. 1. □

As a final remark, we note that the number of message broadcasts of the Phase Queen algorithm could be reduced by one by omitting the queen's broadcast in the last phase, since – according to the proof of Theorem 6.5 – no obedient process uses v_{queen} in the last phase.

[9] At most f_ℓ^{sa} processes could receive the opposite value $1 - v_g$, and the remaining $f_\ell^s - f_\ell^{sa}$ ones could suffer from omissions, that is, deliver $\emptyset$.

[10] Clearly, there is no way to restrict the number of message broadcasts initiated by an arbitrary faulty process.

1: **for** $k := 1$ to $f_a + f_s + f_o + f_m + 2$ **do**
2: /* *Round 1: initial full message exchange* */
3: **send**(v) **to all nodes** [*including itself*]
4: **receive**(v_q) **from all nodes**
5: $C[0] := \left|\left\{v_q : v_q = 0\right\}\right|$
6: $C[1] := \left|\left\{v_q : v_q = 1\right\}\right|$
7: /* *Round 2: C[j] full message exchange* */
8: **for all** $j \in \{0, 1\}$ **do**
9: **if** $C[j] > C[1-j] + f_a + f_o + f_\ell^r + f_\ell^{ra}$ **then**
10: $M[j] := 1$
11: **else**
12: $M[j] := 0$
13: **send**(M) **to all nodes** [*including itself*]
14: **receive**(M_q) **from all nodes**
15: $\forall j \in \{0, 1\} : D[j] := \left|\left\{q : M_q[j] = 1\right\}\right|$
16: **if** $D[1] > f_a + f_s + f_\ell^{ra}$ **then** $v := 1$ **else** $v := 0$
17: /* *Round 3: king's broadcast* */
18: **if** $k = p$ **then**
19: **send**(v) **to all nodes** [*including itself*]
20: **receive**(v_{king})
21: **if** $v_{king} = \emptyset$ **then** $v_{king} := v$
22: **if** $D[v] \leq 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{ra}$ **then**
23: $v := v_{king}$

Fig. 2. Hybrid Phase King algorithm, code for process p.

6.2. Phase King algorithm

The *Phase King Protocol* of [31] improves the Phase Queen algorithm by adding one round to every phase, in which the information on the processes' possible preferences after the first round are exchanged system-wide. This reduces the sub-optimal process failure resilience $n > 4f_a$ of the Phase Queen algorithm to the optimal $n > 3f_a$. We provide a hybrid variant of this algorithm in Fig. 2, which can cope with process and link failures according to Section 3.

The proof for the Phase King Protocol is similar to the one for the Phase Queen Protocol: The first Lemma 6.6 will show that once sufficiently many processes have the same preference value, it will not change any more.

Lemma 6.6 (*Persistence of Agreement*)**.** *If at least $n - f_a - f_s - d^k - f_\ell^s$ obedient processes prefer v at the beginning of phase $k \in \{1, \ldots, f_{p\_all} + 2\}$ in a system with $n > 3f_a + 2f_s + 2f_o + f_m + 2f_\ell^s + 2f_\ell^r + 2f_\ell^{ra}$ processes, then all obedient processes prefer v at the end of phase k.*

Proof. Since at least $n - f_{p\_all} - f_\ell^s$ non-faulty processes broadcast v in the first full message exchange, every obedient process q obtains $C_q[v] \geq n - f_{p\_all} - f_\ell^s - f_\ell^r$ and $C_q[1-v] \leq f_a + f_s + f_\ell^s + f_\ell^{ra}$ according to our failure model in Section 3. Similar to the proof of Lemma 6.3, this leads to

$$C_q[v] > C_q[1-v] + f_a + f_o + f_\ell^r + f_\ell^{ra},$$

which implies that every non-faulty process among the obedient ones will send the same $M[j]$ in the second round. Therefore, every obedient process r obtains $D_r[v] \geq n - f_{p\_all} - f_\ell^r$ and $D_r[1-v] \leq f_a + f_s + f_\ell^{ra}$ and thus sets its local preference to the same value v in line 16 in Fig. 2; the king's value v_{king} is ignored in line 22 since $D_r[v] > 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{ra}$. □

The second major lemma is again a prerequisite for showing that agreement can be reached by means of a non-faulty king breaking tie.

Lemma 6.7 (*Agreement*)**.** *Let g be a phase whose king g is non-faulty. If $n > 3f_a + 2f_s + 2f_o + f_m + 2f_\ell^s + 2f_\ell^r + 2f_\ell^{ra}$, then at least $n - f_a - f_s - d^{g+1} - f_\ell^s$ obedient processes start phase $g + 1$ with the same preference.*

Proof. At the end of phase g, one of the following cases applies in line 22 of Fig. 2:
(1) $D_p[v_p] \leq 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{ra}$ for every obedient process p, which thus assigns the value v_{king} to v_p. Since at most f_ℓ^s processes could have received a value v_{king} different from the value v_g actually sent, at least $n - f_a - f_s - d^{g+1} - f_\ell^s$ obedient processes end up with the same preference v_g as asserted.
(2) $D_p[v_p] > 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{ra}$ for some obedient process p. We have to show that p's preference at the end of the phase is equal to the king's value v_g in this case, which will again imply that all obedient processes except at most f_ℓ^s will end up with a preference for the king's value v_g — either by receiving it, or by being convinced of it in the first place. We distinguish the two possible cases here: If $v_p = 1$, it must be that $D_g[1] > f_a + f_s + f_\ell^{ra}$ and hence $v_g = 1$, since Lemma 6.1

implies $D_g[1] \geq D_p[1] - f_a - f_o - f_\ell^r - f_\ell^{ra} > f_a + f_s + f_\ell^{ra}$. If, on the other hand, $v_p = 0$, we can use the following argument to show that v_g cannot be 1:

We first show that if an obedient process q sends $M_q[b] = 1$ for some $b \in \{0, 1\}$ in the second round, then no obedient process r can send $M_r[1 - b] = 1$. Assuming the contrary, line 9 in Fig. 2 would require both $\Delta_q = C_q[b] - C_q[1 - b] > f_a + f_o + f_\ell^r + f_\ell^{ra}$ and $-\Delta_r = C_r[1 - b] - C_r[b] > f_a + f_o + f_\ell^r + f_\ell^{ra}$, and hence $\Delta_q - \Delta_r > 2f_a + 2f_o + 2f_\ell^r + 2f_\ell^{ra}$, which would contradict Lemma 6.2.

Now, since we have $D_p[0] > 2f_a + f_s + f_o + f_\ell^r + 2f_\ell^{ra}$, at least one non-faulty process must have sent $M[0] = 1$ in the second round. Consequently, as we have just shown, no obedient process can have sent $M[1] = 1$. Since this implies $D_q[1] \leq f_a + f_s + f_\ell^{ra}$ for any obedient process q (including the king g), v_g is set to 0 in line 16, just before the king's broadcast. This eventually completes the proof of Lemma 6.7. □

We can now show the following major Theorem 6.8. Comparison with the lower bound $n > f_\ell^s + f_\ell^{sa} + f_\ell^r + f_\ell^{ra}$ from [14] reveals that the algorithm has again sub-optimal link failure resilience. Unlike the Phase Queen algorithm, it has optimal resilience w.r.t. Byzantine process failures, however. The required number of rounds is more than three times the lower bound $f_a + f_o + 2$.

Theorem 6.8 (*Phase King*)**.** *Under the system model of Section 3 with $f_a, f_s, f_o, f_m, f_\ell^r, f_\ell^{ra}, f_\ell^s, f_\ell^{sa} \geq 0$ and $n > 3f_a + 2f_s + 2f_o + f_m + 2f_\ell^s + 2f_\ell^r + 2f_\ell^{ra}$, the Phase King algorithm of Fig. 2 solves binary consensus using $4(f_a + f_s + f_o + f_m + 2)$ message exchanges with a total of $(f_a + f_s + f_o + f_m + 2)(3n + 1)$ bits being sent by obedient processes.*

Proof. Since Lemmas 6.6 and 6.7 are the same as Lemmas 6.3 and 6.4, respectively, the proof of the agreement (C1) and validity property (C2) is literally the same as in Theorem 6.5. To justify the claimed time and message complexity, Fig. 2 reveals that every process broadcasts three bits per phase in the first two rounds. Adding the single message broadcast by every phase's king, we arrive at $(f_{p\_all} + 2)(3n + 1)$ bits being sent by obedient processes. □

As for the Phase Queen algorithm, it would again be possible to omit the king's broadcast in the last round of this algorithm.

6.3. Srikanth & Toueg algorithm

In this section, we will analyze a hybrid version of the simple binary Byzantine agreement algorithm of [29] under the system model of Section 3. Unlike OMH and the other Byzantine agreement algorithms considered earlier in this paper, which follow the asymmetric "Byzantine Generals" style [4] involving a single transmitter and $n - 1$ receivers, the Byzantine agreement algorithms considered here are more symmetric in that all n processes [including the transmitter] act as receivers. Note that this allows one to directly apply consensus lower bounds, recall the discussion in Section 4.

Recall also from Section 4 that a Byzantine agreement algorithm disseminates some value v of the transmitter to all receivers. Within a fixed number of rounds, every non-faulty receiver p must *deliver* a value v_p ascribed to the transmitter that satisfies the following properties (with $\emptyset := 0$):

(B1) *Agreement*: If receivers p and q are both non-faulty, then both deliver the same $v_p = v_q$.
(B2) *Uniform Validity*: If receiver p is obedient, the value v_p delivered by p is
- v, if the transmitter is non-faulty,
- v or $\emptyset$, if the transmitter is omission or manifest faulty,
- the value actually sent, if the transmitter is symmetric faulty,
- unspecified, if the transmitter is arbitrary faulty.

Note that there is a change in the validity property with respect to manifest faulty transmitters. This is necessary to allow for algorithms where the transmitter also participates in the algorithm after sending out its initial value (as this is the case in the composition of Algorithm 3 with one of the broadcasting algorithms introduced later on).

The algorithm of [29] is built upon a reliable *broadcast service* **broadcast**(m, k) and the corresponding **accept**(s, m, k), where s denotes the identifier of the *sending process*, m is the message, and k denotes the round in which s broadcasts m. The semantics of the broadcast service is fully captured by the following three uniform [42] properties:

(C) *Uniform correctness:* If a non-faulty process s executes **broadcast**(m, k) in round k, then every obedient process **accepts**(s, m, k) in the same round.
(U) *Uniform Unforgeability:* If obedient process s never executes **broadcast**(m, k), then no obedient process ever **accepts**(s, m, k).
(R) *Uniform Relay:* If an obedient process **accepts**(s, m, k) in round $r \geq k$, then every obedient process also **accepts**(s, m, k) in round $r + 1$ or earlier.

One way to implement the above broadcast primitive in systems without link failures is by means of signed messages, as in Section 5.1, in conjunction with relaying, i.e., forwarding of accepted messages to all other processes in the system. While algorithms based on authenticated messages, like the one of Fig. 3, are simple, easy to understand and often have superior fault-tolerance properties (Compare OMH with ZA, in Section 4.1 resp. 5.2), there are some drawbacks:

```
1: /* Transmitter t */
2: if p = t then v := m else v := 0
3: /* Receiver p */
4: for l := 1 to f_a + f_s + f_o + f_m + 1 do
5:    if v = 1 and not yet broadcast(_ , _ ) then
6:       broadcast(1, l)
7:    if accepted(p_k, 1, l_k) from l distinct p_k including t then
8:       v := 1
```

Fig. 3. The hybrid binary Byzantine agreement algorithm ST, code for process p.

Cryptographic methods (see for example [54] for an overview) typically used for authentication are expensive operations, which cannot always be afforded in power/bandwidth-limited systems. Moreover, there is always a non-zero probability of guessing or forging a process's signature, thus no present cryptographic authentication technique is unconditionally secure. An authenticated algorithm's fault-tolerance properties, however, depend critically upon an uncompromised cryptosystem.

Moreover, cryptography does not help in overcoming the effects of link failures, as the above properties cannot be guaranteed when messages can be lost.

Those deficiencies make the non-authenticated implementation of the broadcast primitive developed by Srikanth & Toueg in [29] attractive. Instead of using cryptography, their *simulated broadcast primitive* relies upon the idea of letting all processes witness a process's message broadcast. Using this approach, the algorithm of [29, Fig. 2] unconditionally guarantees the correctness (C), unforgeability (U) and relay (R) properties, provided that $n > 3f$ and at most f processes are arbitrary faulty. In order to do, so processes have to exchange messages in the course of multiple rounds, even if the broadcasting process is non-faulty. In Sections 6.3.2 and 6.3.3, we present variants of this simulation (cf. [51, Chapter 12]) which tolerate link failures.

All these simulations use multiple rounds to ensure the three properties of the broadcast primitive. In order to avoid confusion, we will refer to the rounds that implement the broadcasting primitive as rounds, while the (macro-)rounds of the algorithm using the primitive are referred to as phases.

6.3.1. Hybrid Byzantine agreement (algorithm ST)

Fig. 3 shows our hybrid version of the algorithm for binary Byzantine Agreement of [29], which is itself based upon the algorithm of [55]. Algorithm ST differs from the original algorithm only in that the number of faulty processes t has been replaced by $f_{p\_all} = f_a + f_s + f_o + f_m$ and that we hid relaying, which was done explicitly in [29, Fig. 1], within **broadcast**( ) for simplicity.

The algorithm proceeds in $f_{p\_all} + 1$ phases, where the only value broadcast by obedient processes is 1; the value 0 is decided upon by default. An obedient transmitter broadcasts 1 in phase 1 if agreement is to be reached on the value $m = 1$. Any obedient receiver p sets $v := 1$ at the end of phase l and hence decides 1, if it has accepted l messages from different processes $P_l = \{p_1, \ldots, p_l\}$, one of which must be the transmitter. Note that $p \notin P_l$ since p still has $v = 0$ during phase l. The assumed correctness and relay properties ensure that all other correct processes accept all messages $\in P_l$ plus the one broadcast by p by phase $l + 1$, hence they will also decide 1 by then.

If $m = 0$ is the value to be agreed upon, an obedient transmitter never broadcasts any message. By the unforgeability property, no obedient receiver ever accepts any message originating from the transmitter. Consequently, no obedient process sets $v := 1$ and all must hence decide 0 by default. Note that a manifest faulty transmitter is indistinguishable from a correct transmitter that tries to communicate $m = 0$, which explains why we assumed $\emptyset := 0$ for the validity property (B2).

By formalizing the above line of reasoning, the following Theorem 6.9 shows that our hybrid algorithm achieves agreement (B1) and validity (B2), provided that the underlying communication system guarantees (C), (U) and (R).

Theorem 6.9 (*Hybrid Authenticated Algorithm*)**.** *Given a broadcast primitive that guarantees* (C)*,* (U) *and* (R) *under the system model of Section 3, the hybrid algorithm ST of Fig. 3 achieves properties* (B1) *and* (B2) *of Byzantine agreement within* $f_a + f_s + f_o + f_m + 1$ *phases provided that* $n \geq f_a + f_s + f_o + f_m + 1$*. Moreover, every obedient process calls* **broadcast**( ) *at most once during the whole execution.*

Proof. We first show that validity (B2) is achieved. Since (B2) is void in case of an arbitrary faulty transmitter, we only have to deal with the following cases:

(1) If the transmitter t is non-faulty and agreement is to be reached on 0, then t does not call **broadcast**( ). Hence, no obedient process will ever **accept**(t, 1, _) due to (U) and all obedient receivers will decide upon the default value $v = 0$. If, on the other hand, agreement is to be reached on 1, a non-faulty transmitter t calls **broadcast**(1, 1). By (C), all obedient processes will accept this message in phase 1 and set $v := 1$ as required.

(2) If the transmitter has $m = 0$ and is omission faulty or manifest faulty during its broadcast (that is, it either does not call **broadcast**( ) at all, because it crashed or it broadcasts an obviously bad value), then (U) ensures that no non-faulty

1: /* *Code for initial phase k* */
2: /* *Round 2k: Broadcaster only (entered by* **broadcast**(m, k)*)* */
3: **send**$(init, m, k)$ **to all nodes** [*including itself*];

4: /* *Round* $2k + 1$*: All processes* */
5: **if received**$(init, m, k)$ from s in round $2k$ **then**
6: **send**$(echo, s, m, k)$ **to all nodes** [*including itself*]
7: **if received**$(echo, s, m, k)$ in round $2k + 1$ from $\geq n - f_a - f_s - f_o - f_m - f_\ell^s - f_\ell^r$ **then**
8: **accept**(s, m, k) [once]

9: /* *Code for phase* $l \geq k + 1$*, i.e., rounds* $2l$*,* $2l + 1$*: All processes* */
10: **if received**$(echo, s, m, k)$ in the previous round from $\geq n - 2f_a - f_s - 2f_o - f_m - f_\ell^s - 2f_\ell^r - f_\ell^{ra}$
or sent$(echo, s, m, k)$ in previous round **then**
11: **send**$(echo, s, m, k)$ **to all nodes** [*including itself*]
12: **if accepted**(s, m, k) in previous phase **then**
13: **terminate**
14: **if received**$(echo, s, m, k)$ in this round from $\geq n - f_a - f_s - f_o - f_m - f_\ell^s - f_\ell^r$ **then**
15: **accept**(s, m, k) [once]

Fig. 4. The hybrid version of the broadcast primitive for simulated authentication of Srikanth & Toueg, code for process p.

process will ever **accept**$(t, 1, \_)$. Consequently, all non-faulty receivers will decide upon the default value $v = 0 = \emptyset$ as required.

(3) If the transmitter has $m = 1$ and is omission faulty or manifest faulty (as above), either value is allowed, thus validity holds trivially in this case.

(4) If the transmitter is symmetric faulty, it appears[11] like a non-faulty transmitter (but with a faulty value). Hence, case (1) above applies with m equal to the value actually sent.

In order to prove the agreement property (B1), it suffices to show that if one non-faulty process sets $v := 1$, then all other obedient processes do the same; agreement on 0 occurs per default. Let q be the first non-faulty process that executes $v := 1$, and l be the phase in which this happens. Then, q must have accepted at least l different messages containing a value of 1 in phase l. We distinguish two cases: If $l < f_{p\_all} + 1$, then any obedient process p will accept these messages due to (R) at most one phase later as well. Since p will also accept q's broadcast of $(q, 1, l+1)$ in phase $l+1$ due to (C), it will eventually end up with $v := 1$ as required.

If, on the other hand, $l = f_{p\_all} + 1$, then q must have accepted at least $f_{p\_all} + 1$ messages with a value of 1. However, at least one of these messages originates from a non-faulty process, which must have executed $v := 1$ in some phase $l' < l$. This contradicts our assumption of q being the first process to do so, so at least one non-faulty process must call **accept**( ) strictly before the last phase. □

6.3.2. Hybrid simulated broadcast primitive

In this section, we present a variant of the simulated broadcast primitive of [29, Fig. 2], which works in the model of Section 3.1. Like the original algorithm, our hybrid version (Fig. 4) proceeds in rounds, where two kinds of messages are exchanged. When some process s starts a broadcast by calling **broadcast**(m, k) in phase k, it enters the code in round $2k$. It first sends a message $(init, m, k)$ to all processes in the system, which is answered by every witness with a message $(echo, s, m, k)$ in round $2k + 1$. A process calls **accept**(p, m, k) in some round $l \geq k$ when it receives $(echo, s, m, k)$ from at least $n - f_{p\_all} - f_\ell^s - f_\ell^r$ different processes. When the initial broadcaster is non-faulty (or symmetric faulty) then every process will receive this number of *echo*-messages in round $2k + 1$. When the initial broadcaster p is faulty, additional rounds may be necessary (or the message might never be accepted). In these additional rounds, a process that did not yet send $(echo, s, m, k)$ could get sufficient evidence that it should do so, which could in turn convince some processes to accept later on. In fact, the collusion of a faulty broadcaster with faulty witnesses could let the number of rounds until acceptance of the message grow without bounds, i.e., prohibit termination. Still, this can only happen if no obedient process has accepted yet: By the relay property, all obedient processes accept by the end of the round following acceptance of the first non-faulty process. Note that since every phase consists of two rounds, processes may accept the message in different phases.

In the original algorithm [29, Fig. 2], any process p can terminate its instance involved in **broadcast**(m, k) started by s after having called **accept**(s, m, k), since any other still active process in the system must have seen p's echo message by then. However, in order to deal with link failures, we had to change this detail in a subtle but important way: Link

[11] Recall from Section 3 that a symmetric faulty process consistently sends information that is not obviously bad. A wrong transmitter identifier p or phase number r in a message (p, m, r) would be detected at any receiver.

failures can produce f_ℓ^{ra} erroneous $(echo, s, m, k)$ messages per round at any obedient process, which could accumulate over multiple rounds. Thus we consider only messages received in the current round, which in turn requires that we retransmit $(echo, s, m, k)$ in every additional round (instead of sending it only once and remembering an earlier reception at all active processes). Since other processes may accept one round after q, any process must retransmit $(echo, s, m, k)$ up to and including the round following acceptance.

A similar analysis as used for the asynchronous broadcast primitives in [56,46] leads to the following Theorem 6.10. Note that in our context the maximum running time L below is in fact known in advance, since all broadcast instances can be terminated (even without accepting a message) once the Byzantine agreement algorithm ST using this primitive has terminated.

Theorem 6.10 (*Simulated Broadcast 1*)**.** *Under the system model of Section 3 with* $f_a, f_s, f_o, f_m, f_\ell^r, f_\ell^{ra}, f_\ell^s, f_\ell^{sa} \geq 0$ *and* $n > 3f_a + 2f_s + 2f_o + f_m + f_\ell^s + f_\ell^{sa} + 2f_\ell^r + 2f_\ell^{ra}$*, the simulated broadcast primitive of Fig. 4 guarantees correctness* (C)*, uniform unforgeability* (U)*, and uniform relay* (R)*. During* $0 < L$ *phases (where* L *is the maximum number executed), at most* $(2L-1)n+1$ *broadcasts of* $(1 + \log_2 n + |m| + \log_2 L)$*-bit messages are performed by obedient processes.*

Proof. We show each of the three properties separately:
Uniform correctness: Since the sender s is non-faulty, $n - f_{p\_all} - f_\ell^s$ non-faulty processes receive the message $(init, m, k)$ in round $2k$ and **send**$(echo, s, m, k)$ to all processes in round $2k+1$. Hence, every obedient process receives $(echo, s, m, k)$ from at least $n - f_{p\_all} - f_\ell^s - f_\ell^r$ distinct processes in round $2k + 1$ and thus **accepts**(s, m, k) in round $2k + 1$, i.e., in phase k.
Uniform Unforgeability: The proof of this property proceeds by contradiction: Assume that some obedient process p **accepts**(s, m, k) in some phase $l \geq k$, although the obedient process s did not execute **broadcast**(m, k) and hence did not send any $(init, m, k)$ message in round $2k$. In this case at most f_ℓ^{sa} obedient processes might have received $(init, m, k)$ as the result of a spurious send link failure at s, such that at most $f_a + f_s + f_\ell^{ra} + f_\ell^{sa}$ $(echo, s, m, k)$ messages might arrive at any obedient process. Since $f_a + f_s + f_\ell^{ra} + f_\ell^{sa} < n - 2f_a - f_s - 2f_o - f_m - f_\ell^s - 2f_\ell^r - f_\ell^{ra}$, no obedient process, except the at most f_ℓ^{sa} ones that received an *init* message in the first round and thus retransmit *echo* (cf. line 10), will ever execute **send**$(echo, s, m, k)$ in line 11 of Fig. 4. Since p executed **accept**(s, m, k), however, it must have received $(echo, s, m, k)$ from at least $n - f_{p\_all} - f_\ell^s - f_\ell^r > f_a + f_s + f_\ell^{ra} + f_\ell^{sa}$ processes, which provides the required contradiction.
Uniform Relay: Let k' be the phase in which the first obedient process p accepts (s, m, k) in round $l \in \{2k', 2k' + 1\}$. Process p must have received at least $n - f_{p\_all} - f_\ell^s - f_\ell^r$ *echo*-messages from different processes in round l, which implies that every other non-faulty process must have received at least $n - 2f_a - f_s - 2f_o - f_m - f_\ell^s - 2f_\ell^r - f_\ell^{ra}$ *echo*-messages as well by Lemma 6.1. According to line 10 of Fig. 4, every non-faulty process thus emits an *echo*-message in round $l + 1$. Therefore, by round $l + 1$, every obedient process will receive $(echo, s, m, k)$ at least $n - f_{p\_all} - f_\ell^r$ times and will hence **accept**(s, m, k). From $2k' \leq l \leq 2k' + 1$ we get that the round $l + 1$ must either be in in phase k' or $k' + 1$.

As far as the claimed message complexity of **broadcast**( ) is concerned, it is of course again impossible to bound the number of message broadcasts of non-obedient processes. Every process that faithfully executes the algorithm of Fig. 4, however, executes at most one broadcast of $(echo, s, m, k)$ in every round following the initial one, where only process p broadcasts $(init, m, k)$. Hence, system-wide, there are at most $n+1$ broadcasts in the initial phase, and $2n$ broadcasts in every subsequent phase. Hence, during $0 < l \leq L$ phases, at most $1 + n + 2(l - 1)n = (2l - 1)n + 1$ broadcasts are performed. The claimed size of $(1 + \log_2 n + |m| + \log_2 L)$ bits per message follows simply from the layout of the echo-messages. □

We can plug in this hybrid simulated broadcast primitive into the hybrid algorithm ST of Section 6.3.1 to obtain a non-authenticated algorithm ST1 for Byzantine agreement. Note that the bound L upon the number of phases required by Theorem 6.10 is enforced by the termination of the algorithm in Fig. 3.

Theorem 6.11 (*Algorithm ST1*)**.** *Under the system model of Section 3 with* $f_a, f_s, f_o, f_m, f_\ell^r, f_\ell^{ra}, f_\ell^s, f_\ell^{sa} \geq 0$ *and* $n > 3f_a + 2f_s + 2f_o + f_m + f_\ell^s + f_\ell^{sa} + 2f_\ell^r + 2f_\ell^{ra}$*, algorithm ST1 (the algorithm of Fig. 3 in conjunction with the simulated broadcast primitive of Fig. 4) achieves binary Byzantine agreement in at most* $2(f_a + f_s + f_o + f_m + 1)$ *(basic communication) rounds, with a total of at most* $(2(f_a + f_s + f_o + f_m + 1) - 1)n^2 + n$ *broadcasts of* $\Theta(\log_2 n)$*-bit messages from obedient processes.*

Proof. The major part of our theorem follows immediately by combining Theorem 6.9 with Theorem 6.10.

As far as the claimed time and message complexity is concerned, we know from Theorem 6.9 that the algorithm terminates after $f_{p\_all} + 1$ phases, so any simulated broadcast primitive (of this algorithm) at an obedient process can be terminated by phase $L = f_{p\_all} + 1 \leq n$ and could generate at most $(2(f_{p\_all} + 1) - 1)n + 1$ broadcasts of $(2 \log_2 n + 2)$-bit messages by Theorem 6.10. Since each of the at most n obedient processes calls **broadcast**( ) at most once by Theorem 6.9, the proof of Theorem 6.11 is completed. □

6.3.3. Alternative hybrid simulated broadcast primitive

The algorithm presented in the previous section is clearly not optimal with respect to link failure resilience. In this section, we turn our attention to an alternative broadcasting primitive given in Fig. 5, which is optimal in this respect. This comes at a price, however: The latter requires 3 rounds per phase, instead of only 2, which makes the first alternative more viable when speed is important. To understand how the additional round improves the resilience, it is instructive to investigate where the sub-optimality of the protocol in Fig. 4 comes from: It is the uncertainty about the number of processes sending echo-messages. More specifically, a process can accept a message only when it is sure that every other obedient process will

```
1: /* Code for initial phase k */
2: /* Round 3k: Broadcaster only (entered by broadcast(m, k)) */
3: send(init, m, k) to all nodes [including itself]

4: /* Round 3k + 1 and following: All processes */
5: if received(init, m, k) from s in round 3k then
6:    send(echo, s, m, k) to all nodes [including itself] [once]
7: if received(echo, s, m, k) from q then
8:    send(confirm, s, m, k, q) to all nodes [including itself]
9: if received(confirm, s, m, k, q) from at least n − f_a − f_s − f_o − f_m − f_ℓ^s − f_ℓ^r processes then
10:   witness(s, m, k) = witness(s, m, k) ∪ {q}
11: if |witness(s, m, k)| ≥ n − f_a − f_s − f_o − f_m − f_ℓ^s then
12:   accept(s, m, k)
13: if |witness(s, m, k)| > f_a + f_s + f_ℓ^sa then
14:   send(echo, s, m, k) to all nodes [including itself] [once]
15: if accepted(p, m, k) 2 rounds ago then
16:   terminate
```

Fig. 5. An alternative broadcasting primitive for simulated authentication, code for process p.

receive sufficiently many echo-messages that trigger sending its own echo-message. Our third round can be understood as a way to reduce the uncertainty about the number of processes that sent echo-messages.

Just like in the protocol of Fig. 4, a broadcast is started by some process s by sending an $(init, m, k)$ message to all processes also in the algorithm of Fig. 5. Every process receiving such a message echoes this message as $(echo, s, m, k)$. In an additional round, all the processes exchange information about which processes they have received echo-messages from: Every process confirms that it has received an echo-message from q by broadcasting $(confirm, s, m, k, q)$. If a process receives sufficiently many such confirmations for one process q, it will add q to its set of witnesses. A process will accept only when there are enough witnesses for the broadcast. Moreover, when some process p did not receive the init-message for some broadcast, it will send an echo-message later on only if its witness set becomes large enough (unlike in Fig. 4, where this is triggered by the reception of sufficiently many echo-messages). Note that p's echo has to be confirmed by sufficiently many other processes before p is added to some witness set. Consequently, the acceptance of a broadcast at some obedient process can be delayed up to two rounds after the first obedient process has accepted. This is compatible with the specification of the relay property (R), however, since every phase is made up of three rounds here.

From Fig. 5, it is apparent that the exchange of *echo* and *confirm* messages is very similar to the broadcast primitive of Fig. 4. The statements of the following Lemma 6.12 can in fact be understood as variants of Correctness (C) and Unforgeability (U) involving the sending of a *echo* message (= broadcasting) and the inclusion of the sender in the *witness* set at all obedient processes.

Lemma 6.12. *If*

(1) *a process p consistently*[12] *sends an echo message for some broadcast, every obedient process adds p to its set of witnesses at most two rounds later.*
(2) *an obedient process p never sends an echo message for some broadcast, then no obedient process ever adds p to its witness set, provided that $n > 2f_a + 2f_s + f_o + f_m + f_\ell^s + f_\ell^r + f_\ell^{sa} + f_\ell^{ra}$.*

Proof. (1) When a process q consistently sends *echo* for some broadcast in round r, all but f_ℓ^s correct processes will receive it and respond with a *confirm* message. This results in every obedient process receiving $n - f_{p\_all} - f_\ell^s - f_\ell^r$ of these messages, and thus to add q to the appropriate *witness* set in round $r + 2$.
(2) When a process does not send *echo* for some broadcast, at most $f_a + f_s + f_\ell^{sa}$ processes may send a "spurious" *confirm* message for some process q in any round. Every obedient process p could hence receive up to $f_a + f_s + f_\ell^{sa} + f_\ell^{ra}$ such confirmations. Since $f_a + f_s + f_\ell^{sa} + f_\ell^{ra} < n - f_{p\_all} - f_\ell^s - f_\ell^r$, this is not sufficient for p to add q to its *witness* set (or to perform any other action that might cause this indirectly) in any round. □

Theorem 6.13 (*Simulated Broadcast 2*)**.** *Under the system model of Section 3 with $f_a, f_s, f_o, f_m, f_\ell^r, f_\ell^{ra}, f_\ell^s, f_\ell^{sa} \geq 0$ and $n > 2f_a + 2f_s + f_o + f_m + f_\ell^s + f_\ell^{sa} + \max(f_a + f_o, f_\ell^r + f_\ell^{ra})$, the simulated broadcast primitive of Fig. 5 guarantees correctness* (C)*, uniform unforgeability* (U)*, and uniform relay* (R)*. During $0 < L$ phases (where L is the maximum number executed), fewer than $1 + n + n^2$ broadcasts of $(2 + 2\log_2 n + |m| + \log_2 L)$-bit messages are performed by obedient processes.*

Proof. *Correctness:* When a correct process s sends *init*, then all obedient processes except at most f_ℓ^s receive that message. Every correct process q among those responds with an *echo* message. By Lemma 6.12, q will be added to the *witness* set at

[12] That is, p is either non-faulty or symmetric faulty, or is manifest or omission faulty but behaves correctly in round k (recall Section 3.2).

each obedient process p within two rounds. These sets thus grow to at least $n - f_{p\_all} - f_\ell^s$ members, which causes p to accept within 3 rounds, i.e., within the same phase.

Unforgeability: If s never sends an *init* message for a broadcast, at most f_ℓ^{sa} obedient processes and $f_a + f_s$ faulty ones can reply with a matching *echo* message. These messages can make the *witness* set at any obedient process p grow to a size of $f_a + f_s + f_\ell^{sa}$, which is not sufficient to trigger the broadcast of additional echo messages. Consequently, due to (2) of Lemma 6.12, p's witness set cannot grow further in later rounds either. Since $f_a + f_s + f_\ell^{sa} < n - f_{p\_all} - f_\ell^s$, process p will never call accept for this broadcast.

Relay: When some obedient process p accepts a broadcast, at least $n - f_{p\_all} - f_\ell^s$ processes must be in its *witness* set at the end of some round r. By (1) of Lemma 6.12, all processes in p's witness set that have consistently broadcast their *echo* message must also show up in the *witness* sets of all other obedient processes by the end of round r. Thus, every obedient process ends up with at least $n - f_{p\_all} - f_\ell^s - f_a - f_o > f_a + f_s + f_\ell^{sa}$ processes in its *witness* set, which causes all correct ones among those to broadcast an *echo* message in round $r + 1$. This will cause the witness set of every obedient process to grow to $n - f_{p\_all} > n - f_{p\_all} - f_\ell^s$ by the end of round $r + 2$, since all the confirm messages will have been received by then. Note that rounds r and $r + 2$ may lie in the same or in adjacent phases.

The bound on the number of processes in the statement of Theorem 6.13 comes from combining $n - f_{p\_all} - f_\ell^s - f_a - f_o > f_a + f_s + f_\ell^{sa}$, (i.e., $n > 3f_a + 2f_o + 2f_s + f_m + f_\ell^s + f_\ell^{sa}$) as required for Relay (and Unforgeability), and $n > 2f_a + 2f_s + f_o + f_m + f_\ell^s + f_\ell^r + f_\ell^{sa} + f_\ell^{ra}$ as required for Lemma 6.12.

As far as the claimed message complexity is concerned, every process that faithfully executes the algorithm of Fig. 5 executes at most one broadcast of *echo* in the round following the initial one (where only process s issues a single broadcast of *init*), followed by at most n broadcasts of *confirm* messages (for every *echo* received) in the subsequent round. Hence, there are at most $1 + n + n^2$ broadcasts in the initial phase. Moreover, note that processes do not repeatedly retransmit echo-messages (as in our first primitive). Therefore, only those processes which did not send *echo* in the round following the initial one will send an *echo* message in a later round, such that the *total* number of *echo* messages sent by obedient processes is bounded by n. This – in turn – implies that no more than n^2 *confirm* messages are ever sent. Thus this number of broadcasts by obedient processes is independent of L, and the same as the bound given for the initial phase above. The claimed size of at most $(2 + 2\log_2 n + |m| + \log_2 L)$ bits per message follows simply from the layout of messages. □

By plugging in the hybrid simulated broadcast primitive of Fig. 5 into the "generic" hybrid algorithm ST introduced in Section 6.3.1, a non-authenticated algorithm ST2 for Byzantine agreement is obtained that tolerates hybrid process and link failures. The following major Theorem 6.14 establishes its properties. Comparison with the lower bound $n > f_\ell^s + f_\ell^{sa} + f_\ell^r + f_\ell^{ra}$ from [57] reveals that it has optimal resilience with respect to link failures. Like the other polynomial algorithms studied in the previous two sections, however, it suffers from a definitely sub-optimal round complexity, which is more than three times the lower bound.

Theorem 6.14. *Under the system model of Section 3 with* $f_a, f_s, f_o, f_m, f_\ell^r, f_\ell^{ra}, f_\ell^s, f_\ell^{sa} \geq 0$ *and* $n > 2f_a + 2f_s + f_o + f_m + f_\ell^s + f_\ell^{sa} + \max(f_a + f_o, f_\ell^r + f_\ell^{ra})$*, the algorithm ST2 (Fig. 3 in conjunction with the simulated broadcast primitive of Fig. 5) achieves binary Byzantine agreement in at most* $3(f_a + f_s + f_o + f_m + 1)$ *rounds, with a total of less than* $n(1 + n + n^2)$ *broadcasts of* $\Theta(\log_2 n)$*-bit messages from obedient processes.*

Proof. The major part of our theorem follows immediately from combining Theorem 6.9 with Theorem 6.13. As far as the claimed message complexity is concerned, we can see from Fig. 3 that every obedient process calls **broadcast**( ) at most once by Theorem 6.9, the proof of Theorem 6.14 is completed. □

Theorem 6.9 also allows us to derive a lower bound for the number of processes required for establishing *Correctness, Unforgeability* and *Relay*, cp. [29]: If there were a broadcast primitive that needed only $f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$ processes, it would be possible to solve Byzantine agreement by means of the algorithm of Fig. 3, thereby violating the lower bound for consensus [14]. We thus have the following Theorem 6.15, which is tight due to Theorem 6.14:

Theorem 6.15 (*Lower Bound*)**.** *Achievement of Correctness* (C)*, Unforgeability* (U)*, and Relay* (R) *under the system model of Section 3 requires* $n > f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$ *processes.*

7. Discussion

In the previous sections, we introduced and analyzed a suite of deterministic agreement algorithms, which prove that it is not too difficult to adapt such algorithms to work correctly also in the presence of link failures – despite the impossibility result of [15,10] – if the number of send and receive link failures is suitably restricted. In this section, we will relate the respective results to each other and to the lower bounds established in the companion paper [14], and discuss some consequences.

Summary of accomplishments

We introduced a perception-based hybrid failure model, which grants every process at most f_ℓ^r receive link failures (with at most f_ℓ^{ra} non-omission failures among those) and f_ℓ^s send link failures in each round, in addition to at most f_a, f_s, f_o, f_m arbitrary, symmetric, omission, and manifest process failures. Apart from its ability to handle transient and mobile link failures, in a round-by-round manner, our model can also deal with incomplete communication graphs.

For $m \geq f_a + f_o + 1$, we analyzed several $m+1$-round Byzantine agreement algorithms under our failure model, namely, the non-authenticated OMH as well as its authenticated variants OMHA and ZA, ZAr. Their respective required number of processes is

$$\begin{aligned} n_{OMH} &> 2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m, \\ n_{OMHA} &> 2f_\ell^s + f_\ell^r + 2(f_a + f_s) + f_o + f_m + m, \\ n_{ZA,ZAr} &> f_\ell^s + f_\ell^r + f_a + f_s + f_m + 1. \end{aligned}$$

We also proposed and analyzed a uniform variant of OMH, which achieves agreement and validity even at obedient processes, at the cost of one additional round.

With respect to message complexity, OMH, OMHA and ZA are exponential; ZAr is a polynomial algorithm that requires authentication. We also considered several non-authenticated polynomial algorithms, namely, the Phase Queen and Phase King consensus algorithms, and the Byzantine agreement algorithms ST1 and ST2 (which consist of Srikanth & Toueg's algorithm in conjunction with two different broadcasting primitives) under our failure model. The respective number of processes was found to be

$$\begin{aligned} n_{PhQ} &> 2f_\ell^s + 3f_\ell^r + 3f_\ell^{ra} + 4f_a + 2f_s + 2f_o + f_m, \\ n_{PhK} &> 2f_\ell^s + 2f_\ell^r + 2f_\ell^{ra} + 3f_a + 2f_s + 2f_o + f_m, \\ n_{ST1} &> f_\ell^s + f_\ell^{sa} + 2f_\ell^r + 2f_\ell^{ra} + 3f_a + 2f_s + 2f_o + f_m, \\ n_{ST2} &> f_\ell^s + f_\ell^{sa} + \max(f_\ell^r + f_\ell^{ra}, f_a + f_o) + 2f_a + 2f_s + f_o + f_m. \end{aligned}$$

Compared to the exponential algorithms like OMH, the round complexity of the polynomial algorithms is quite bad:

$$\begin{aligned} m_{OMH} &= f_a + f_o + 2, \\ m_{PhQ} &= 2(f_a + f_s + f_o + f_m + 2), \\ m_{PhK} &= 3(f_a + f_s + f_o + f_m + 2), \\ m_{ST1} &= 2(f_a + f_s + f_o + f_m + 1), \\ m_{ST2} &= 3(f_a + f_s + f_o + f_m + 1). \end{aligned}$$

Benefits of authentication

It is well-known that authentication considerably improves the resilience of consensus algorithms against arbitrary process failures. Our analysis revealed that the same is true for arbitrary link failures, which are effectively prohibited by authentication: Neither f_ℓ^{ra} nor f_ℓ^{sa} appears in the required number of processes of OMHA, ZA and ZAr.

Moreover, it turned out that authenticated algorithms specifically designed for written messages are superior over algorithms adapted from solutions based on oral messages: Whereas OMHA does not profit much from authentication, ZA and ZAr benefit considerably—but also depend critically upon it. In particular, OMHA degrades to OMH and hence only requires an additional f_ℓ^{ra} in the number of processes even if all signatures are broken, whereas every process with a compromised signature must be considered as arbitrary faulty and therefore counted in f_a in ZA.

Process failures

With respect to process failures, there are several interesting observations. First, it is apparent that (send) omission faulty processes appear as $2f_o$ in the required number of processes, which contrasts with the optimal resilience of $n > f_o$ established in [3]. We conjecture that this is inevitable in hybrid failure models if there are also arbitrary faulty processes in the system ($f_a > 0$). Moreover, it turns out that an algorithm (Phase Queen) may have a considerably sub-optimal resilience with respect to some class of process failures (arbitrary), but may nevertheless have very good tolerance against other failures.

The probably most striking result is the required number of nodes of algorithm ST2. It reveals that there is some "sharing" of resource redundancy for tolerating process and link failures: Rather than just the sum $f_\ell^r + f_\ell^{ra} + f_a + f_o$, the expression for n_{ST2} contains a term $\max\{f_\ell^r + f_\ell^{ra}, f_a + f_o\}$. Hence, nodes required for tolerating arbitrary process failures are also used for tolerating arbitrary link failures, and vice versa.

Link failures

In a companion paper [14], we established lower bounds for the required number of processes and rounds in our link failure model: For omission resp. arbitrary link failures, $n > f_\ell^r + f_\ell^s$ resp. $n > f_\ell^s + f_\ell^{sa} + f_\ell^r + f_\ell^{ra}$ processes are required. In order to tolerate f process crashes (or more severe failures), in addition to link failures, at least $f + 2$ rounds are required. Our results reveal that these lower bounds are indeed all tight, cp. the expressions for *ZAr*, n_{ST2} and m_{OMH} above.[13]

[13] Recall our discussion of how to convey consensus lower bounds to Byzantine agreement algorithms with $n - 1$ receivers in Section 4.

Table 1
Overview of our results: Resilience against arbitrary process failures (all other resiliences are the same, hence $n_0 = Cf_a + 2(f_s + f_o) + f_m$), and additional costs of tolerating $f_\ell^r = f_\ell^s = f_\ell$ omission resp. arbitrary link failures in terms of number of processes and number of rounds (m_0 is the number of rounds for $f_\ell = 0$).

Algorithm	Auth.?	$\mathbf{Cf_a}$	Omissions	Arbitrary	# rounds
OMH	No	$3f_a$	$n_0 + 3f_\ell$	$n_0 + 4f_\ell$	$m_0 + 1$
OMHA	Yes	$3f_a$	$n_0 + 3f_\ell$	$n_0 + 3f_\ell$	$m_0 + 1$
ZA&ZAr	Yes	f_a	$n_0 + 2f_\ell$	$n_0 + 2f_\ell$	$m_0 + 1$
PhQ	No	$4f_a$	$n_0 + 5f_\ell$	$n_0 + 8f_\ell$	m_0
PhK	No	$3f_a$	$n_0 + 4f_\ell$	$n_0 + 6f_\ell$	m_0
ST1	No	$3f_a$	$n_0 + 3f_\ell$	$n_0 + 6f_\ell$	m_0
ST2	No	$3f_a$	$n_0 + 2f_\ell$	$n_0 + 4f_\ell$	m_0

In order to simplify the comparison of our algorithms with respect to link failures, we summarize the associated costs for $f_\ell^s = f_\ell^r = f_\ell$ in Table 1: The second resp. third column gives the number of processes required for tolerating f_ℓ omission resp. f_ℓ arbitrary link failures[14]; n_0 denotes the number of processes required for $f_\ell = 0$, where only process failures are present. The last column gives the number of rounds for either type of link failure; m_0 is the number of rounds for $f_\ell = 0$.

First, we note that all our algorithms tolerate up to $\mathcal{O}\big((m+1)n^2\big)$ link failures during the whole execution; recall that f_ℓ could be as much as $\mathcal{O}(n)$. This dramatically outperforms the $\lfloor (n-2)/2 \rfloor = \mathcal{O}(n)$ resilience of previous work on Byzantine agreement under link failures (see Section 2). It is important to understand, though, that this does not mean that our algorithms [except ST2] are resilient to link failures *per se*. After all, we had to add $\mathcal{O}(f_\ell)$ processes to n_0 in order to mask f_ℓ link failures per process, which means that we added $\mathcal{O}(n^2)$ links. What we really gained, however, is that any link – and not just the ones added – may experience a failure.

Considering our exponential algorithms, they all match the lower bound for the required number of rounds developed in [14] and are hence optimal in this respect. As far as the required number of processes is concerned, our best algorithms need only $2f_\ell$ additional processes to cope with f_ℓ (purely omissive) link failures per process in each round; for $f_\ell = 1$, the algorithm of Srikanth & Toueg with the improved broadcast primitive needs only $n = 3$ processes (in the absence of process failures), for example. In the case where all f_ℓ link failures are arbitrary ones, OMH also matches the appropriate lower bound; OMH is hence optimal for $f_\ell^s = f_\ell^{sa}$, but slightly suboptimal for $f_\ell^{sa} < f_\ell^s$, cf. Remark 6 on Theorem 4.4.

Comparison of the link failure resilience of the Phase Queen and King algorithms with the lower bounds reveals considerable sub-optimality, in particular in case of arbitrary link failures. Note, however, that both algorithms use constant-size (1 or 2 bit) messages, which makes them particularly suitable for very-low-bandwidth situations. The algorithms ST1 and ST2, on the other hand, inherit their link failure resilience from the underlying broadcast primitives. Depending on the number of rounds one wants to spend, the resilience varies from the optimal $2f_\ell$ (omissions only) and $4f_\ell$ (arbitrary link failures only) for ST2 (3 rounds per broadcast) to $4f_\ell$ resp. $6f_\ell$ for ST1 (2 rounds), with message sizes in the order of $\log n$.

7.1. Model coverage

In [14], we analyzed the model coverage of "generic" distributed algorithms in a simple probabilistic "benchmarking scenario", where every link in a system of n processes fails with some probability p, independently, in every round. A detailed probabilistic analysis for "reasonable" choices of p revealed that our link failure model is the only one that guarantees a model coverage approaching 1 for $n \to \infty$; the coverage of all alternative approaches goes to 0.

In another companion paper [28], we analyzed the model coverage of both the exponential and the polynomial algorithms considered in this paper for substantial link failure rates in small systems. It turned out that the polynomial algorithms achieve an excellent coverage in almost every setting, and that even the exponential algorithms achieve a very decent model coverage in most cases.

8. Conclusions

In this paper, we introduced a comprehensive perception-based hybrid failure model for synchronous distributed algorithms, which covers both process and link failures. We showed that several well-known consensus and Byzantine agreement algorithms can be adapted to work correctly under this model, and analyzed their resilience and round complexity. The comparison with lower bound results for both classic process failures and link failures confirms/reveals that these bounds are tight, and shows a number of interesting trade-offs. Our findings reveal that the reliable communications assumption that has usually been employed in the analysis of such algorithms can actually be avoided.

[14] That is, $f_\ell^{sa} = f_\ell^{ra} = 0$ respectively $f_\ell^{sa} = f_\ell^{ra} = f_\ell$.

References

[1] N. Lynch, Distributed Algorithms, Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996.
[2] M. Raynal, Consensus in synchronous systems: A concise guided tour., in: Proceedings 9th Pacific Rim International Symposium on Dependable Computing, PRDC 2002, Tsukuba-City, Ibarski, Japan, 2002, pp. 221–228.
[3] K.J. Perry, S. Toueg, Distributed agreement in the presence of processor and communication faults, IEEE Transactions on Software Engineering SE-12 (3) (1986) 477–482.
[4] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Transactions on Programming Languages and Systems 4 (3) (1982) 382–401.
[5] F.J. Meyer, D.K. Pradhan, Consensus with dual failure modes, in: In Digest of Papers of the 17th International Symposium on Fault-Tolerant Computing, Pittsburgh, 1987, pp. 48–54.
[6] P.M. Thambidurai, Y.K. Park, Interactive consistency with multiple failure modes, in: Proceedings 7th Symposium on Reliable Distributed Systems, 1988, pp. 93–100.
[7] J.A. Garay, K.J. Perry, A continuum of failure models for distributed computing, in: Proceedings WDAG, 1992, pp. 153–165.
[8] P. Lincoln, J. Rushby, A formally verified algorithm for interactive consistency under a hybrid fault model, in: Proceedings Fault Tolerant Computing Symposium 23, Toulouse, France, 1993, pp. 402–411.
[9] C.J. Walter, N. Suri, The customizable fault/error model for dependable distributed systems, Theoretical Computer Science 290 (2002) 1223–1251.
[10] N. Santoro, P. Widmayer, Time is not a healer, in: Proc. 6th Annual Symposium on Theor. Aspects of Computer Science, STACS'89, in: LNCS, vol. 349, Springer-Verlag, Paderborn, Germany, 1989, pp. 304–313.
[11] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D.-W. Wang, L. Zuck, Reliable communication over unreliable channels, Journal of the ACM (JACM) 41 (6) (1994) 1267–1297.
[12] A. Basu, B. Charron-Bost, S. Toueg, Crash failures vs. crash + link failures, in: Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, ACM Press, Philadelphia, Pennsylvania, United States, 1996, p. 246.
[13] M.K. Aguilera, W. Chen, S. Toueg, Failure detection and consensus in the crash-recovery model, Distributed Computing 13 (2) (2000) 99–125. http://www.cs.cornell.edu/home/sam/FDpapers/crash-recovery-finaldcversion.ps.
[14] U. Schmid, B. Weiss, I. Keidar, Impossibility results and lower bounds for consensus under link failures, SIAM Journal on Computing 38 (5) (2009) 1912–1951.
[15] J.N. Gray, Notes on data base operating systems, in: G.S.R. Bayer, R.M. Graham (Eds.), Operating Systems: An Advanced Course, in: Lecture Notes in Computer Science, vol. 60, Springer, New York, 1978, p. 465. (Chapter 3.F).
[16] N. Santoro, P. Widmayer, Agreement in synchronous networks with ubiquitous faults, Theoretical Computer Sciences 384 (2–3) (2007) 232–249.
[17] R. Reischuk, A new solution for the Byzantine generals problem, Information and Control 64 (1–3) (1985) 23–42.
[18] S.S. Pinter, I. Shinahr, Distributed agreement in the presence of communication and process failures, in: Proceedings of the 14th IEEE Convention of Electrical & Electronics Engineers in Israel, 1985.
[19] H.M. Sayeed, M. Abu-Amara, H. Abu-Amara, Optimal asynchronous agreement and leader election algorithm for complete networks with Byzantine faulty links, Distributed Computing 9 (3) (1995) 147–156.
[20] H.-S. Siu, Y.-H. Chin, W.-P. Yang, Byzantine agreement in the presence of mixed faults on processors and links, IEEE Transactions on Parallel and Distributed Systems 9 (4) (1998) 335–345.
[21] B. Weiss, U. Schmid, Consensus with written messages under link faults, in: 20th Symposium on Reliable Distributed Systems, SRDS'01, New Orleans, LA, USA, 2001, pp. 194–197.
[22] U. Schmid, B. Weiss, J. Rushby, Formally verified Byzantine agreement in presence of link faults, in: 22nd International Conference on Distributed Computing Systems, ICDCS'02, Vienna, Austria, 2002, pp. 608–616.
[23] M. Biely, An optimal Byzantine agreement algorithm with arbitrary node and link failures, in: Proc. 15th Annual IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS'03, Marina Del Rey, California, USA, 2003, pp. 146–151.
[24] J. Rushby, Formal verification of hybrid Byzantine agreement under link faults, Tech. rep., Computer Science Laboratory, SRI International, Menlo Park, CA, 2001.
[25] E. Gafni, Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony, in: Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, ACM Press, Puerto Vallarta, Mexico, 1998, pp. 143–152.
[26] M. Biely, B. Charron-Bost, A. Gaillard, M. Hutle, A. Schiper, J. Widder, Tolerating corrupted communication, in: Proceedings of the 26th ACM Symposium on Principles of Distributed Computing, PODC'07, ACM, Portland, OR, USA, 2007, pp. 244–253.
[27] B. Charron-Bost, A. Schiper, The Heard-Of model: computing in distributed systems with benign faults, Distributed Computing 22 (1) (2009) 49–71.
[28] U. Schmid, Failure model coverage under transient link failures, Research Report 2/2004, Technische Universität Wien, Institut für Technische Informatik, Treitlstraße 3, A-1040 Vienna, Austria, 2004.
[29] T. Srikanth, S. Toueg, Simulating authenticated broadcasts to derive simple fault-tolerant algorithms, Distributed Computing 2 (1987) 80–94.
[30] L. Gong, P. Lincoln, J. Rushby, Byzantine agreement with authentication: observations and applications in tolerating hybrid and link faults, in: Proceedings Dependable Computing for Critical Applications-5, Champaign, IL, 1995, pp. 139–157.
[31] P. Berman, J.A. Garay, K.J. Perry, Asymptotically optimal distributed consensus, (A combination of results from ICALP'89, FOCS'89, and WDAG'91), 1992.
[32] J. Rushby, A formally verified algorithm for clock sychronization under a hybrid fault model, in: Proceedings ACM Principles of Distributed Computing, PODC'94, Los Angeles, CA, 1994, pp. 304–313.
[33] C.J. Walter, N. Suri, M.M. Hugue, Continual on-line diagnosis of hybrid faults, in: Proceedings DCCA-4, 1994, pp. 233–249.
[34] F. Cristian, C. Fetzer, Fault-tolerant internal clock synchronization, in: Proceedings of the Thirteenth Symposium on Reliable Distributed Systems, Dana Point, Ca., 1994, pp. 22–31.
[35] M.H. Azadmanesh, R.M. Kieckhafer, New hybrid fault models for asynchronous approximate agreement, IEEE Transactions on Computers 45 (4) (1996) 439–449.
[36] U. Schmid, Orthogonal accuracy clock synchronization, Chicago Journal of Theoretical Computer Science 2000 (3) (2000) 3–77.
[37] U. Schmid, K. Schossmaier, How to reconcile fault-tolerant interval intersection with the Lipschitz condition, Distributed Computing 14 (2) (2001) 101–111.
[38] M.H. Azadmanesh, R.M. Kieckhafer, Exploiting omissive faults in synchronous approximate agreement, IEEE Transactions on Computers 49 (10) (2000) 1031–1042.
[39] N. Santoro, P. Widmayer, Distributed function evaluation in the presence of transmission faults., in: SIGAL International Symposium on Algorithms, 1990, pp. 358–367.
[40] Z. Lipták, A. Nickelsen, Broadcasting in complete networks with dynamic edge faults, in: 4th International Conference on Principles of Distributed Systems, OPODIS'00, 2000, pp. 123–142.
[41] F. Cristian, H. Aghili, H.R. Strong, D. Dolev, Atomic broadcast: from simple message diffusion to byzantine agreement, Information and Computation 118 (1) (1995) 158–179.
[42] V. Hadzilacos, S. Toueg, Fault-tolerant broadcasts and related problems, in: Mullender S. (Ed.), Distributed Systems, 2nd Edition, Addison-Wesley, 1993, pp. 97–145 (Chapter 5).
[43] L. Lamport, Lower bounds on consensus, (unpublished manuscript, available via http://lamport.org, 2000).
[44] V. Hadzilacos, Connectivity requirements for Byzantine agreement under restricted types of failures, Distributed Computing 2 (1987) 95–103.
[45] G. Varghese, N.A. Lynch, A tradeoff between safety and liveness for randomized coordinated attack protocols, in: Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, 1992, pp. 241–250.

[46] U. Schmid, C. Fetzer, Randomized asynchronous consensus with imperfect communications, in: 22nd Symposium on Reliable Distributed Systems, SRDS'03, Florence, Italy, 2003, pp. 361–370.
[47] U. Schmid, C. Fetzer, Randomized asynchronous consensus with imperfect communications, Tech. Rep. 183/1-120, Department of Automation, Technische Universität Wien, (Extended version of [46]) (January 2002).
[48] D. Dolev, The Byzantine generals strike again, Journal of Algorithms 3 (1) (1982) 14–30.
[49] D. Powell, Failure mode assumptions and assumption coverage, in: Proc. 22nd IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-22, Boston, MA, USA, 1992, pp. 386–395, (Revised version available as LAAS-CNRS Research Report 91462, 1995).
[50] B. Charron-Bost, A. Schiper, Uniform consensus is harder than consensus, Journal of Algorithms 51 (1) (2004) 15–37. also published as Tech. Rep. DSC/2000/028, Ecole Polytechnique Fédérale de Lausanne.
[51] H. Attiya, J. Welch, Distributed Computing, 2nd Edition, John Wiley & Sons, 2004.
[52] M. Biely, U. Schmid, B. Weiss, Synchronous consensus under hybrid process and link failures, Research Report 58/2009, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-2, 1040 Vienna, Austria (2009).
[53] D. Dolev, H.R. Strong, Authenticated algorithms for byzantine agreement, SIAM Journal on Computing 12 (4) (1983) 656–666.
[54] B. Schneier, Applied Cryptography, 2nd Edition, John Wiley & Sons, 1996.
[55] D. Dolev, H.R. Strong, Polynomial algorithms for multiple processor agreement, in: Proceedings 14th Annual ACM Symposium on Theory of Computing, STOC'82, San Francisco, 1982, pp. 401–407.
[56] U. Schmid, How to model link failures: A perception-based fault model, in: Proceedings of the International Conference on Dependable Systems and Networks, DSN'01, Göteborg, Sweden, 2001, pp. 57–66.
[57] U. Schmid, B. Weiss, Synchronous Byzantine agreement under hybrid process and link failures, Tech. Rep. 183/1-124, Department of Automation, Technische Universität Wien (Nov. 2002).