Technical Report 183/1-124, Dept. of Automation, TU Vienna, October 2002 April 25, 2003

Synchronous Byzantine Agreement under Hybrid Process and Link Failures

ULRICH SCHMID
and
BETTINA WEISS
Technische Universität Wien

This paper shows that deterministic consensus in synchronous distributed systems with link failures is possible, despite the impossibility result [Gray 1978]. Instead of using randomization, we overcome this impossibility result by moderately restricting the inconsistency that link failures may cause system-wide. Relying upon a novel hybrid failure model that provides different classes of failures for both processes and links in a round-by-round fashion, we prove that the $m+1$-round Byzantine agreement algorithms OMH [Lincoln and Rushby 1993] and its authenticated variants OMHA, ZA [Gong et al. 1995] require

$$\begin{aligned} n &> 3f_\ell + f_\ell^a + 2(f_a + f_s) + f_o + f_m + m \\ n &> 3f_\ell + 2(f_a + f_s) + f_o + f_m + m \\ n &> 2f_\ell + f_a + f_s + f_o + f_m + 1 \end{aligned}$$

processes for transparently masking at most f_ℓ link failures (including at most f_ℓ^a arbitrary ones) per process in each round, in addition to at most f_a, f_s, f_o, f_m arbitrary, symmetric, omission, and manifest process failures, provided that $m \geq f_a + f_o + 1$. If authentication fails, we show that OMHA degrades to OMH, whereas ZA can be made tolerant to broken signatures by increasing f_a accordingly. A uniform variant of OMH is also proposed, which guarantees validity and agreement even on benign faulty processes. These specific results are complemented by a systematic theoretical study of consensus under link failures. We provide impossibility results and (tight) lower bounds for the required number of processes and rounds, as well as a precise characterization of what makes a process failure Byzantine resp. omissive. Moreover, we explore the applicability of our approach to systems with incomplete communication graphs. Finally, an analysis of the assumption coverage in systems where links fail independently is provided, which reveals that the probability of violating the failure model can be made arbitrarily small by sufficiently increasing n.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; C.4 [**Performance of Systems**]: Fault tolerance and Modeling techniques; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and concurrency*

General Terms: Algorithms, Performance, Reliability, Security, Theory

Additional Key Words and Phrases: Assumption coverage, authentication, Byzantine agreement, failure models, fault-tolerant distributed systems, impossibility results, link failures, lower bounds, uniform consensus

---

Authors address: Technische Universität Wien, Department of Automation, Treitlstrasse 1, A-1040 Vienna. Email: {s, bw}@auto.tuwien.ac.at

This research is part of our W2F-project, which targets a wireline/wireless fieldbus based upon spread-spectrum CDMA communications, see *http://www.auto.tuwien.ac.at/Projects/W2F/* for details. W2F is supported by the Austrian START programme Y41-MAT.

1. OVERVIEW

Although process failure models, like the one that at most f of n processes in a distributed system may be faulty during a particular execution, have always been applied most successfully in the analysis of fault-tolerant distributed algorithms, they do have limitations. In fact, given the steadily increasing dominance of communication over computation in modern distributed systems, it becomes increasingly difficult to apply failure models that capture only process failures.

Indeed, 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 can cause any data packet to be lost or even modified (without being detected). The resulting failure, however, cannot reasonably be attributed to the innocent sender process. Declaring the receiver process as faulty would be overly conservative either, since a packet error does not usually imply a process failure. After all, the—fault-tolerant—algorithm is executed correctly. Link failures should hence be a category of their own in a more realistic failure model.

The first contribution of our paper is a suitable failure model for synchronous systems with high link failure rates, which distinguishes several types of process and link failures. It is a round-by-round model [Gafni 1998] that can even be applied to systems with incomplete communication graphs. Belonging to the class of hybrid failure models, it allows maximum resilience under real operating conditions and is hence particularly beneficial for small systems. By analyzing the assumption coverage in systems where individual links fail independently, we show that our model can reasonably be applied even to communications-expensive algorithms in wireless systems, where link loss probabilities up to $p = 10^{-2}$ are common.

Reaching consensus ("Byzantine agreement") among the processes of a distributed system is widely recognized as one of the most fundamental problems in fault-tolerant distributed computing [Lamport et al. 1982]. Unfortunately, there is a discouraging impossibility result for deterministic consensus in presence of link failures (see Section 4.4 for its proof), which goes back to Gray's 1978 paper [Gray 1978] on atomic commitment in distributed databases:

THEOREM 1.1 GRAY'S IMPOSSIBILITY [LYNCH 1996, THM. 5.1]. *There is no deterministic algorithm that solves the coordinated attack problem in a synchronous two-process system with lossy links.*

Due to this result, almost all deterministic algorithms for consensus developed during the past 20+ years considered process failures only.[1] To be able to employ deterministic algorithms in systems with high link failure rates, one might ask, however, whether some of the pivotal assumptions of the impossibility result could be relaxed.

[1]Just compare Chapter 5 ("Distributed Consensus With Link Failures") in Lynch's book [Lynch 1996] with Chapter 6 ("Distributed Consensus With Process Failures") ...

The present paper is the first one[2] to show that this can indeed be done: As the second major contribution of our paper, we provide a comprehensive theoretical study of consensus under link failures. It reveals that, if the power of link failures is slightly restricted with respect to the inconsistencies caused system-wide, consensus can be solved despite of a large number of link failures if the number of processes n is moderately increased. Using new instances of bivalency and "easy impossibility proof" techniques, we provide 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 upon a generalization of Theorem 1.1, which stresses the importance of unimpaired bidirectional communication for solving consensus. Our results also allow us to precisely characterize what makes a process failure actually Byzantine resp. omissive.

As the third major contribution of our paper, we prove the correctness of all known hybrid *oral* and *written* messages algorithms[3] for Byzantine agreement under our failure model; detailed formulas for the required number of processes and rounds are also established. Moreover, we provide an algorithm that guarantees uniform agreement. Our results for written messages algorithms reveal that authentication is beneficial with respect to both process and link failures, and that processes with broken/disclosed signatures can be accounted for by increasing the number of arbitrary process failures appropriately.

The remaining sections of our paper are organized as follows:

—*Section 2:* Overview of related work.

—*Section 3:* Introduction and discussion of our hybrid failure model, both in complete and incomplete communication graphs.

—*Section 4:* Development of theoretical results, including tight lower bounds on the required number of processes (Subsection 4.2), required number of rounds (Subsection 4.3), and a precise characterization of Byzantine and omission process failures (Subsection 4.4).

—*Section 5:* Analysis of the *Hybrid Oral Messages* algorithm OMH (Subsection 5.1), and its uniform variant OMHU (Subsection 5.2).

—*Section 6:* Introduction of authentication issues for *Hybrid Written Messages* algorithms.

—*Section 7:* Analysis of the authenticated algorithms OMHA (Subsection 7.1), ZA (Subsection 7.2), and ZAr (Subsection 7.3), as well as application in systems with a broadcast network (Subsection 7.4).

—*Section 8:* Analysis of OMH's assumption coverage in systems with transient link failures.

[2]Part of our work has been presented—in preliminary form, and usually without proofs—in [Schmid et al. 2002] and [Weiss and Schmid 2001] (short paper). A formal verification of some of our proofs can be found in the SRI technical report [Rushby 2001].

[3]Note that we are aware of the fact that those algorithms suffer from an exponential number of messages. Given that they are hybrid instances of the most well-researched algorithm [Lamport et al. 1982] for Byzantine agreement, however, they are certainly the most suitable candidates for introducing our fairly general approach. More efficient (polynomial) consensus algorithms are treated in [Biely and Schmid 2001].

—*Section 9:* Summary and discussion of our accomplishments (Subsection 9.1 and 9.2) and some directions of further research (Subsection 9.3).

2. RELATED WORK

There are a number of *hybrid failure models* in the literature [Meyer and Pradhan 1987; Thambidurai and Park 1988; Lincoln and Rushby 1993; Rushby 1994; Walter et al. 1994; Cristian and Fetzer 1994; Azadmanesh and Kieckhafer 1996; Schmid 2000; Siu et al. 1998; Schmid and Schossmaier 2001; Walter and Suri 2002; Azadmanesh and Kieckhafer 2000], which differ primarily in the classes of process failures considered: Early models like [Thambidurai and Park 1988] distinguish only manifest and Byzantine failures, whereas fully-fledged models like the ones of [Walter and Suri 2002] and [Azadmanesh and Kieckhafer 2000] 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[4] in the literature that deal with link failures at all. One obvious approach is to simply consider link failures as (sender) process failures, as in [Gong et al. 1995], for example. Still, allowing every process in the system to commit at most f_ℓ receive link failures, as in our model, would cause many failure patterns where all $f = n$ processes must be considered faulty; Figure 1 shows an example for $n = 4$ and $f_\ell = 1$. This (falsely) suggests that problems like clock synchronization and consensus are not solvable in presence of link failures.

A similar argument applies to the more detailed send/receive-omission failure model of [Perry and Toueg 1986], where receive omissions are mapped to receiver process failures. Although only the number of send omission faulty processes (and not the receive omission faulty ones) needs to be counted in f, correctness properties have only been established for processes that do not commit either type of failure. Hence, in the example of Figure 1, no process would remain that could be considered correct.

Declaring a process as faulty due to receive omissions is overly conservative, however: Absent—even faulty—messages do not necessarily cause a fault-tolerant algorithm to fail. For example, existing work on *uniform* consensus algorithms [Charron-Bost and Schiper 2000] shows that validity and agreement can be guaranteed for receive omission faulty processes as well, provided that more than $n/2$ processes are correct. Still, only specific processes (namely, the less than $n/2$ omission

[4]Note that it is relatively easy to handle 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 [Afek et al. 1994; Basu et al. 1996; Aguilera et al. 2000]. Such time redundancy techniques cannot be used in synchronous systems, however, without unduly increasing the duration of the rounds according to the maximum number of successive message losses that are to be tolerated. In sharp contrast, our approach uses resource (process) redundancy only and therefore does not suffer from this problem. A detailed survey of link failures in partially synchronous and asynchronous systems may be found in [Schmid and Fetzer 2002].

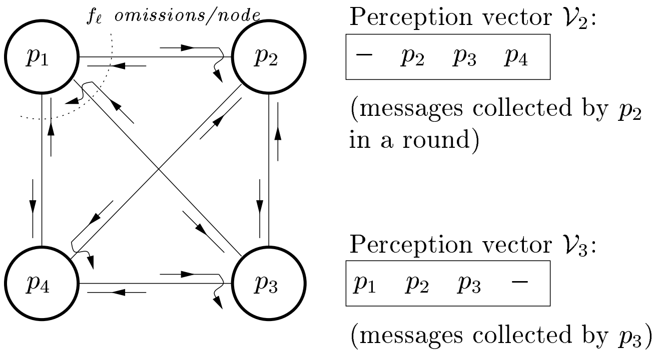

Fig. 1. *Example of a 4-process system with $f_\ell = 1$ receive failures per process in each round, where all processes must be considered faulty in traditional process-centric failure models.*

faulty ones, but no correct one) may experience link failures here. This assumption is dropped in the mobile failure model [Santoro and Widmayer 1989], where at most one arbitrary process may suffer from any number of link failures in any round. Consensus cannot be solved in this model, however.

Another class of models [Pinter and Shinahr 1985; Sayeed et al. 1995; Siu et al. 1998] 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. Leader election algorithms can even tolerate up to $\mathcal{O}(n)$ link failures *per process* [Abu-Amara and Lokre 1994; Singh 1996]. Still, none of the above models can deal with the fact that (transient) link failures usually hit different links in different rounds of the execution. Since failures are usually considered persistent during an execution, the "exhaustion" of non-faulty processes and links would progress rapidly with every round, which makes any attempt to solve consensus in models like [Gong et al. 1995; Perry and Toueg 1986; Hadzilacos 1987] even more hopeless. Link failures must hence be handled in a round-by-round fashion, similar to the idea underlying round-by-round fault detectors [Gafni 1998].

The model introduced in [Reischuk 1985] can be seen as a first step in this direction. For a system with $n \geq 20f + 1$ processes, at most f of those being Byzantine faulty during a single round, a consensus algorithm was given that tolerates a small number $l = n/20$ of link failures at every node. The only work that provides a link failure tolerance compareable to ours—without making this explicit, however—is [Cristian et al. 1985], which describes a suite of synchronous atomic broadcast protocols. Although reliable/atomic broadcasting is usually investigated in a more communication-oriented context [Hadzilacos and Toueg 1993], it obviously solves Byzantine agreement as well. The three algorithms of [Cristian et al. 1985] 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; [Hadzilacos 1987] explores the connecticity requirements associated with such problems. Since our failure model (see Theorem 3.2) respects the requirements of [Cristian et al. 1985], we can compare our results in some detail in Section 7.3.

Finally, we note that link failures have been considered in *randomized* consensus algorithms like the one of [Varghese and Lynch 1992]. Such algorithms circumvent the impossibility result of Theorem 1.1 by adding non-determinism (coin tossing) to the computations. Still, due to the inherent non-zero probability of failure/non-termination within a fixed number of rounds, randomized algorithms are unsuitable for some applications. Moreover, there is a lower bound $1/(R+1)$ for the probability of disagreement after R rounds [Lynch 1996, Thm. 5.5]. Note, however, that we recently discovered in [Schmid and Fetzer 2002] that our approach of modeling link failures allows to circumvent this lower bound and achieves a probability of disagreement of only $(1/2)^R$.

3. THE PERCEPTION-BASED HYBRID FAILURE MODEL

We consider a distributed system of n *nodes*, which execute one or more *processes* that communicate via a fully connected point-to-point network. In case of $c > 1$ processes per node, it is assumed that every such process has a unique peer at every other node. The entire execution can hence be viewed as the execution of c concurrent systems of n processes here. All processes executed by a non-faulty node must be non-faulty; an arbitrary number of processes executed by a faulty node may be faulty as specified.[5] All links between processes are bidirectional, consisting of two unidirectional channels that may be hit by failures independently.

The entire system is synchronous in the usual sense, that is, the distributed computation evolves in a series of rounds at all processes in lock-step. In every round, all processes (or a subset of those) "broadcast" a single value to each other; the received values are used to compute the value to be broadcast in the next round. Broadcasting just means non-atomic, non-reliable sending of the same message to all receivers (including the transmitter) here. In case of failures, inconsistent reception of a broadcast message may occur. It is the purpose of the failure model to cleanly specify the failures that are to be tolerated by the distributed algorithm in question.

The *perception-based hybrid failure model* introduced below is a generalization of the failure model used for the analysis of clock synchronization and other single-round agreement algorithms in [Schmid 2001]. In this model, the global, i.e., system-wide, number of failures is replaced by the number of failures that are observable in the processes' local "perceptions" of the system. Formally, process r's *perception vector*

$$\mathcal{V}_r = (V_r^1, V_r^2, \ldots, V_r^n) \tag{1}$$

of any specific round R is considered (we suppress the round number R for clarity), where every *perception* $V_r^s \in \mathcal{V}_r$ represents the message process r received from its peer process s in round R; type and value(s) depend upon the particular algorithm considered. For approximate agreement algorithms, for example, the V_r^s would be real values that represent the receiver's opinion about the sender's local value V^s.

[5]The terms process and node failures can hence be used more or less synonymously. We usually prefer the more common term process failure in this paper.

In case of the clock synchronization algorithm analyzed in [Schmid 2001], we found it sufficient to just impose a bound upon the maximum number of failures in any *pair* of perception vectors $\{\mathcal{V}_p, \mathcal{V}_q\}$, i.e., p's and q's lines in the "matrix" of perceptions on the right-hand side of

$$\begin{aligned} \mathcal{V}_1 &= (V_1^1, V_1^2, \ldots, V_1^n) \\ \mathcal{V}_2 &= (V_2^1, V_2^2, \ldots, V_2^n) \\ &\vdots \\ \mathcal{V}_n &= (V_n^1, V_n^2, \ldots, V_n^n). \end{aligned}$$

For example, if every process may lose at most $f_\ell = 1$ messages due to faulty links (see Figure 1 in Section 2), any two non-faulty processes' perception vectors can differ only in at most $2f_\ell = 2$ perceptions, namely, the ones where either receiver process experienced its omission. Moreover, only at most $f_\ell = 1$ of the non-faulty perceptions present at some non-faulty process can be missing or faulty at any other non-faulty process. Even more, since f process failures can produce at most f faulty perceptions in any $\mathcal{V}_r$, our perception-based model will be compatible with traditional process failure models. Hence, all existing lower bound and impossibility results remain valid.

Depending upon the type of failure of a perception, e.g., missing or erroneous value, several different classes of failures (manifest/omission/symmetric/asymmetric) can be distinguished. This leads to a *hybrid failure model*, see [Azadmanesh and Kieckhafer 2000] for an overview, which exploits the fact that less severe failures can be usually handled with fewer processes than more severe ones. In case of consensus, for example, masking f symmetric failures usually requires only $n \geq 2f+1$ processes, whereas $n \geq 3f+1$ is needed if all failures are asymmetric (Byzantine) ones. Since a large number of asymmetric failures is quite unlikely in practice, this effectively leads to a smaller n for tolerating a given number of failures. This, in turn, positively affects dependability by reducing the number n of components that could be faulty, cf. [Powell 1992]. System designers will hence appreciate our very detailed hybrid failure model for getting the maximum fault-tolerance out of a given—and usually quite small—n. Obviously, an algorithm's resilience in standard models (like all-Byzantine) is easily[6] obtained by setting some model parameters to 0.

For Srikanth & Toueg's consistent broadcasting primitive [Srikanth and Toueg 1987] in asynchronous systems, for example, our analysis in [Schmid 2001] revealed that

$$n \geq 4f_{ia} + 3f_a + 2(f_s + f_{is} + f_o + f_{io}) + f_m + 1$$

processes are sufficient for tolerating at most f_m, f_o, f_s, f_a crash, omission, symmetric, and arbitrary process failures and at most f_{io}, f_{is}, and f_{ia} omission, symmetric, and arbitrary link failures at each receiver process.

In view of the results of [Schmid 2001], it was natural to consider the question of whether a perception-based failure model can also be used to attack deterministic

[6]Note, however, that the general hybrid analysis might be too conservative for certain restricted cases, see Remark 4 on Theorem 5.4 for an example.

consensus in presence of link failures. More specifically, as the general problem is unsolvable by Theorem 1.1, one might ask whether there is a meaningful restriction of the power of link failures that can be expressed in a perception-based manner. And indeed, as will be shown in the subsequent sections, there is a suitable perception-based failure model that allows even consensus with oral messages [Lamport et al. 1982] in systems with high link failure rates. Ignoring process failures for the moment, it is based upon constraining two quantities in the matrix of perceptions of any single round:

($A1^r$) Any receiver process may encounter at most f_ℓ^r receive link failures on its in-bound links, without being considered faulty. Hence, there may be at most f_ℓ^r columns with erroneous perceptions in any fixed single line. This puts a limit upon the number of sender processes that may appear faulty to a single receiver (already employed in [Schmid 2001]). Figure 2 shows an example with $f_\ell^r = 2$.

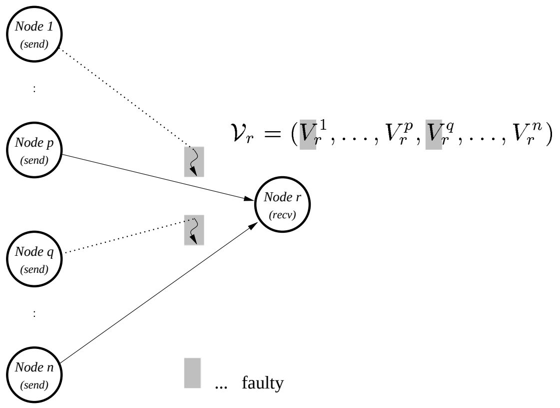

Fig. 2. *Example of a process r that suffers from two receive link failures in a round, hitting the links from sender processes 1 and q.*

($A1^s$) Every sender process may commit at most f_ℓ^s send link failures (also termed broadcast link failures) on its out-bound links, without being considered faulty. Hence, there may be at most f_ℓ^s lines with erroneous perceptions in any fixed single column. This puts a limit upon the number of receiving processes that may obtain a wrong (or no) message in the broadcast of a single sender. Figure 3 shows an example with $f_\ell^s = 2$.

Note carefully that we allow every process in the system to commit up to f_ℓ^s broadcast and up to f_ℓ^r receive link failures in every round, without considering the process as faulty in the usual sense. In addition, the particular links actually hit by a link failure may be different in different rounds. A process must be considered (omission) faulty, however, if it exceeds its budget f_ℓ^s of broadcast link failures in some round. Note that a process that experiences more than f_ℓ^r receive link failures in some round must usually be considered (arbitrary) faulty, since it might be unable to correctly follow the algorithm after such an event.

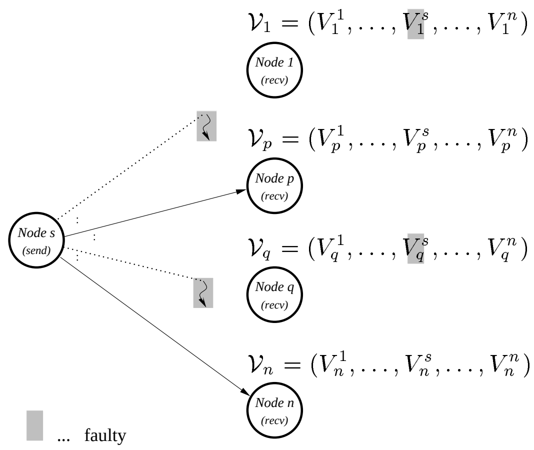

Fig. 3. *Example of a process s that produces two send link failures, hitting the links to receiver processes 1 and q.*

The above distinction makes sense due to the fact that we consider the unidirectional channels, rather than the bidirectional links, as single fault containment regions: Broadcast link failures affect outbound channels, whereas receive link failures affect inbound channels. Still, broadcast and receive link failures are of course not independent of each other: If a message from process p to q is hit by a failure in p's message broadcast, it obviously contributes a failure in process q's message reception as well.

Nevertheless, our failure model considers ($\mathrm{A1}^s$) and ($\mathrm{A1}^r$) as independent of each other and of process failures, for any process in the system and any broadcast/reception. Only the model parameters f_ℓ^s and f_ℓ^r do depend on each other: According to ($\mathrm{A1}^s$), at most $n \cdot f_\ell^s$ messages may be faulty system-wide in any round's broadcasts. By ($\mathrm{A1}^r$), they must be spread over all message receptions in a way that no process experiences more than f_ℓ^r faulty messages. This is only possible if $r_l = n \cdot f_\ell^r = n \cdot f_\ell^s = s_l$, however: If $s_l > r_l$ and hence $f_\ell^s > f_\ell^r$, then there would be at least one receiver that gets more than f_ℓ^r faulty messages and thus violates ($\mathrm{A1}^r$) if all senders generated their maximum number f_ℓ^s of send link failures. Similarly, if $s_l < r_l$ and hence $f_\ell^s < f_\ell^r$, there would be at least one sender process with more than f_ℓ^s broadcast link failures if all receivers experienced their maximum f_ℓ^r of receive link failures, thereby violating ($\mathrm{A1}^s$).

Consequently, since both ($\mathrm{A1}^s$) and ($\mathrm{A1}^r$) must[7] hold in any execution, there are only two possibilities: (1) $f_\ell^s = f_\ell^r = f_\ell$, in which case ($\mathrm{A1}^r$) and ($\mathrm{A1}^s$) always hold, or (2) $f_\ell^s \neq f_\ell^r$, in which case the failure model only applies if link failure patterns are restricted to feasible ones by some additional assumptions. An example of (2) is the alternative interpretation of restricted process failures as link failures introduced in Section 4.4.

[7]Note carefully, however, that there are algorithms that do not need ($\mathrm{A1}^s$) and ($\mathrm{A1}^r$) simultaneously in any single round, but only one of those, see Remark 5 on Theorem 5.4.

To align the present paper with the existing literature, the above perception-based model is recast into a simple modification of the original oral messages assumption of [Lamport et al. 1982]. Process failures are modeled according to a generalized version of the hybrid failure model of [Lincoln and Rushby 1993; Walter et al. 1997]. Our contribution here is to add the important class of *omission process failures* to the originally provided manifest, symmetric, and arbitrary failures. An omission faulty process p may fail to send a message to any subset $\mathcal{R}_p$ of its receivers. In sharp contrast to send link failures in Figure 3, $|\mathcal{R}_p|$ may be larger than f_ℓ^s here. By adding the seperate class of omission failures to our failure model, those frequently encountered failures [Azadmanesh and Kieckhafer 2000] need not be counted as arbitrary, which further decreases the required number of processes n (but see Remark 4 on Theorem 5.4).

In order to cleanly specify the semantics of restricted failures, we will need the notion of *obedient* processes: An obedient process is an operational (= not crashed) process that gets its inputs and faithfully executes the particular algorithm like a non-faulty process. Unlike a non-faulty process, however, it may fail in specific ways to communicate its value to the outside world. For example, an obedient but not non-faulty process could omit to send its (correctly computed) value to some subset of its receivers. Note that Definition 3.1 given below will actually restrict the meaning of obedient to non-faulty or manifest faulty or omission faulty (but not crashed).

We will subsequently use the term obedient instead of non-faulty whenever a process is considered from a receiver's perspective only. Note carefully, however, that a generalization from non-faulty to obedient processes may not always work, cp. algorithms OMH vs. OMHU in Section 5 for an example.

Definition 3.1 System Model. *We consider a synchronous distributed system consisting of* n *nodes executing one or more processes, which are interconnected by a fully connected point-to-point network made up of pairs of unidirectional channels.*

(P1) In any round, there may be at most f_a, f_s, f_o, *and* f_m *arbitrary, symmetric, omission, and manifest faulty nodes. The failure modes of their processes are defined via the set* $rvals(V_p, p)$ *of admissible values delivered by obedient receivers when process* p *attempts to send them the value* V_p*:*

—*A* manifest *faulty process* p *fails to send a message, or sends an obviously bad value, to any receiver. All obedient receivers* q *in the system (including* p *itself) deliver the distinguished value* $V_q^p = \emptyset$ *in this case, i.e.,* $rvals(V_p, p) = \{\emptyset\}$.

—*An* omission *faulty process* p *may fail to send the correct value* V_p *to some of its obedient receivers* q_i, *which deliver* $V_{q_i}^p = \emptyset$ *instead of* V_p *in this case. Hence,* $rvals(V_p, p) = \{V_p, \emptyset\}$.

—*A* symmetric *faulty process* p *sends the same wrong (but not usually obviously bad) value* X_p *to every receiver* q. *All obedient receivers (including* p *itself) deliver* $V_q^p = X_p$—*the value* "actually sent"—*in this case, such that* $rvals(V_p, p) = \{X_p\}$.

—*An* arbitrary (asymmetric) *faulty process may inconsistently send any value to any receiver, so* $rvals(V_p, p)$ *is the set of all possible values, including* $\emptyset$.

A process that is manifest or omission faulty is called benign *faulty and is as-*

sumed to be obedient (if not crashed). A process is consistent *if it is either non-faulty, manifest faulty, or symmetric faulty.*

(A1^s) If a single [faulty or non-faulty] process p sends a value V_p to some set of obedient receiver processes $q_i \in \mathcal{R}$ in a round, at most f_ℓ^s of the delivered values $V_{q_i}^p$ may differ from the admissible receive values in $rvals(V_p, p)$. Let $f_\ell^{sa} \le f_\ell^s$ be the maximum number of non-omissive, i.e., non-empty and hence value faulty, $V_{q_i}^p$ among those.

(A1^r) If all processes $p_i \in \mathcal{S}$ of a set of [faulty or non-faulty] processes send a message containing V_{p_i} to some obedient receiver process q in a round, at most f_ℓ^r of the delivered values $V_q^{p_i}$ may differ from the admissible receive values in $rvals(V_{p_i}, p_i)$. Let $f_\ell^{ra} \le f_\ell^r$ be the maximum number of non-omissive, i.e., non-empty and hence value faulty, $V_q^{p_i}$ among those.

(A2) The receiver of a message knows who sent it.

(A3) The absence of a message from sender p can be detected at any receiver q at the end of a round, which leads to $V_q^p = \emptyset$.

Remarks:

(1) A single round consists of (1) all the processes' local computations based upon the values received in the previous round, (2) the broadcasts (= successive sends) of the resulting messages according to (A1^s), and (3) the reception of those messages according to (A1^r).

(2) Properties (A1^s) and (A1^r) must hold simultaneously in any execution and are assumed to be independent of each other. Any process has an individual "budget" f_ℓ^r (resp. f_ℓ^s) of link failures that may hit arbitrary inbound (resp. outbound) links. Without restricting link failure patterns, however, this can only be guaranteed if $f_\ell^s = f_\ell^r$ and $f_\ell^{sa} = f_\ell^{ra}$.

(3) The system model of Definition 3.1 considers process and link failures independently. Therefore, even a manifest or omission faulty process's broadcast could generate erroneous values at f_ℓ^{sa} receivers, for example. By contrast, the original model in [Schmid and Weiss 2001, Def. 1] assumed that link failures hit only messages from non-faulty senders/receivers. The new model is more natural and has a better coverage in real systems, but requires a slightly more involved analysis.

(4) For generality, Definition 3.1 has been specified as a round-by-round model also with respect to process failures. Hence, the f_a, f_s, f_o or f_m faulty nodes might even change from round to round. Typically, however, it will be assumed in our analysis that the set of faulty processors is the same in the entire execution. In that case, faulty processes must not change their failure mode, i.e., must be counted in f_a, f_s, f_o or f_m according to their most severe behavior. Hence, a node that hosts one or more processes that e.g. behave symmetric faulty in some round and omission or manifest faulty in some round must be considered arbitrary faulty.

(5) Our manifest failures differ from the systemwide detectable ones of [Powell 1992] and the "benign failures" of [Azadmanesh and Kieckhafer 2000; Walter and Suri 2002] by also including symmetric omissions (produced e.g. by clean crashes), where no receiver gets a non-$\emptyset$ value. Since a receiver does not know locally whether a message is missing due to a symmetric omission or e.g. a Byzantine

failure, symmetric omissions are usually more difficult to handle than "benign failures". This is not true for the algorithms analyzed in this paper, however, so integrating both into manifest failures as in [Lincoln and Rushby 1993] makes sense.

(6) Our failure model provides only *strong* manifest failures, where all receivers, including the sender itself, get $\emptyset$-values only. Some related work like [Rushby 2001] 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. Alternatively, they are equivalent to strong manifest failures if at least one arbitrary link failure may also occur (i.e., if $f_\ell^{ra} = f_\ell^{sa} \geq 1$): Self-delivery of the correct (even incorrect) value sent by a manifest faulty process can then be explained by a spurious message generated on the link to itself.

(7) 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. Crash failures are in fact more severe than our manifest failures but weaker than omission failures [Perry and Toueg 1986]. Nevertheless, both manifest and omission failures are more severe than crashes in that faulty processes may resume correct operation in any later round. Bear in mind, however, that this behavior is not equivalent to the crash-recovery model of [Aguilera et al. 2000], since our processes may not lose state but must continuously follow the algorithm, see Remark 8 below.

(8) Benign faulty processes, i.e., manifest and omission faulty ones, are allowed to convey either the correct value or else $\emptyset$ to their receivers, which implies that they must know the correct value at least 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 (or have crashed).

(9) 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.

Since the failure model of Definition 3.1 allows link failures to be both transient and permanent, it is possible to model incomplete communication graphs by considering missing links as faulty, cp. [Siu et al. 1998]. The following Theorem 3.2 will show, however, that sparsely connected graphs and, in particular, partitioned ones are disallowed here.

Recall from elementary graph theory that a graph G is *c-connected* if it remains connected by removing at most c processes and their adjacent edges. Two paths connecting processes p and q are *process-disjoint* iff they do not have common processes except p and q.

THEOREM 3.2 CONNECTIVITY. *The communication graph of any system of $n > f_\ell^s + f_\ell^r$ processes complying to the system model of Definition 3.1 is c-connected with $c = n - f_\ell^s - f_\ell^r > 0$. Moreover, any pair of processes p, q is connected by c process-disjoint paths consisting of at most 2 non-faulty links.*

Proof: We use Menger's theorem, which says that G is c-connected iff any pair of processes p, q is connected by c process-disjoint paths. To show the latter, we argue as follows: From ($\mathrm{A1}^s$) in Definition 3.1, we know that p is connected to at least $n - f_\ell^s$ processes (possibly including itself) via non-faulty links. From ($\mathrm{A1}^r$), it follows that q is connected to a set of at least $n - f_\ell^s - f_\ell^r = c$ of these processes via non-faulty links. Let $\mathcal{I}$ with $|\mathcal{I}| \geq c$ be this set of processes. If $p \notin \mathcal{I}$ and $q \notin \mathcal{I}$, then p and q are connected by c paths consisting of 2 non-faulty links routed over the processes in $\mathcal{I}$. Otherwise, there are only $c - 1$ paths of length 2 and a direct path from p to q, which are of course also process disjoint. □

4. THEORETICAL FRAMEWORK

In this section, we will develop a theoretical framework for consensus[8] in presence of link failures. We restrict our attention to a synchronous distributed system of n processes according to Definition 3.1 here, where only link failures but no process failures are considered ($f_a = f_s = f_o = f_m = 0$).

Our major tool will be a generalization of the well-known Theorem 1.1 [Gray 1978], which reveals the importance of unimpaired *bidirectional* communication for solving consensus. It allows to derive (tight) lower bounds on the number of processes required for solving consensus, both for the case of pure omission link failures and arbitrary ones. We also show that solving consensus in presence of link failures requires one additional round. A precise characterization of what actually makes a process failure Byzantine (resp. omissive)—and thus requires even more rounds to be executed by a consensus algorithm—eventually concludes this section.

Binary consensus is the problem of computing a common binary output value from binary input values distributed among all processes. We assume that every process p provides an *input value* $x_p \in \{0, 1\}$, which is supplied to the local instance of a distributed consensus algorithm that starts simultaneously at all processes. Within a finite number of rounds (that may be different for different processes, and may even depend upon the particular execution), p irreversibly computes ("decides upon") an *output value* $y_p \in \{0, 1\}$, which must satisfy the following properties:

(C1) (*Agreement*): Every two processes p and q compute the same output value $y_p = y_q$.

(C2) (*Validity*): If all processes start with the same input value, then every process p computes
— $y_p = 1$ if $\forall q : x_q = 1$ and no link failure has occured in the entire execution,
— $y_p = 0$ if $\forall q : x_q = 0$.

Note that practical consensus algorithms usually guarantee a stronger validity property, where, in case of $\forall q : x_q = 1$, every process p computes $y_p = 1$ even when link failures have occured. Considering a weaker form of validity in this theory section makes sense, however, since impossibility of consensus under (C2) obviously implies imposibility of consensus under any stronger validity property as well.

[8]Since Byzantine agreement can be used to implement consensus, impossibility results and lower bounds derived for consensus carry over to Byzantine agreement as well, see our discussion at the beginning of Section 5.1 and Remark 1 on Definition 5.1.

4.1 Basic Results

We start with the well-known proof of Gray's Theorem 1.1 in the formalization of [Lynch 1996, Thm. 5.1], which uses the following argument: Suppose that the failure-free execution $\overline{E}$ of a two-process system with omission faulty links terminates at the end of round r when starting with initial values $[1, 1]$. By validity, the common decision value must be 1 in $\overline{E}$. Since decisions are irreversible, we can safely drop all the messages some algorithm might sent in rounds $> r$ without changing the decision value. The resulting "truncated" execution E shown in Figure 4 is obviously feasible.

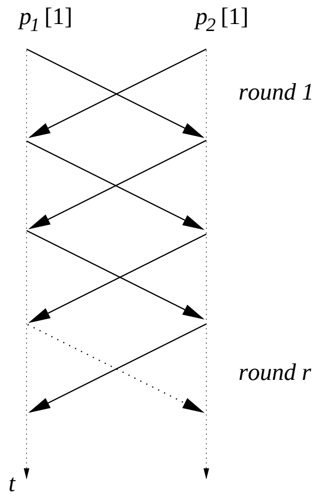

Fig. 4. *Execution of a two-process synchronous consensus algorithm with link omission failures, starting with initial values* $[1, 1]$*, truncated after round r by which both processes decide.*

If we now drop the last (dotted) round r-message from $p_1 \to p_2$ in E, the resulting execution E' is indistinguishable for p_1, so p_1 and, by agreement, also p_2 must eventually decide on 1 as well. Note carefully, however, that p_2 could decide later now, i.e., in some round $r' > r$, in E' — we can only guarantee that the decision value is the same. An analogous argument reveals that we can also drop the round r-message from $p_2 \to p_1$ in E' without changing the decision value. Note carefully that, since p_2 could decide in round $r' > r$ here, the initial "truncation" made it impossible for p_1 to tell p_2 about the lost message in some later round $r + 1, \ldots, r'$.

The above procedure can be repeated until all the messages in all rounds have been dropped, without changing the decision value. Since the processes are now fully isolated from each other in the final execution, changing the initial values to $[1, 0]$ and then to $[0, 0]$ cannot affect the decision value either, but now the outcome of the final execution would violate validity.

As our first result, we will now show that solving consensus is even impossible when a link—viewed as a pair of unidirectional links—loses or, in case of arbitrary link failures, corrupts messages only in one direction, i.e., when either process (but not necessarily both) can withold its information for an arbitrary number of rounds. Eventually bidirectional communication is hence mandatory for any deterministic consensus algorithm. Solutions exist, however, if the direction of the message loss is fixed, see the remarks following Theorem 4.3 below.

Unfortunately, this stronger result cannot be shown by generalizing the proof of Theorem 1.1: We are not allowed to simply drop all messages in later rounds to "hide" the effect of dropping/restoring round r-messages here, since this would amount to a link failure in both directions and hence an infeasible execution. We found out, however, that bivalency-type proofs [Fischer et al. 1985] are a powerful tool in our setting as well (which could be employed in an alternative proof of Theorem 1.1 as well).

In what follows, we use the following notation: A configuration $C^r = (c_1^r, c_2^r)$ of our 2-process system is the vector of p_1's and p_2's local state c_1^r and c_2^r at the end of round $r \geq 1$; the initial state—which primarily incorporates the initial values of the consensus algorithm—is C^0. A configuration is *v-decided* (*decided* for short) if all processes have decided upon a common decision value $v \in \{0, 1\}$. A configuration C is *v-valent* (*univalent* for short) if all decided configurations reachable from C are v-decided; in particular, it is impossible to reach a 1-decided configuration from a 0-valent C.

Clearly, since we are considering deterministic algorithms in synchronous systems under link failures only, the entire execution is solely determined by C^0 and the pattern of message losses/corruptions. Actually, given any configuration C^{k-1}, there are only four possible (sets[9] of) *successor configurations* C_{00}^k, C_{01}^k, C_{10}^k, and C_{11}^k: Depending upon whether the message $p_2 \to p_1$ (x) and/or $p_1 \to p_2$ (y) is lost/faulty (0) or correct (1) in round $k \geq 1$, C^{k-1} is followed by a successor in C_{xy}^k in this execution (message "self-transmission", from $p_1 \to p_1$ and $p_2 \to p_2$ is always assumed to be failure-free here). Note that C_{00}^k is feasible only in Theorem 1.1, since both messages may be lost there. In the context of the following Theorem 4.3, however, C_{00}^k is empty since loosing both messages will not be feasible. Our notation can be generalized to $n > 2$ in the obvious way: (Sets of) successor configurations are indexed by strings of $n(n-1)$ 0's or 1's, corresponding to each link in a n-process system, with 0 denoting a lost or faulty message, and 1 denoting a non-faulty one.

In order to deal with different feasible link failure patterns in a consistent way, we introduce some more notation: Two successor configurations C_x^k and C_y^k are called *neighbors* if the sets $\mathcal{M}_x$ and $\mathcal{M}_y$ of received round k messages that led to C_x^k and C_y^k, respectively, differ in at most one message. For example, all configurations in C_{00}^k and C_{01}^k (and hence the entire sets) are neighbors in the above system, but the ones in C_{01}^k and C_{10}^k are not. The *successor graph* $\mathcal{G}_C$ of some configuration C

[9]To keep the notation simple, we group individual successor states together according to the pattern of message losses/corruptions here. For example, C_{01}^k actually consists of all configurations reachable from C^{k-1} where only the message $p_2 \to p_1$ is lost or faulty. Multiple configurations in C_{01}^k are possible in case of arbitrary link failures, since different faulty messages $p_2 \to p_1$ might result in different states of p_1. By assuming that univalence of a set of configurations C_{xy}^k means that all individual configurations $C^k \in C_{xy}^k$ are univalent, whereas bivalence means that least one individual configuration $C^k \in C_{xy}^k$ is bivalent, we can use sets of configurations instead of individual configurations in our proofs as well.

consists of all successor configurations of C, where all neighbors are connected by an edge. We can make the following fairly obvious observation:

LEMMA 4.1. *The successor graph $\mathcal{G}_C$ of any configuration C of a consensus algorithm under the system model of Definition 3.1 without process failures ($f_a = f_s = f_o = f_m = 0$) is connected.*

Proof: Let $k \geq 1$ be the round at the end of which the transition from C to one of its successor configurations takes place. Obviously, the failure-free successor configuration C_{1*} where no round k messages has been lost or corrupted must be in $\mathcal{G}_C$. Let C_x be any other successor configuration caused by a feasible link failure pattern, with $\overline{\mathcal{M}}_x$ denoting the corresponding set of lost or faulty messages. Since the link failure pattern $\overline{\mathcal{M}}'_x$, obtained from $\mathcal{M}_x$ by removing (= repairing) exactly one of the lost or faulty messages, is of course also feasible according to Definition 3.1, the resulting successor configuration C'_x is a neighbor of C_x and obviously $C'_x \in \mathcal{G}_C$. Since $|\overline{\mathcal{M}}'_x| = |\overline{\mathcal{M}}_x| - 1$, this argument can be repeated until the failure-free successor configuration $C'_x = C_{1*}$ is reached. Hence, there is a path from any C_x to C_{1*} in the successor graph $\mathcal{G}_C$. □

The result of Lemma 4.1 will be used primarily in conjunction with the following Lemma 4.2:

LEMMA 4.2. *Suppose that all successor configurations of some configuration C with connected successor graph $\mathcal{G}_C$ are univalent. If there are two arbitrary successor configurations C' and C'' among those that are 0-valent and 1-valent, respectively, then there are also two* neighboring *successor configurations $\overline{C}'$ and $\overline{C}''$ that are 0-valent and 1-valent.*

Proof: Since C' and C'' are connected in $\mathcal{G}_C$ and have different valences, there is a path of configurations connecting C' and C''. This implies that there must be neighbors $\overline{C}'$ and $\overline{C}''$ on this path where the valence changes. □

We are now ready to show that eventually bidirectional communications is mandatory for solving consensus in a 2-process system. Note that this theorem considers omission link failures only and strenghtens Gray's Theorem 1.1.

THEOREM 4.3 UNIDIRECTIONAL 2-PROCESS IMPOSSIBILITY. *There is no deterministic algorithm that solves consensus in a synchronous system with two non-faulty processes connected by a lossy link, if communication is reliable only in one direction that may change arbitrarily.*

Proof: Assume that there are programs $\mathcal{C}_{p_1}$ and $\mathcal{C}_{p_2}$ running on processes p_1 and p_2 that jointly solve consensus in a two-process system with unidirectional communication. We will show inductively that any bivalent configuration has at least one bivalent successor. This implies that it is impossible to always reach a final decision within any finite number of rounds.

For the base case $k = 0$ of our inductive construction, we have to show that there is a bivalent initial configuration Consider the configuration $C^0(01)$ where p_1 starts with initial value 0 and p_2 starts with initial value 1. If $C^0(01)$ is bivalent, we are done. If $C^0(01)$ is 0-valent, the execution where all messages from $p_1 \to p_2$ are lost in all rounds must also lead to a 0-decided configuration. This execution is indistinguishable for p_2 from the equivalent execution that starts from $C^0(11)$ (where p_1 has initial value 1 instead of 0), however, which implies that the common decision value must also be 0 here. Since $C^0(11)$ must lead to a 1-decided configuration in case of no link failures by validity, we have shown that $C^0(11)$ is bivalent in this case. An analogous argument can be used to show that $C^0(00)$ must be bivalent when $C^0(01)$ is 1-valent.

For the induction step $k \geq 1$, we assume that we have already reached a bivalent configuration C^{k-1} at the end of round $k-1$ and show that at least one of the feasible successor configurations C^k_{01}, C^k_{10}, and C^k_{11} reached at the end of round k is bivalent. Assuming the contrary, all of those must be univalent. The bivalence of C^{k-1} implies that at least one of C^k_{01}, C^k_{10}, and C^k_{11} must be 0-valent and one must be 1-valent (i.e., C^{k-1} must be a *fork* [Tel 1994]). By Lemma 4.2 in conjunction with Lemma 4.1, either the neighbors C^k_{11} and C^k_{01}, or C^k_{11} and C^k_{10} must be v-valent and $1-v$-valent, respectively. W.l.o.g. assume that C^k_{11} is v-valent and C^k_{01} is $1-v$-valent for some $v \in \{0,1\}$. Since the only difference between those two configurations is that the message from $p_2 \to p_1$ has been arriving in the former but not in the latter, we only have to consider the execution where all messages from $p_1 \to p_2$ are lost in all rounds $> k$. However, as p_1 cannot tell p_2 whether it has received a round k message, the appropriate executions starting from C^k_{11} resp. C^k_{01} are indistinguishable for p_2. Since p_2 must hence decide upon the same value, C^k_{11} and C^k_{01} cannot have different valences. □

Remarks:

(1) If message losses can only occur in one and the same direction, and if that direction is known to the algorithm, then there is a trivial 1-round algorithm that solves consensus in a 2-process system: The process that can communicate with its peer sends its own value and decides upon it; the other process decides upon the value received from its peer.

(2) If message losses can only occur in one and the same (but unknown) direction, there is a 2-round algorithm that solves consensus in a 2-process system: In the first round, the initial values v_1, v_2 are exchanged. In the second round, Acks/Nacks signaling successful reception in the former round are sent. The possible message receive patterns for the lucky process, say, p_2, that gets all messages are $(v_1, Ack) \models \min\{v_1, v_2\}$ and $(v_1, Nack) \models v_1$, whereas the less lucky p_1 could get $(v_2, Ack) \models \min\{v_1, v_2\}$, $(v_2, \emptyset) \models \min\{v_1, v_2\}$, $(\emptyset, Ack) \models v_1$, or $(\emptyset, \emptyset) \models v_1$. It is easy to verify that the decision values (given after $\models$ in each case) satisfy validity and agreement.

Our next goal will be to show that consensus cannot be solved in any system of $n \geq 2$ processes if, for every process, eventually bidirectional communication with at least one peer cannot be guaranteed. More specifically, if every process p could *withold* its information from some other process $q(p)$ (but not necessarily vice versa) arbitrarily long, in the sense that there is a failure pattern for rounds $R+1, R+2, \ldots$ such that $q(p)$ has the same view of the resulting execution after round R, independent of the information p has gathered in round R, then it is impossible to solve consensus. Note that this implies that there is no information flow from $p \rightarrow q(p)$ at all, neither directly nor indirectly via other processes.

Lemma 4.4 n-Process Impossibility. *Consider a synchronous n-process system with omission and/or arbitrary link failures, where the successor graph $\mathcal{G}_C$ of any configuration C is connected. There is no deterministic algorithm that solves consensus in such a system if, for every process p, it could happen that p witholds its information from some other process $q(p)$ for an arbitrary number of rounds.*

Proof: The proof is a generalization of the proof of Theorem 4.3, although more involved: We assume here that there are n programs $\mathcal{C}_1, \ldots, \mathcal{C}_n$ running on the n processes $p_1, \ldots, p_n$ in the system that jointly solve consensus. We will show inductively that there is an infinite sequence of bivalent configurations, which makes it impossible to always reach a decision within a finite number of rounds.

For the base case $k = 0$ of our inductive construction, we have to show that there is a bivalent initial configuration C^0. Consider the initial configuration $C^0(111*)$, where all processes start with the initial value 1. If this configuration is bivalent, we are done. Otherwise, $C^0(111*)$ can only be 1-valent, since validity requires a decision value 1 in the failure-free case. Now consider $C^0(011*)$, where process p_1 starts with 0 and all others with 1. If this configuration is bivalent, we are done. If not, we assume first that it is 0-valent and choose the execution starting from $C^0(011*)$ where p_1 cannot tell its value to some process $q(p_1) = p_x \neq p_1$; such a process must exist due to the assumptions of our lemma. This execution is indistinguishable for p_x from the analogous execution starting from $C^0(111*)$, however, so p_x's (and hence the common) decision must be 1 here. This contradicts the stipulated single-failure 0-valence of $C^0(011*)$, however, which could hence only be 1-valent.

The whole argument can now be repeated for p_2 in place of p_1, etc., until either a bivalent initial configuration has been found or the 1-valent initial configuration $C^0(*001)$ has been reached; in $C^0(*001)$, the processes $p_1, \ldots, p_{n-1}$ start with 0 and p_n starts with 1. We now consider the execution where p_n cannot communicate its value to some process $q(p_n) = p_y \neq p_n$. For p_y, this execution is indistinguishable from the analogous one starting from $C^0(*000)$, which must lead to a decision value of 0 by validity (C2). This contradicts the stipulated 1-validity of $C^0(*001)$, however.

For the induction step $k \geq 1$, we assume that we have already reached a bivalent configuration C^{k-1} at the end of round $k-1$. We must show that at least one of the feasible successor configurations C^k that could be reached at the end of round k is bivalent. If this is true in the first place, we are done. If not, all successor configurations C^k must be univalent. However, the bivalence of C^{k-1} implies that at least one of those must be 0-valent and one must be 1-valent (i.e., C^{k-1} must be a *fork* [Tel 1994]). By Lemma 4.2, there must also be 0-valent and 1-valent successor configuration C_0^k and C_1^k, respectively, that are neighbors. Assume that they differ only in the state of process r that has got some specific round k message correct in C_0^k but not or faulty in C_1^k (or vice versa). Now consider the two executions starting with C_0^k resp. C_1^k, where r witholds its round k-information from some process $q(r) = p_z \neq r$ in any future round $> k$. They are indistinguishable for p_z, which means that p_z and, by agreement, all other processes must compute the same decision in both executions. This contradicts the stipulated different valences of C_0^k and C_1^k, however. □

Lemma 4.4 has a number of interesting consequences. First of all, there is an interesting asymmetry in the "severeness" of receive link failures ($A1^r$) vs. send link failures ($A1^s$) in Definition 3.1. This can be seen by considering two instance of a 3-process systems, where two processes A, B cannot communicate bidirectionally due to receive resp. send link failures: In the system of type R shown in Figures 5, processes A and B may not receive the messages from both peers ($f_\ell^r = 2$ and $f_\ell^s = 1$). In the system of type S shown in Figure 6, processes A and B may fail to send to both peers ($f_\ell^r = 1$ and $f_\ell^s = 2$).

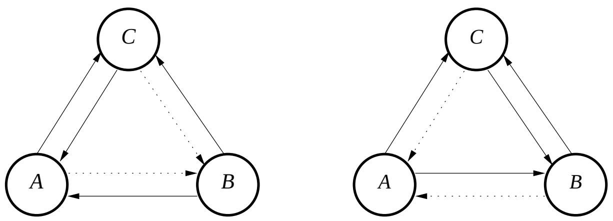

Fig. 5. *3-process system (type R), where processes A and B cannot communicate in one direction. The left scenario shows the case $A \not\to B$, the right one $B \not\to A$, which may alternate arbitrarily.*

It follows from Lemma 4.4 that no algorithm can solve consensus in a system of type R, even if process C is fixed and known to the algorithm. For, since A may fail to receive any information from any other process in the system, A is the process $q(p)$ required by Lemma 4.4 for any $p \neq A$. Since B may also fail to receive the information from any peer, it provides the still required $q(p)$ for process A. Hence, consensus is impossible in a 3-process system with $f_\ell^r = 2$ and $f_\ell^s = 1$; note that C is not fixed here, which makes consensus even harder to solve.

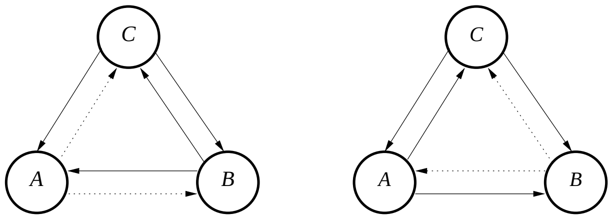

Fig. 6. *3-process system (type S), where processes A and B cannot communicate in one direction. The left scenario shows the case $A \not\to B$, the right one $B \not\to A$, which may alternate.*

On the other hand, for systems of type S where C is fixed and known to the algorithm, there is a trivial solution that lets all processes decide upon the value of process C. If process C is fixed but not known, consensus can be solved by means of the algorithm described in [Gridling 2002]. No solution exists in a system of type S only if C is not fixed — as is the case in a 3-process system with $f_\ell^r = 1$ and $f_\ell^s = 2$ according to Definition 3.1.

4.2 Number of Processes

Using the results of Section 4.1, we will first establish a lower bound for purely omissive link failures ($f_\ell^{ra} = f_\ell^{sa} = 0$ in Definition 3.1).

For the special case $f_\ell^s = f_\ell^r = f_\ell > 0$, such a lower bound can be infered from Theorem 1.1, by a straightforward simulation-type proof: Assume that there is a deterministic algorithm $\mathcal{C}$ that solves consensus for $n = 2f_\ell$. Using $\mathcal{C}$, it is possible to construct a solution for consensus in a 2-process system with lossy links, however, which is impossible.

The detailed proof is as follows: Partition the n processes into two subsets P_A and P_B of size f_ℓ each. Two *super-processes* A and B are used to simulate the execution of the processes in P_A and P_B, respectively. All the links between the simulated processes in the two super-processes are routed over a single *super-link*. For a super-process' decision value, any simulated process' decision value can be taken. In order to ensure that $\mathcal{C}$ achieves consensus among all (simulated) processes, we must show that Definition 3.1 is not violated for any simulated process in any super-process in case of a super-link failure: Any simulated process must not get more than $f_\ell^r = f_\ell$ receive link failures, and must not produce more than $f_\ell^s = f_\ell$ send link failures. This is trivially satisfied since $f_\ell^s = f_\ell^r = f_\ell$, however. Hence, our solution would achieve consensus among the two super-processes, which violates Theorem 4.3 (even Theorem 1.1, since bidirectional partitioning could happen here).

For the general case of arbitrary f_ℓ^s and f_ℓ^r, the lower bound for omission link failures can immediately be derived from Lemma 4.4:

THEOREM 4.5 LOWER BOUND PROCESSES 1. *Any deterministic algorithm that solves consensus under the system model of Definition 3.1 with $f_\ell^s, f_\ell^r > 0$ arbitrary needs $n > f_\ell^r + f_\ell^s$.*

Proof: We first show that, for any process p, we can choose a set $\mathcal{V}_p$ of f_ℓ^r arbitrary processes including p, where no process in $\mathcal{V}_p$ sends any messages to a process outside $\mathcal{V}_p$ in case of $n = f_\ell^s + f_\ell^r$: Since there are f_ℓ^s processes outside $\mathcal{V}_p$, every process in this set may commit broadcast link failures that omits all outside processes. Any outside process thus experiences exactly f_ℓ^r receive link failures, which is also feasible with respect to our failure model.

Any of the $f_\ell^s > 1$ processes outside $\mathcal{V}_p$ can hence be used as the process $q(p)$ required by Lemma 4.4, which implies that consensus cannot be solved here. □

Remarks:

(1) The lower bound $n > f_\ell^r + f_\ell^s$ provided by Theorem 4.5 is tight, since it is matched by the authenticated algorithm ZA analyzed in Section 7.2, see Theorem 7.6.

(2) It is interesting to note that the result of Theorem 4.5 implies that, in order to be able to solve consensus, any link failure may affect at most a minority of processes only. In the setting of Theorem 1.1, however, there is no point in considering this case at all: There is no non-empty minority of processes for

$n = 2$. We are reasonably convinced that our fairly simple escape from the impossibility result would have been discovered earlier during the past 20 years if Jim Gray had not presented his result for a simple 2-process system.

In order to find a lower bound for arbitrary link failures, we will again start with the special case $f_\ell^s = f_\ell^r = f_\ell^{sa} = f_\ell^{ra} = f_\ell > 0$. Our derivation will be based upon the following Theorem 4.6, which shows that no algorithm can solve consensus in a 4-process system in presence of a single arbitrary link failure ($f_\ell^{ra} = f_\ell^{sa} = 1$). This will be proved by means of a technique, which is well-known from showing the impossibility of consensus in a system of 3 processes with one Byzantine fault, see e.g. [Attiya and Welch 1998, p.104].

THEOREM 4.6 4-PROCESS IMPOSSIBILITY. *There is no deterministic algorithm that solves consensus under the system model of Definition 3.1 for a single arbitrary link failure in a 4-process system.*

Proof: We employ a new instance of the "easy impossibility proof techniques" of [Fischer et al. 1986] to show that any deterministic algorithm violates agreement if every process may see an inconsistent value from one of its neighbors. Suppose that our 4 processes execute a distributed algorithm consisting of specific programs A, B, C, D, which solve consensus under the system model of Definition 3.1 with $f_\ell^{ra} = f_\ell^{sa} = f_\ell^s = f_\ell^r = 1$. In order to derive a contradiction, we arrange 8 processes in a cube as shown in Figure 7. For example, the lower leftmost process labeled $A[0]$ executes program A starting with initial value 0 (the $\boxed{0}$ on its left displays this process's decision value, as explained below). Note carefully that all processes and all links are assumed to be non-faulty here.

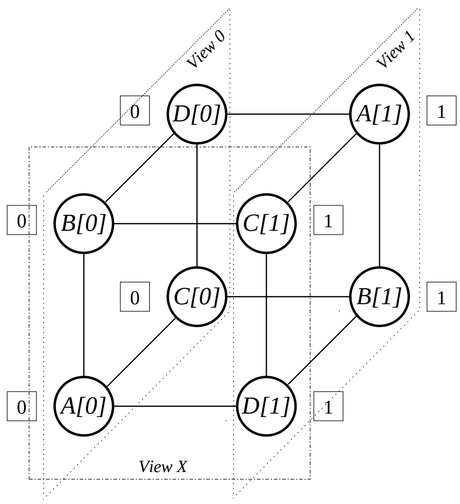

Fig. 7. *Topology used for proving the violation of agreement in a 4-process system. 8 non-faulty processes with perfect links are arranged in a cube in a neighborhood-preserving way. The assignment of initial values ensures that all processes in view 0 resp. view 1 decide* 0 *resp.* 1*, but this violates agreement in view* X.

Of course, dealing with a solution for a 4-process system, we cannot expect to achieve consensus in the 8-process system of Figure 7. However, due to the special assignment of programs to processes, each process observes a neighborhood as in a 4-process system. More specifically, the 4 processes at any side of the cube (we call it a *view*) can be interpreted as an instance of a legitimate 4-process system. In fact, as can be checked easily, our assignment ensures that any process in a view is connected to exactly one process outside this view. Since we assumed that every process may see an arbitrary faulty input from at most one neighbor, the input from the process outside the view may be arbitrary—it just appears like an arbitrary faulty process.

Now consider the processes in view 0, which all have initial value 0. By the validity property for consensus, all processes must decide 0 here (the initial value 1 of the processes outside view 0 do not matter, as they are considered arbitrary faulty w.r.t. view 0). Similarly, in view 1, all processes must decide 1 since they have initial value 1. But now the processes in view X face a problem: Since e.g. the lower leftmost process $A[0]$ observes exactly the same messages in view X as in view 0, it must decide 0 as observed above. Similarly, as the lower rightmost process $D[1]$ observes exactly the same messages in view X as in view 1, it must decide 1—but this would violate agreement in the 4-process system corresponding to view X. We hence established the required contradiction, thereby completing the proof of Theorem 4.6. □

Using Theorem 4.6, a similar simulation-type argument as in the pure omission failure case can be used to show the lower bound $n > 4f_\ell$ for $f_\ell^s = f_\ell^{sa} = f_\ell^r = f_\ell^{ra} = f_\ell > 0$ arbitrary link failures. Note that this lower bound is also tight, since it is matched by the non-authenticated algorithm OMH analyzed in Section 5.1, cf. Theorem 5.4.

THEOREM 4.7 LOWER BOUND PROCESSES 2. *Any deterministic algorithm that solves consensus under the system model of Definition 3.1 with $f_\ell^r = f_\ell^s = f_\ell^{sa} = f_\ell^{ra} = l > 0$ needs $n > 4f_\ell$.*

Proof: Assume that there is a deterministic algorithm $\mathcal{C}$ that solves consensus for $n = 4f_\ell$ in our model. We use $\mathcal{C}$ to construct a solution for a 4-process system of Theorem 4.6, which provides the required contradiction.

We partition the set of all processes P into 4 subsets P_A, P_B, P_C, P_D of the same cardinality f_ℓ, and let each *super-process* A, B, C, D simulate all the instances of the algorithm in the respective subset. For the super-process' decision value, any simulated process' decision value can be taken. In order to ensure that $\mathcal{C}$ achieves consensus among all (simulated) processes, we must show that Definition 3.1 is not violated for any process in any super-process if at most one super-link per process may experience an arbitrary link failure. Since any super-link hosts the links to and from exactly f_ℓ processes, this is trivially fulfilled, however: In case of a super-link failure, every sender process commits at most f_ℓ send link failures (affecting the f_ℓ processes in the receiving super-process), and every receiver process experiences at most f_ℓ receive link failures (from the f_ℓ processes in the sending super process).

Therefore, we have constructed an implementation of a consensus algorithm for a 4-process system, which can withstand a single arbitrary link failure. Since this is impossible by Theorem 4.6, the proof of Theorem 4.7 is completed. □

Unfortunately, we did not find an easy way to generalize the above simulation-type argument for an arbitrary number $f_\ell^s, f_\ell^{sa}, f_\ell^r, f_\ell^{ra} \geq 0$ of link failures. In order to derive a lower bound for n for this general case, we must hence resort to our key Lemma 4.4 again. What needs to be shown here, however, is that every process p could withold its information from at least one other process $q(p)$ arbitrary long: Lemma 4.9 below will prove that as much as $f_\ell^r + f_\ell^{ra}$ processes could withold their information from as much as $f_\ell^s + f_\ell^{sa}$ processes in case of $n = f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$, provided that

$$\frac{f_\ell^{ra}}{f_\ell^r} = \frac{f_\ell^{sa}}{f_\ell^s}. \tag{2}$$

Hence, by Lemma 4.4, $n > f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$ is a lower bound for solving consensus if (2) holds. If (2) does not hold, consensus can be solved for $n = f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$. The lower bound in this case is $n > \overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$, however, where $\overline{f}_\ell^r \leq f_\ell^r$, $\overline{f}_\ell^{ra} \leq f_\ell^{ra}$, $\overline{f}_\ell^s \leq f_\ell^s$, $\overline{f}_\ell^{sa} \leq f_\ell^{sa}$ are such that (2) holds and n is maximal.

Diving into the details of our lower bound proof, we start with a simple "balls and boxes" technical lemma. It shows that it is possible to drop white, orange and purple balls into a matrix such that each row and each column contains some specific numbers of balls of each color. This result will be used subsequently to assert the existence of a certain mapping of send and receive link failures, by interpreting a white, orange, and purple ball as a correct, omission faulty, and arbitrary faulty transmission between a particular sender process (column index) and receiver process (row index).

Lemma 4.8 Balls and Boxes. *Consider a matrix with $s + s^a$ rows and $r + r^a$ columns, where $s \geq s^a > 0$ and $r \geq r^a > 0$ and*

$$r/r^a = s/s^a. \tag{3}$$

Then it is possible to drop white, orange, and purple balls into the matrix (one ball per entry), such that any single row contains exactly r^a white, $r - r^a$ orange, and r^a purple balls, whereas any single column contains exactly s^a white, $s - s^a$ orange, and s^a purple balls.

Proof: First, we note that such an assignment could indeed exist, since summing up the number of balls of the same color by rows and columns, respectively, yields the same result. For example, we need $w_r = (s + s^a)r^a$ white balls when summing over rows, and $w_c = s^a(r + r^a)$ white balls when summing over columns. Since (3) implies $sr^a = s^a r$, it follows that $w_r = w_c$. We will now construct such an assignment explicitly.

Consider the first row in our matrix, and let

$$\pi_0, \pi_1, \ldots, \pi_{r+r^a-1}$$

with $\pi_i \in \{\mathit{white}, \mathit{orange}, \mathit{purple}\}$ be its assignment of balls to places according to the following rule: For any integer $x \geq 0$,

$$\pi_x = \begin{cases} \mathit{orange} \vee \mathit{purple} & \text{if } x = c(i) \text{ for some integer } i \geq 0, \\ \mathit{purple} & \text{if } x = c\big(a(j)\big) \text{ for some integer } j \geq 0, \\ \mathit{white} & \text{otherwise,} \end{cases}$$

where

$$\begin{aligned}
c(i) &= \left\lfloor \frac{r+r^a}{r} \cdot i \right\rfloor \\
a(j) &= \left\lfloor \frac{r}{r^a} \cdot j \right\rfloor \\
p(j) &= c\big(a(j)\big) = \left\lfloor \frac{r+r^a}{r} \cdot \left\lfloor \frac{r}{r^a} \cdot j \right\rfloor \right\rfloor .
\end{aligned}$$

This assignment distributes colored (orange or purple), as well as purple balls alone, as regularly as possible over the $r+r^a$ available places in the first row. The following periodicity properties are immediately apparent from the above definitions: For $0 \leq i \leq r-1$, $0 \leq j \leq r^a - 1$, and any integer $y \geq 0$,

$$0 \leq c(i) \leq r + r^a - 1 \quad \text{and} \quad c(i+r) = c(i) + r + r^a \tag{4}$$

$$0 \leq p(j) \leq r + r^a - 1 \quad \text{and} \quad p(j+r) = p(j) + r + r^a \tag{5}$$

$$\pi_{y+r+r^a} = \pi_y . \tag{6}$$

From the properties of $c(i)$ and $p(j)$, it follows immediately that $\pi_0, \pi_1, \ldots, \pi_{r+r^a-1}$ contains exactly r colored balls and r^a white ones. Clearly, every cyclic permutation (rotation) $\pi_y, \pi_{y+1}, \ldots, \pi_{y+r+r^a-1}$ of the original $\pi_0, \pi_1, \ldots, \pi_{r+r^a-1}$ has this property as well. Note that index addition in this cyclic permutation should actually be modulo $r + r^a$; (6) reveals that this is automatically taken care of, however. Below, we will assign $\pi_y, \pi_{y+1}, \ldots, \pi_{y+r+r^a-1}$ to row y of our matrix to prove our lemma.

By using the equivalences $(r + r^a)/r = (s + s^a)/s$ and $r/r^a = s/s^a$, which follow immediately from (3), in the definitions of $c(i)$ and $p(i)$, we obtain similar periodicity properties for $0 \leq i \leq s-1$, $0 \leq j \leq s^a - 1$ and any integer $x \geq 0$:

$$0 \leq c(i) \leq s + s^a - 1 \quad \text{and} \quad c(i+s) = c(i) + s + s^a \tag{7}$$

$$0 \leq p(j) \leq s + s^a - 1 \quad \text{and} \quad p(j+s) = p(j) + s + s^a \tag{8}$$

$$\pi_{x+r+s^a} = \pi_x . \tag{9}$$

As before, this implies that $\pi_0, \pi_1, \ldots, \pi_{s+s^a-1}$ contains exactly s colored balls and s^a white ones. Even more, the periodicity properties (7) and (8) imply that every cyclic permutation (rotation) $\pi_x, \pi_{x+1}, \ldots, \pi_{x+s+s^a-1}$ of $\pi_0, \pi_1, \ldots, \pi_{s+s^a-1}$ has this property as well; again, (9) takes care of index addition modulo $s + s^a$.

Hence, we just have to assign $\pi_y, \pi_{y+1}, \ldots, \pi_{y+r+r^a-1}$ to row y of our matrix, meaning that the entry in column 0 of row y contains the same ball as the entry in column y of row 0, for example. Our findings on the number of balls in cyclic permutations of $\pi_0, \ldots, \pi_{r+r^a-1}$ shows that this assignment respects our lemma's requirement on rows. Similarly, the inspection of the resulting matrix shows that column x contains the pattern $\pi_x, \pi_{x+1}, \ldots, \pi_{x+s+s^a-1}$, which respects our requirement on the number of balls in columns as well. This completes the proof of Lemma 4.8. □

Now we are ready for proving our major Lemma 4.9, which shows that, in case of $n = f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$ processes satisfying (2), any two executions that lead to two sufficiently "similar" configurations, in the sense that at least $f_\ell^s + f_\ell^{sa}$ processes have identical state in both, can be extended by one round in a way that again yields two "similar" configurations. This implies that as much as $f_\ell^r + f_\ell^{ra}$ processes can withold their information from as much as $f_\ell^s + f_\ell^{sa}$ processes, which is amply sufficient to finally apply Lemma 4.4 for establishing our general lower bound result.

Lemma 4.9 Similarity. *Consider two configurations $C = (c_1, \dots, c_n)$ and $C' = (c'_1, \dots, c'_n)$ generated by executions E and E' in a system of $n = f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$ processes according to Definition 3.1 satisfying $f_\ell^r / f_\ell^{ra} = f_\ell^s / f_\ell^{sa}$, where the states $c_1 = c'_1, \dots, c_{f_\ell^s + f_\ell^{sa}} = c'_{f_\ell^s + f_\ell^{sa}}$ of at least $f_\ell^s + f_\ell^{sa}$ processes are the same. Then, E and E' can be feasibly extended by one round, such that all those $f_\ell^s + f_\ell^{sa}$ processes have again the same states $d_1 = d'_1, \dots, d_{f_\ell^s + f_\ell^{sa}} = d'_{f_\ell^s + f_\ell^{sa}}$ in the resulting successor configurations $D = (d_1, \dots, d_n)$ and $D' = (d'_1, \dots, d'_n)$.*

Proof: Let $\mathcal{S} = \{s_0, \dots, s_{f_\ell^s + f_\ell^{sa} - 1}\}$ be the set of processes with equal states, and $\mathcal{R} = \{r_0, \dots, r_{f_\ell^r + f_\ell^{ra} - 1}\}$ be the set of the remaining processes with possibly different states in C and C'. We claim that there is a feasible link failure pattern L extending E by one round, such that every process in $\mathcal{S}$ gets exactly f_ℓ^{ra} arbitrary link failures from some processes in $\mathcal{R}$, delivering the message that would have been sent in the absence of link failures in E'. In addition, every process in $\mathcal{S}$ also experiences exactly $f_\ell^r - f_\ell^{ra}$ omission link failures from some processes in $\mathcal{R}$, whereas the messages from the remaining f_ℓ^{ra} processes in $\mathcal{R}$ are received correctly. All message transmissions from processes in $\mathcal{S}$ to processes in $\mathcal{S}$, as well as all transmissions to processes in $\mathcal{R}$ are failure-free.

Not surprisingly, the required link failure pattern has already been established in Lemma 4.8: We just have to map $r = f_\ell^r$, $r^a = f_\ell^{ra}$, $s = f_\ell^s$ and $s^s = f_\ell^{sa}$ and interpret white, orange, and purple balls as correct, omission faulty, and arbitrary faulty transmissions from the $f_\ell^r + f_\ell^{ra}$ processes in $\mathcal{R}$ (columns) to the $f_\ell^s + f_\ell^{sa}$ processes in $\mathcal{S}$ (rows). The results of Lemma 4.8 reveals that the corresponding link failure pattern respects both the maximum number of send and receive link failures.

Knowing that L exists, our lemma follows immediately by extending E' with the pattern L', which is exactly L except that a process that commited an arbitrary send link failure in L transmits correctly in L', whereas a process that transmitted correctly in L commits an arbitrary send link failure in L', which erroneously generates the message that would have been transmitted in E. Obviously, every process in $\mathcal{S}$ has the same view of the execution both in $E \cup L$ and $E' \cup L'$ and hence reaches the same configuration as asserted. $\Box$

Now it is not difficult to prove our general lower bound result as given by Theorem 4.7. Although none of the algorithms presented in this paper matches this bound, we are reasonably convinced that it is tight. In fact, it is not difficult to verify that an exponential algorithm like OMH that is allowed to execute more than 2 rounds solves consensus for $n = f_\ell^r + f_\ell^{ra} + f_\ell^s + f_\ell^{sa}$ if (2) is violated.

Theorem 4.10 Lower Bound Processes 3. *Any deterministic algorithm that solves consensus under the system model of Definition 3.1 needs $n > \overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$, where $\overline{f}_\ell^r \le f_\ell^r$, $\overline{f}_\ell^{ra} \le f_\ell^{ra}$, $\overline{f}_\ell^s \le f_\ell^s$, $\overline{f}_\ell^{sa} \le f_\ell^{sa}$ are such that $\overline{f}_\ell^r / \overline{f}_\ell^{ra} = \overline{f}_\ell^s / \overline{f}_\ell^{sa}$ holds and the sum $\overline{f}_\ell^r + \overline{f}_\ell^{ra} + \overline{f}_\ell^s + \overline{f}_\ell^{sa}$ is maximal.*

Proof: Due to Theorem 4.5, it only remains to provide an impossibility proof for $f_\ell^{sa}, f_\ell^{ra} > 0$. According to Lemma 4.4, we just have to show that, for every process p, it could happen that p witholds its information from at least one process $q(p)$ arbitrary long under the conditions of our theorem: More specifically, we need a failure pattern for rounds $R+1, R+2, \dots$ such that $q(p)$ has the same view of the resulting execution after round R, independent of the information p has gathered in round R.

This follows easily from inductively applying Lemma 4.9, however: If we assume that p is just one of the $f_\ell^r + f_\ell^{ra}$ processes that may have different state in two R-round executions E resp. E' leading to configurations C resp. C', we are guaranteed that the remaining $f_\ell^s + f_\ell^{sa}$ processes that had identical state in E and E' have identical state in some suitable 1-round extension $E \cup L$ and $E' \cup L'$ of E and E' again. Hence, no such process ever gets information from p. Since this can go on arbitrarily often, we can choose $q(p)$ to be any of these $f_\ell^s + f_\ell^{sa}$ processes. □

4.3 Number of Rounds

In this subsection, we will show that being able to handle link failures comes at the price of additional running time. More specifically, compared to the case without link failures, solving consensus in case of $f_\ell^s, f_\ell^r > 0$ requires one additional round. Our proofs will again be based upon bivalency arguments and re-use some of the results developed in the previous subsections.

THEOREM 4.11 LOWER BOUND ROUNDS 1. *Any deterministic algorithm that solves consensus under the system model of Definition 3.1 for $f_\ell^s, f_\ell^r > 0$ needs at least 2 rounds.*

Proof: Assume that there is a 1-round algorithm that solves consensus in presence of link failures. Obviously, since any process p may suffer from a send link failure to any receiver process q, any p could withold its information from at least one $q(p)$ here. Note carefully that the 1-round assumption does not allow other processes to learn about p's information in some later round. Therefore, Lemma 4.4 reveals that solving consensus is impossible here. □

The above result can be extended to the case where both link and process failures may occur. Using our ideas in the simple bivalency proof of the well-known $f + 1$ lower bound for the number of rounds required for solving consensus in presence of f process crashes (omission failures in our model) developed in [Aguilera and Toueg 1998], it is possible to show that $f + 2$ is a tight lower bound (matched e.g. by our OMH in Section 5.1) for the required number of rounds.

THEOREM 4.12 LOWER BOUND ROUNDS 2. *Any deterministic algorithm that solves consensus under the system model of Definition 3.1 with $n \geq f + f_\ell^s + f_\ell^r$, where f denotes the maximum number of process crash failures and $f_\ell^s, f_\ell^r > 0$, needs at least $f + 2$ rounds.*

Proof: In [Aguilera and Toueg 1998], a simple forward induction based upon a bivalency argument involving message losses due to process crashes is used to show that any consensus algorithm has executions that lead to at least one bivalent configuration at the end of round $f - 1$. The executions considered in this proof are such that at most one process may crash in every round; clearly, no link failures are assumed to occur here. The impossibility of consensus within f follows by contradiction: It is shown in [Aguilera and Toueg 1998, Lemma 1] that the existence of such a solution would imply that all configurations reached after $f-1$ rounds must be univalent.

In order to prove Theorem 4.12, we only have to provide an analogon to [Aguilera and Toueg 1998, Lemma 1]: That the existence of a consensus algorithm that decides after $f+1$ rounds in our failure model would imply that all configurations reached after $f-1$ rounds are univalent. Assuming the contrary, there must be a bivalent configuration C^{f-1} after round $f-1$. Note carefully that, unlike in Section 4.2, the message losses—with respect to bivalence—considered here are only due to process crashes, not link failures. Since at most one process may have crashed during each of the first $f-1$ rounds, there is still one process p that may crash in round f or $f+1$, which is required for C^{f-1} to be bivalent.

Let v be the algorithm's decision in the execution E, where p does not crash in the two rounds following C^{f-1}. Due to the bivalence of C^{f-1}, there must be a different execution $\overline{E}$ also starting from C^{f-1}, where p crashes in either round f or $f+1$ and the decision is $1-v$. Assume first that p crashes in round f in $\overline{E}$. Then, there must be two executions E^q leading to the decision value v, and $\overline{E}^q$ leading to $1-v$, which differ only in that the (crashing) p sends its round f message to q in E^q but not in $\overline{E}^q$. For, starting from E where p sends all its messages, we remove the messages p succeeds to send one by one until the decision value changes; this happens at latest when we arrived at the execution $\overline{E}^q$.

By construction, q is the only process that can differentiate between E^q and $\overline{E}^q$ after round f. If we allow q to produce a send link failure to some other correct process r (this process must exist since $n \geq f+2$) in the final round $f+1$, then r has the same view at the end of E^q and $\overline{E}^q$. Hence, the resulting decisions cannot be different.

We still have to deal with the second case, where p is failure-free in round f but crashes in round $f+1$ in $\overline{E}$. In that case, the configuration C^f obtained from C^{f-1} in the failure-free execution E must be bivalent as well. After all, E and $\overline{E}$ are the same up to and including round f. In that case, however, the original proof of [Aguilera and Toueg 1998, Lemma 1] applies, thereby providing the required contradiction also in this case. □

4.4 Characterization of Process Failures

There are some consequences of the results of the previous sections related to process failures, which are also interesting from a theoretical point of view: The effect of an omission resp. arbitrary faulty process on its peers is principally the same as the effect of an omission resp. arbitrary link failure committed by a non-faulty process. Hence, the question arises why f omission faulty processes require at least $f+1$ rounds of execution (Theorem 4.12), whereas an arbitrary number of link failures can be handled in just 2 rounds (Theorem 4.11). We will address this question in this subsection.

From the failure model in Definition 3.1, it is apparent that an arbitrary broadcast failure (A1^s) can also be viewed as the consequence of a *restricted process* *failure* with inconsistency limited to f_ℓ^s. Since $f_\ell^s < n$, the inconsistency caused by such a broadcast failure is strictly less than that of an arbitrary faulty process, however, since the latter is not restricted in the number of recipients that may get a faulty message. If at most $f_\ell^r = f_\ell^{ra}$ processes suffer from restricted process failures with inconsistency limited to $f_\ell^s = f_\ell^{sa}$ ($f_\ell^r \geq f_\ell^s$, as mentioned in Remark 2 on Definition 3.1, need not hold here), both (A1^s) and (A1^r) are satisfied, which shows that Definition 3.1 indeed allows this alternative interpretation as well. Nevertheless, (A1^s) and (A1^r) admit more general failure patterns than restricted process failures: A receive failure may hit any incoming link in the former, but is restricted to one of the links from the f_ℓ^r restricted faulty processes in the latter.

Anyway, using the alternative interpretation of arbitrary link failures as restricted arbitrary process failures, the result of Theorem 4.10 allows us to characterize what makes a process failure a truly arbitrary (Byzantine) one: Setting $f_a = f_s = f_o = f_m = 0$, $f_\ell^r = f_\ell^{ra} > 0$ arbitrary and fixing $n > 2f_\ell^{ra}$, the optimal consensus algorithm can tolerate f_ℓ^{ra} restricted arbitrary process failures with inconsistency limited to

$$f_\ell^{sa} \leq \lfloor (n-1)/2 \rfloor - f_\ell^{ra} \tag{10}$$

since $n \geq 2f_\ell^{sa} + 2f_\ell^{ra} + 1$ here. For example, $n = 10$ processes are required for tolerating one restricted arbitrary failure with inconsistency $f_\ell^{sa} = 3$. Due to (10), at least $\lceil (n-1)/2 \rceil + f_\ell^{ra}$ processes, i.e., a majority[10] of the non-faulty processes, get the correct message even in the broadcast of a restricted arbitrary faulty process. Provided that n is chosen appropriately, any number f_ℓ^{ra} of process failures with inconsistency limited to f_ℓ^{sa} can be tolerated in two rounds here, i.e., in a number of rounds that does not depend upon the number of failures f_ℓ^{ra}. If, on the other hand, more than f_ℓ^{sa}, i.e., more than a minority of the non-faulty processes, can get a faulty message in the broadcast of a process, then the sender must be considered arbitrary faulty and thus increases the number of rounds required for solving consensus.

A similar observation can be made for omission failures. Again setting $f_a = f_s = f_o = f_m = f_\ell^{sa} = f_\ell^{ra} = 0$, $f_\ell^r > 0$ arbitrary and fixing $n > f_\ell^r$, the optimal consensus algorithm can tolerate f_ℓ^r restricted omission process failures with inconsistency

$$f_\ell^s \leq n - 1 - f_\ell^r \tag{11}$$

according to Theorem 4.5. It follows from (11) that at least $f_\ell^r + 1$ processes, i.e., at least one non-faulty process, must get the correct message even in the broadcast of a restricted omission faulty process. If this is warranted, any number f_ℓ^r of such restricted process failures can be tolerated within 2 rounds.

It hence follows that a process that disseminates inconsistent information cannot do much harm—in the sense of requiring additional rounds for solving consensus—if at most a certain number of recipients can get inconsistent information:

—A process must be considered arbitrary faulty and hence accounted for in f_a if it can supply wrong information to a majority of the non-faulty processes.

[10]For a sub-optimal algorithm, even more than a majority of the non-faulty processes must get the correct message here.

—A process must be considered omission faulty and hence accounted for in f_o if it can fail to provide information to any and all non-faulty processes.

For sub-optimal algorithms, the thresholds (numbers of affected receivers) are of course smaller.

Note finally that our observations do not contradict the lower-bound $f+1$ for the number of rounds required for consensus in presence of f faulty processes, recall Theorem 4.12 or [Attiya and Welch 1998, Sec. 5.1.4], since this result relies heavily upon the fact that a faulty process can disseminate inconsistent information to as many processes as desired.

5. HYBRID ORAL MESSAGES ALGORITHMS

In this section, we will show that the *Hybrid Oral Messages* algorithm OMH derived from [Lamport et al. 1982] in [Lincoln and Rushby 1993], 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 itself, as may omission faulty processes.

5.1 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*; 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 consensus algorithm is obtained by using a separate instance of Byzantine agreement for disseminating any process's local value and using a suitable choice function (majority) for the consensus result.

The algorithm OMH as specified in Definition 5.1 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 $R^{-1}(R(v)) = v$. Note that only $\emptyset$, $R(\emptyset)$, $R(R(\emptyset))$, $R(R(R(\emptyset)))$, ... must actually be distinguishable here; for each legitimate value v, we can allow $R(v) = R^{-1}(v) = v$.

—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 presence of an omission faulty transmitter, see the proof of Lemma 5.3.

Consult [Lincoln and Rushby 1993] for a detailed discussion of the above primitives and the operation of OMH in general.

Definition 5.1 Algorithm OMH [Lincoln and Rushby 1993]. *The Hybrid Oral Message algorithm OMH is defined recursively as follows:*

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.[11]

(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 5.2 given below in OMH's analysis.

(3) During the recursive execution of OMH, multiple (in fact, quite many, see Table I in Section 8) instances of OMH are concurrently active[12] 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.

[11] It is interesting to note that this requires only a "weak" consensus algorithm in case of $f_\ell^s < f_\ell^r$ (and no process failures). In this case, a majority of the receivers of the initial broadcast already get the correct value, which makes the job of the consensus algorithm much easier.

[12] Note that all the peer processes belonging to a single concurrent instance are considered a separate system of n processes.

The unique *id* of the transmitter process is appended to each message for this purpose, which produces a string of ids $p_1p_2\dots p_k$ 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.

(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 [Attiya and Welch 1998, p. 105]. Consequently, not only non-faulty but also obedient processes can always compute the correct value to be forwarded in each particular instance—and hence in every round—independently of their past behavior, cp. Remark 8 on Definition 3.1.

By adopting and extending the analysis of [Lincoln and Rushby 1993] to our perception-based failure model, we will now show that OMH satisfies (B1) and (B2).

We start our derivations with a widely applicable technical Lemma 5.2, 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, recall Remark 2 on Definition 5.1. Note carefully that a process p_m on a manifest faulty sender node may very well send its $R(w_{p_m})$ in the conveying round, since the manifest fault may occur in some other round or process.

LEMMA 5.2. *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 $\mathcal{S}$ with*

(a) $|\mathcal{S}| > 2f + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + 2f_o + f_m$ resp.
(b) $|\mathcal{S}| > 2f + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m$

be given. All processes in $\mathcal{S}$ are assumed to convey their values to an obedient receiver q in some round of the execution of a distributed algorithm. If all but at most f of the (a) non-faulty or manifest faulty resp. (b) obedient processes $p \in \mathcal{S}$

(1) own the same value $w_p = \nu$, then q computes $v_q = \nu$,
(2) own 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[13] 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 a faulty value).

[13] 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.

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

We start with proving item (1) of the statements-part of our lemma. In case (a), where all non-faulty and manifest faulty processes $\in \mathcal{S}\backslash\mathcal{S}_f$ own the same value $R(\nu)$, our obedient receiver q will obtain at least

$$n'_q = n' - f - f_a - f_s - f_o - f'_m - {f^r_{\ell n}}'$$

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

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

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

$$\begin{aligned} 2n'_q - n''_q &= n' - 2f - 2(f_a + f_s) - 2f_o - f'_m - 2{f^r_{\ell n}}' + f'_o + f^{ro\prime\prime}_\ell - f^{ra\prime}_{\ell o} - {f^{ra}_{\ell m}}' \\ &> (f_m - f'_m) + f'_o + (f^r_\ell - {f^r_{\ell n}}') + (f^{ra}_\ell + f^{ro\prime\prime}_\ell - {f^r_{\ell n}}' - f^{ra\prime}_{\ell o} - {f^{ra}_{\ell m}}') \\ &\geq 0. \end{aligned} \tag{12}$$

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

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

$$\overline{n}'_q = n' - f - f_a - f_s - f'_o - f'_m - {f^r_{\ell n}}'$$

identical values $R(\nu)$ and will get at most

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

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

$$\begin{aligned} 2\overline{n}'_q - \overline{n}''_q &= n' - 2f - 2f_a - 2f_s - f'_o - f'_m - 2{f^r_{\ell n}}' + f^{ro\prime\prime}_\ell - f^{ra\prime}_{\ell o} - {f^{ra}_{\ell m}}' \\ &> (f_o - f'_o) + (f_m - f'_m) + (f^r_\ell - {f^r_{\ell n}}') + (f^{ra}_\ell + f^{ro\prime\prime}_\ell - {f^r_{\ell n}}' - f^{ra\prime}_{\ell o} - {f^{ra}_{\ell m}}') \\ &\geq 0, \end{aligned} \tag{13}$$

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

As far as part (2) of the statements of Lemma 5.2 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). However, since (12) resp. (13) holds if we temporally consider each value from $\mathcal{W}$ 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 5.3 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 5.3 VALIDITY OMH. *For any $m \geq \min\{1, f_\ell^s\}$ and any f_a, f_s, f_o, f_m, f_ℓ^s, f_ℓ^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[14] process p, including the transmitter, delivers

(1) $v_p = \nu = v$, if the transmitter is non-faulty,

(2) $v_p = \nu = \emptyset$, if the transmitter is manifest faulty,

(3) $v_p = v$ or $v_p = \emptyset$, if the transmitter is omission faulty,

(4) $v_p = \nu$, 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 ν ($\nu = v$ 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 Definition 3.1.

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

$$n' \geq 2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2f_a + 2f_s + f_o + f_m + 1,$$

all but at most f_ℓ^s of the obedient receivers p of step 1 of OMH(1) get the same $w_p = \nu$ (we can simply assume $\nu = \emptyset$ 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 (1), case (b) of Lemma 5.2 with $f = f_\ell^s$ to conclude that every obedient receiver will deliver the same value $w_p = \nu$ as the result of OMH(1). In the remaining case (3), where the transmitter is omission faulty, all obedient receivers get either $w_p = \nu$ or $w_p = \emptyset$. In this case, item (2), case (b) of Lemma 5.2 with $f = f_\ell^s$ and $\mathcal{W} = \{\nu, \emptyset\}$ shows that every obedient receiver q must deliver either ν or $\emptyset$ as required.

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 $R(w_q) = R(\nu)$. 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 $R(\nu)$ in this step for every

[14]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) as introduced in Section 5.2.

non-faulty transmitter q. Consequently, any obedient receiver p must have at least

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

values equal to $R(\nu)$ 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 $m > 0$,

$$\begin{aligned} 2\overline{n}'_p - \overline{n}''_p &= n' - 2f^s_\ell - 2f_a + f'_a - 2f_s - f'_o - f'_m \\ &\geq f^r_\ell + f^{ra}_\ell + f'_a + (f_o - f'_o) + (f_m - f'_m) + m \\ &> 0, \end{aligned} \tag{14}$$

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

For the remaining case (3), exactly the same reasoning as in the proof of item (2) of Lemma 5.2 reveals that only the default value $R(\emptyset)$ can be returned by hybrid-majority if no majority of either $R(\nu)$ or $R(\emptyset)$ exists. This eventually completes the proof of Lemma 5.3. □

With the help of Lemma 5.3, it is not too difficult to show the major Theorem 5.4.

Theorem 5.4 Agreement & Validity OMH. *For any $m \geq f_a+f_o+\min\{1, f^s_\ell\}$ and f_a, f_o, f_s, f_m, f^s_ℓ, f^r_ℓ, $f^{ra}_\ell \geq 0$, the algorithm OMH(m) satisfies agreement (B1) and validity (B2) if there are strictly more than $2f^s_\ell+f^r_\ell+f^{ra}_\ell+2(f_a+f_s)+f_o+f_m+m$ participating processes.*

Proof: Since Lemma 5.3 is applicable under the conditions of Theorem 5.4, 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 [Lincoln and Rushby 1993]. In the base case $m = \min\{1, f^s_\ell\}$, we must have $f_a = f_o = 0$ since $m \geq f_a + f_o + \min\{1, f^s_\ell\}$ by assumption. Hence, the transmitter must not be arbitrary or omission faulty here, such that Lemma 5.3 also implies (B1).

We can therefore assume that (B1) is satisfied by OMH$(m-1)$ with $m - 1 \geq \min\{1, f^s_\ell\}$, 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 5.3 also implies (B1) in the other cases. Since we have at most $f_a + f_o \leq m - \min\{1, f^s_\ell\} \geq 1$ 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 $2f^s_\ell + f^r_\ell + f^{ra}_\ell + 2(f_a + f_s) + f_o + f_m + m - 1$ processes. Since obviously

$$\begin{aligned} &2f^s_\ell + f^r_\ell + f^{ra}_\ell + 2(f_a + f_s) + f_o + f_m + m - 1 \; > \\ &\quad 2f^s_\ell + f^r_\ell + f^{ra}_\ell + 2([f_a - 1] + f_s) + f_o + f_m + [m - 1] \end{aligned}$$

as well as

$$\begin{aligned} &2f^s_\ell + f^r_\ell + f^{ra}_\ell + 2(f_a + f_s) + f_o + f_m + m - 1 \; > \\ &\quad 2f^s_\ell + f^r_\ell + f^{ra}_\ell + 2(f_a + f_s) + [f_o - 1] + f_m + [m - 1], \end{aligned}$$

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^q_p

in step (3). 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 for 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 of this type that achieves validity for a given m will also achieve agreement for $m' = m + f_a + f_o$.

(2) Lemma 5.3 and Theorem 5.4 and their proofs are also valid if the at most f_m manifest faults were not strong but rather weak manifest faults, cf. Remark 6 on Definition 3.1. 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 it may not even be hit by a link failure, by assumption.

(3) The proof of Lemma 5.3 revealed that OMH satisfies *uniform* validity (UB2) as introduced in Section 5.2, recall Footnote 14: 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 5.4, 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 5.4) is valid when one of the two receivers is the manifest faulty transmitter process q here as well.

(4) It is apparent from Theorem 5.4 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 [Perry and Toueg 1986] 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.

(5) From the proof of Lemma 5.3, 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 Remark 2 on Definition 3.1.

(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 5.4, 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 broadcast 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 $R()$.

5.2 Uniform Byzantine Agreement

Properties (B1) and (B2) in Section 5.1 require agreement and validity to hold only on non-faulty processes. Since there are applications where the behavior of benign faulty processes also matters, cp. [Schmid et al. 2002, Sec. 4], one could ask whether it is possible to extend those properties to obedient processes. In this subsection, we will hence consider the following *uniform* properties [Hadzilacos and Toueg 1993]:

(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 this would contradict the impossibility of uniform consensus in presence of Byzantine failures postulated in [Charron-Bost and Schiper 2000]. Nevertheless, if symmetric and arbitrary faulty processes would faithfully execute the algorithm, i.e., would produce their faults only when emitting messages, then our algorithm below would achieve uniform consensus for them as well.

Originally, we (erroneously) conjectured that the proofs of Lemma 5.3 and Theorem 5.4 carry over literally to uniform validity (UB2) and uniform agreement (UB1). Formal verification in [Rushby 2001] showed that this is indeed true[15] for (UB2), but fails in case of (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 5.4 (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 [Rushby 2001], 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.

[15] We mentioned already in Remark 3 on Theorem 5.4 that the proof of Lemma 5.3 actually shows (UB2).

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 5.5 and Theorem 5.6 below show that OMHU satisfy uniform agreement (UB1) and uniform validity (UB2).

LEMMA 5.5 VALIDITY OMHU. *For any $m \geq \min\{1, f_\ell^s\}$ and any f_a, f_o, f_s, f_m, f_ℓ^s, f_ℓ^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 5.3, we know that, in case of a consistent transmitter, all obedient processes in OMH deliver the same value ν 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 (1), case (a) of Lemma 5.2 with $f = 0$ to conclude that every obedient process will receive enough values to eventually deliver ν.

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 ν or $\emptyset$. However, item (2), case (a) of Lemma 5.2 with $f = 0$ and $\mathcal{W} = \{\nu, \emptyset\}$ applies here. Since $\mathcal{W} \cup \{\emptyset\} = \mathcal{W}$, any obedient process can deliver ν or $\emptyset$ only. □

THEOREM 5.6 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_ℓ^s, f_ℓ^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 5.5, it only remains to prove (UB1). First of all, as noted in Remark 3 on Theorem 5.4, the original OMH satisfies agreement (B1) not only on non-faulty, but also on (strong) manifest faulty processes. Hence, both non-faulty and (alive) manifest faulty processes in OMH deliver the same value ν. 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 (1), case (a) of Lemma 5.2 for $f = 0$ amply ensures that all obedient processes will deliver ν 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, where $n(n-1)$ messages are exchanged.

6. AUTHENTICATION

Written messages [Lamport et al. 1982] 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 [Gong et al. 1995]). With electronic signatures, each process p uses its private *signature*

σ_p to sign some piece of information v, thereby generating the corresponding *signed information*[16] $\sigma_p(v)$. Without knowing σ_p, it is computationally infeasible (ideally impossible) to compute $\sigma_p(v)$ from v. Recovering the content v of some valid signed information $\sigma_p(v)$ is easily done, however, by applying the (usually public) *inverse signature* σ_p^{-1}.

We will place the following specific assumptions on the signature scheme:

(SA1) Only process p can be the originator of $\sigma_p(v)$.

(SA2) Only process p can change the content v of $\sigma_p(v)$.

(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 $\sigma_p^{-1}(SI) = v$ iff $SI = \sigma_p(v)$, whereas $\sigma_p^{-1}(SI) = \text{ERROR}$ for some distinguished value ERROR otherwise.

Note carefully that (SA1) implicitly requires countermeasures against replay attacks, since an eavesdropping process could otherwise inject pre-recorded messages from an earlier execution run. This could 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 v out of an incorrect SI), cf. [Gong et al. 1995].

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 anymore. More specifically, some other process $q \neq p$ could then manufacture signed messages $\sigma_p(v)$ as if they originated from process p.

In the subsequent sections, we will investigate authenticated Byzantine agreement algorithms that disseminate the transmitter t's value v by forwarding signed messages via paths of distinct nodes, one hop per round, as does the non-authenticated algorithm OMH of Section 5.1: In round k, a value v that is sent from process $p_1 = t$ along the chain of distinct processes $p_2 \dots p_k$ usually arrives at some receiver q as a multiply signed message $M = \sigma_{p_k} \cdots \sigma_{p_1}(v)$. Due to an omission/value failure, however, it might happen that 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 $\emptyset$ (we assume that manifest faults of messages, like invalid signatures, reception of multiple messages in the same round etc. are mapped to $\emptyset$ as usual). The algorithm ZAr (see Section 7.3) simply stops forwarding in this case, so that no message M will arrive at q in this case. Other algorithms like OMHA (Section 7.1) and ZA (Section 7.2) cause $p_{\ell+1}$ to generate a message $\sigma_{p_{\ell+1}}(R_\emptyset)$ containing a specific value $R_\emptyset$ meaning "I am reporting $\emptyset$" in this case, which is forwarded along the remaining path as usual. Consequently, the final receiver q could get a signed message $\sigma_{p_k} \cdots \sigma_{p_{\ell+1}}(R_\emptyset)$ instead of $M = \sigma_{p_k} \cdots \sigma_{p_1}(v)$ here as well.

All our algorithms explicitly need the string of forwarding process id's $p_k, \dots, p_1$ of any round k message, both for assigning an arriving message to the appropriate

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

concurrent instance of the algorithm (recall Remark 3 on Definition 5.1) 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 $R_\emptyset$ must encode $p_\ell, \dots, p_1$ in $R_\emptyset = R_\emptyset^{p_\ell,\dots,p_1}$. We usually prefer the shorthand notation $R_\emptyset$, however, to keep the notation simple.

The following Definition 6.1 summarizes the properties of *valid* forwarded messages. A message that is not valid is called *manifest faulty*.

Definition 6.1 Valid Messages. *For $k \geq 1$, a message M arriving at some process p via a forwarding path of length k is* valid *iff it satisfies the following properties:*

(0) $M = \sigma_{p_k} \cdots \sigma_{p_{\ell+1}}(v)$ for some ℓ with $k > \ell \geq 0$, where all p_i, $k \geq i \geq \ell+1$, are different and all signatures in M are valid.

(1) If M arrived at p via the link from process q, then $p_k = q$.

(2) If $\ell = 0$, i.e., $M = \sigma_{p_k} \cdots \sigma_{p_1}(v)$, then $p_1 = t$ and v is a legitimate transmitter value.

(3) If $\ell > 0$, then the transmitter has not signed the message and $v = R_\emptyset^{p_\ell,\dots,p_1}$ (only for some forwarding schemes).

Although faulty processes and links can of course generate non-valid messages, their capabilities in doing so are considerably limited due to authentication. To prove this fact, we first summarize the extensions of our system/failure model due to to authentication:

—Any obedient process adheres to the particular algorithm at all times. Note that manifest faulty messages will usually be discarded at some point[17] and $\emptyset$ be used instead (recall hybrid majority in Section 5.1, for example).

—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 that arrive in the same round.

—An *arbitrary* faulty process may perform arbitrarily. In particular, it may forward multiple messages in the same round, collude with other arbitrary faulty processes, disclose its signature, etc.

—Links are incapable of generating valid signatures. However, an arbitrary faulty link could inject signed messages generated elsewhere in the system.

As an immediate corollary, Definition 6.1 in conjunction with (SA1)–(SA3) implies that neither faulty forwarding processes nor faulty links can introduce any v' (except $v' \in \{\emptyset, R_\emptyset\}$) different from the value v sent by a not arbitrary faulty transmitter p_1 with unbroken signature: Any valid signed message conveying such a v' must carry p_1's signature, which cannot happen if v' has not been sent by p_1.

Authentication thus allows us to set $f_\ell^{sa} = f_\ell^{ra} = 0$ in our system model, that is, arbitrary link failures need only be counted as omissive ones anymore. In addition,

[17]Note that this does not necessarily occur during forwarding, since checking all signatures is a time-consuming task.

the value X_p actually sent by a symmetric faulty process p should be a valid message (typically with incorrect content v), otherwise p would actually exhibit a manifest fault only. The same is true for arbitrary faulty processes, which would otherwise appear as omission faulty only.

In the remainder of this section, we will establish some general results that greatly simplify the analysis of the authenticated algorithms in the subsequent section. The first Lemma 6.2 shows that any information forwarded via paths with a common prefix that includes at least one not arbitrary faulty process with an unbroken signature is unique.

LEMMA 6.2 UNIQUENESS. *Consider any Byzantine agreement algorithm that forwards the transmitter t's value along paths of different processes, one hop per round. Let $M = \sigma_{p_k} \dots \sigma_{p_1}(v)$ resp. $M' = \sigma_{p'_{k'}} \dots \sigma_{p'_1}(v')$ 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., $p_1 = p'_1 = t, p_2 = p'_2, \dots, p_c = p'_c$. If at least one of those c processes is not arbitrary faulty and has an unbroken signature, then $v = v'$.*

Proof: Assuming the contrary, some obedient or symmetric faulty process p_x, $1 \leq x \leq c$, must have forwarded two different signed messages, namely, $M_x = \sigma_{p_x} \dots \sigma_{p_1}(v)$ and $M'_x = \sigma_{p'_x} \dots \sigma_{p'_1}(v')$, to its successor(s) on each path in the same round. Since both obedient and symmetric faulty processes may broadcast at most a single message per round according to Definition 3.1, p_x cannot have generated and sent both M_x and M'_x but at most one of those, say, M_x. Still, M'_x could have been generated out of thin air by an arbitrary link failure hitting the link from $p_x = p'_x$ to p'_{x+1} (with $p'_{k+1} = p'$). This is impossible according to (SA1), however, since M'_x can only be generated by process p_x as it involves σ_{p_x}. □

Of course, Lemma 6.2 does not rule out the possibility of receiving inconsistent information: Apart from different values disseminated by a Byzantine transmitter, processes p and/or p' might get $\emptyset$'s due to omissions or valid messages containing $R_\emptyset$'s (in case of algorithms like OMHA and ZA). Our lemma does not apply here, since M and M' must have a common prefix, which is impossible if either one (but not both) contains $R_\emptyset$. What is ensured by Lemma 6.2, however, is that the information routed via any two paths with a common prefix incorporating a single not arbitrary faulty process with an unbroken signature is the same. Consequently, if the common prefix of the forwarding path is long enough, i.e., $k \geq f_a + 1$, all receivers that obtain a valid non-$R_\emptyset$ message agree on the transmitter's value. We can prove an even stronger result, however:

LEMMA 6.3 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 transmitter's value along all possible paths of different processes. If an obedient process p receives some 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, $1 \leq x \leq k - \min\{1, f_\ell^s\}$ being the smallest index, is consistent and thus sends $M_x = \sigma_{p_x} \dots \sigma_{p_1}(v)$. Note that p_x cannot be manifest faulty here, since M would not carry σ_{p_x} in this case. Since p_x must have received v from itself in this case, 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 $n' \geq n - 1 - f_a - f_s - f_o - f_m - f_\ell^s > f_\ell^r + 1$ 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 6.3 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 could be extended to (strong) manifest processes but not to omission faulty ones: 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 only remains to address the question of how to model broken signatures. More specifically, since arbitrary faulty processes might already know all their signatures, we now assume that the signatures of at most f_b not arbitrary faulty processes are also "common knowledge".

Knowing a signature allows some malicious process to generate a signed message on behalf of any of the f_b processes, say, process p, even during forwarding. An attacker, sitting either on p or on an arbitrary faulty forwarding process q, could hence make p appear arbitrary faulty. However, Lemma 6.2 and 6.3 revealed that it cannot do more. Consequently, the problem of incorporating f_b additional broken signatures can be solved by simply increasing f_a to $f_a + f_b$.

7. HYBRID WRITTEN MESSAGES ALGORITHMS

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 6.3 that all non-faulty processes get the same set of values $\notin \{\emptyset, R_\emptyset\}$ 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 [Lamport et al. 1982], 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 subsections, we will provide a detailed analysis of a few alternative Hybrid Written Messages Algorithms.

7.1 Algorithm OMHA

In this section, we will analyze a variant of the algorithm OMHA developed in [Gong et al. 1995] under the system model of Sections 3 and 6. The original algorithm of [Gong et al. 1995] is the same as OMH except that every message sent in OMHA(m) with $m > 0$ must be signed. In our variant, messages must be signed in OMHA(0) as well. Moreover, we utilize the signature as the wrapper function R and report missing and manifest faulty messages by a distinguished value $R_\emptyset \neq \emptyset$. Note that $R_\emptyset$ is also the default value returned by hybrid-majority (see Section 5.1) if no majority exists.

DEFINITION 7.1 ALGORITHM OMHA [GONG ET AL. 1995]. *The Authenticated Hybrid Oral Message algorithm OMHA is defined recursively as follows:*

OMHA(0):

(1) *The transmitter t sends its signed value $w_t = \sigma_t(v)$ to every receiver and delivers $v_t = \sigma_t^{-1}(w_t) = v$, or $v_t = \emptyset$ if no or a manifest faulty w_t results from self-reception, recall Remark 2 on OMH's Definition 5.1.*

(2) *Every receiver p delivers $v_p = \sigma_t^{-1}(w_p)$ if w_p was received from the transmitter, or $v_p = \emptyset$ if no or a manifest faulty message was received.*

OMHA(m), $m > 0$:

(1) *The transmitter t sends its signed value $w_t = \sigma_t(v)$ to every receiver and delivers $v_t = \sigma_t^{-1}(w_t) = v$, or $v_t = \emptyset$ in case of no or a manifest faulty w_t.*

(2) *For every receiver p, let w_p be the value (= signed message) it obtains from the transmitter, or $R_\emptyset$ if no or a manifest faulty message was received.*
Every receiver p acts as the transmitter in algorithm OMHA($m-1$) to communicate 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 OMHA($m-1$) initiated by receiver q in step 2 above. Every receiver p calculates the hybrid-majority H_t of all values w_p^q and its own value $w_p^p = w_p$. If $H_t \neq R_\emptyset$, then $\sigma_t^{-1}(H_t)$ is delivered as v_p, else $\emptyset$ is delivered.*

As shown in the previous section, neither faulty processes nor links can generate new values, so the only values that may occur during the execution of OMHA are those originally sent by the transmitter, $\emptyset$, and $R_\emptyset$. Unfortunately, the existence of inconsistent values from arbitrary faulty transmitters in conjunction with $R_\emptyset$ values caused by arbitrary/omission faulty forwarding processes and links is enough to make the performance of OMHA no better than that of OMH — except that link failure tolerance is slightly improved.

LEMMA 7.2 VALIDITY OMHA. *For any $m \geq min\{1, f_\ell^s\}$ and any f_a, f_s, f_o, f_m, f_ℓ^s, $f_\ell^r \geq 0$, our algorithm OMHA(m) satisfies the validity property (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.*

THEOREM 7.3 VALIDITY & AGREEMENT OMHA. *For any $m \geq f_a + f_o + min\{1, f_\ell^s\}$ and any f_a, f_s, f_o, f_m, f_ℓ^s, $f_\ell^r \geq 0$, our 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.*

Proof: We showed in Section 6 that adding authentication amounts to restricting the behavior of faulty processes and links. Since OMHA can be mapped one-to-one to OMH with restricted failures, we can immediately re-use Lemma 5.3 and

Theorem 5.4 by plugging in modified parameter values. However, whereas ruling out arbitrary link failures yields some improvement over the non-authenticated case, there is no real gain with respect to process-failures: Arbitrary faulty transmitters may still send inconsistent information, and arbitrary faulty forwarding processes may inconsistently send (faulty) information and $R_\emptyset$. Hence, we cannot improve the numbers f_a, f_s etc. of process failures in the authenticated case.

Consequently, we only have to plug in $f_\ell^{sa} = f_\ell^{ra} = 0$ in Lemma 5.3 and Theorem 5.4 to obtain our results. □

Remarks:

(1) Note that OMHA does not depend critically upon authentication. In fact, it just degrades to OMH even if all signatures are broken, provided we spend f_ℓ^{ra} additional processes.

(2) We already mentioned that the original OMHA in [Gong et al. 1995] did not sign messages sent in OMHA(0). Recall that (A2) in Definition 3.1 assumes a point-to-point network where the transmitter of a message can be uniquely identified. If a link failure could only cause an omission or a manifest failure, Theorem 7.3 would remain valid for the original algorithm as well. However, if a link failure can substitute an $R_\emptyset$ value for the real message, then the original algorithm performs no better than OMH. Hence, by Theorem 5.4, we would need strictly more than $2f_\ell^s + f_\ell^r + f_\ell^{ra} + 2(f_a + f_s) + f_o + f_m + m$ processes for the original version of OMHA.

Since OMHA just adds signatures to OMH, both algorithms send and receive the same messages. Therefore, the results of OMH's assumption coverage analysis in Section 8, namely, Theorem 8.3 and 8.4, will remain valid for OMHA as well. Note carefully, however, that the numerical results in Tables III–VIII will not apply since they assume $n = 4f_\ell + 3m + 1$ and not OMHA's setting $n = 3f_\ell + 3m + 1$.

7.2 Algorithm ZA

In this section, we will analyze the authenticated algorithm ZA of [Gong et al. 1995] under the failure model of Definition 3.1. ZA has been derived from the flawed algorithm Z of [Thambidurai and Park 1988] and provides a much better resilience than OMHA. However, its correctness depends critically upon unbroken signatures of not arbitrary faulty processes (but recall the end of Section 6 for how to incorporate this).

DEFINITION 7.4 ALGORITHM ZA [GONG ET AL. 1995]. *Let $val(M)$ be a function that extracts the value v contained in a valid signed message $M = \sigma_{p_k} \dots \sigma_{p_1}(v)$, $k \geq 0$[18], by removing all signatures, or returns $val(M) = \emptyset$ if M is manifest faulty or $M = \emptyset$. The algorithm ZA is defined recursively as follows:*

ZA(0):

(*1*) *The transmitter t signs its value v and sends the resulting $w_t = \sigma_t(v)$ to every receiver. The transmitter delivers $v_t = val(w_t) = v$, or $v_t = \emptyset$ if no or a manifest faulty w_t was (self-)received, recall Remark 2 on OMH's Definition 5.1.*

[18] For $k = 0$, this means $val(v) = v$.

(2) *Every receiver p delivers $v_p = val(w_p)$, where w_p is the signed message received from the transmitter.*

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

(1) *The transmitter t sends its signed value $w_t = \sigma_t(v)$ to every receiver and delivers $v_t = val(w_t)$.*

(2) *For every process p, let w_p be the value p has obtained from the transmitter, or $\emptyset$ if no or a manifest faulty message has been received. Every receiver p acts as the transmitter in algorithm ZA($m-1$) to send w_p to all receivers.*

(3) *For every process p and $q \neq p$, let w_p^q be the (unsigned) value process p obtained from receiver q in step (2) using algorithm ZA($m-1$). Every receiver p delivers the hybrid-majority of all w_p^q and its own unsigned value $w_p^p = val(w_p)$; if no majority exists, $\emptyset$ is delivered.*

Unlike OMHA, which uses a distinguished reporting value $R_\emptyset$, ZA just uses $R_\emptyset = \emptyset$. Note that $\emptyset$ is also the default value returned by hybrid-majority in the absence of a majority. This implies that the only values that may occur during the execution of ZA are the value(s) sent by the transmitter and $\emptyset$. Consequently, there is no need for an "overwhelming" number of non-faulty values in order to compensate $R_\emptyset$ values. In fact, even a single non-$\emptyset$-value is sufficient here. The following Lemma 7.5 and Theorem 7.6 show that ZA satisfies validity (B2) and agreement (B1) with optimal resilience.

LEMMA 7.5 VALIDITY ZA. *For any $m \geq min\{1, f_\ell^s\}$ and any f_a, f_s, f_o, f_m, f_ℓ^s, $f_\ell^r \geq 0$, algorithm ZA(m) satisfies the validity property (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: Using induction, we show that every non-faulty process delivers the value sent by a consistent transmitter, and/or $\emptyset$-values in case of a manifest/omission faulty transmitter. Recall that considering those cases is sufficient for validity (B2). Lemma 6.2 ensures that no other value can be delivered; for brevity, we will not mention this explicitly in every step of our proof.

Induction starts at $m = 0$ in the absence of link failures, i.e., $f_\ell^s = f_\ell^r = 0$. In case of a consistent transmitter t sending ν, every non-faulty receiver will obtain $\sigma_t(\nu)$ in ZA(0) and will hence deliver ν (and so does the transmitter); a manifest faulty transmitter causes all receivers and itself to deliver $\emptyset$. In case of an omision faulty transmitter that sends either ν or causes $\emptyset$ to be delivered, every process will deliver either ν or $\emptyset$ in ZA(0) as required.

In case of $f_\ell^s > 0$, induction starts at $m = 1$. In ZA(1), a non-faulty or symmetric faulty transmitter signs and sends its value ν to all $n-1$ receivers, $n' \geq n - 1 - f_a - f_s - f_o - f_m \geq f_\ell^s + f_\ell^r + 1$ of which are non-faulty. In addition, the (non-faulty) transmitter delivers ν. At least $n' - f_\ell^s$ of the receivers will hence receive and broadcast ν in step 1 of ZA(0). Therefore, by the end of ZA(0), every non-faulty receiver gets ν from at least $n' - f_\ell^s - f_\ell^r \geq 1$ processes and will hence deliver it. Recall that any transmitter also uses its own value in step 3 of ZA(0), cf. Remark 2 on Definition 5.1. As soon as a process has obtained at least one ν as the result of ZA(0), however, it will deliver it as the result of ZA(1) as well.

If the transmitter in ZA(1) is manifest faulty, no receiver can get a valid w_p since such a message must contain t's signature. Hence, only $\emptyset$-values can be forwarded and delivered in ZA(0), and ZA(1) must return $\emptyset$ here as required. Finally, if the transmitter is omission faulty, any non-faulty process can only deliver ν or $\emptyset$ since these are the only possible values. Validity is hence fulfilled in this case as well.

For the induction step, we assume that our lemma holds for ZA$(m-1)$ with $m-1 \geq \min\{1, f_\ell^s\}$ and show that it also holds for ZA(m). Using the same argument as above, a non-faulty or symmetric faulty transmitter signs and sends its value ν to all $n-1$ receivers, $n' \geq n-1-f_a-f_s-f_o-f_m \geq f_\ell^s + f_\ell^r + 1$ of which are non-faulty. In addition, a (non-faulty) transmitter also delivers ν. In case of a manifest faulty transmitter, no receiver can get a valid w_p; so any receiver could forward only $\nu = \emptyset$. By the induction hypothesis, all non-faulty receivers will therefore deliver ν as the result of ZA$(m-1)$ and hence, due to hybrid-majority in conjunction with Lemma 6.2, as the result of ZA(m) as well. If the transmitter is omission faulty, any non-faulty process can only deliver ν or $\emptyset$ since these are the only valid values, so validity is fulfilled in this case as well. □

THEOREM 7.6 AGREEMENT AND VALIDITY ZA. *For any $m \geq f_a + f_o + min\{1, f_\ell^s\}$ and any f_a, f_s, f_o, f_m, f_ℓ^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.*

Proof: As argued in Remark 1 on Theorem 5.4, the proof is the same as that for agreement in OMH. □

Remarks:

(1) We slightly modified the original definition of ZA in [Gong et al. 1995], which lacked a few details. First, every transmitter in recursive instances must send its value to itself and use it as additional input to hybrid-majority; it does of course not further participate in the recursion. Secondly, our algorithm removes the whole signature chain in ZA(0) at once before delivery.

(2) Unlike OMHA, which requires signatures also in the final instance OMHA(0) to cope with link failures in the last round, ZA would work without it. In fact, in case of $f_\ell^s > 0$, any non-$\emptyset$ signed message submitted to ZA(0) in ZA's execution bears already $m \geq f_a + f_o + 1$ signatures, i.e., at least one from a consistent process. By Lemma 6.2, this is sufficient to ensure agreement — adding another signature within ZA(0) is hence not required.

(3) Although ZA is defined recursively like OMHA, its execution develops quite differently: According to Lemma 6.3, every process gets sufficient information to achieve validity within the first two rounds of ZA(m) in case of a not arbitrary faulty transmitter, regardless of the number of rounds actually employed. Validity of OMHA(m), however, is achieved only after its full number of $m+1$ rounds. Moreover, m needs to be included into n for OMHA, since faulty transmitters can inject $R_\emptyset$'s in additional rounds that must be balanced. This is not true for ZA, since the latter faces valid signed messages containing ν and $\emptyset$ only.

Obviously, apart from signatures, ZA sends and receives the same messages as OMH. The results of OMH's assumption coverage analysis in Section 8, namely, Theorem 8.3 and 8.4, will hence remain valid for ZA as well. Note carefully, however, that the numerical results in Tables III–VIII will not apply since they assume $n = 4f_\ell + 3m + 1$ and not ZA's minimum setting $n = 2f_\ell + m + 1$.

7.3 Algorithm ZAr

Even though ZA has a much better resilience than OMHA, it does not fully exploit the benefits of authenticated messages. More specifically, we know from Lemma 6.3 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 a specific 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 6.1 is hence void here.

The algorithm ZAr given in Definition 7.7 below is based upon those observations. It dramatically reduces both message and computational complexity of ZA, from exponential to polynomial.

DEFINITION 7.7 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 6.3, it is not difficult to show that ZAr satisfies validity and agreement:

LEMMA 7.8 VALIDITY ZAR. *For any $m \geq min\{1, f_\ell^s\}$ and any f_a, f_s, f_o, f_m, f_ℓ^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 6.3, 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 6.2 ensures that $\emptyset$ and v are the only possible values. Hence, validity is always fulfilled. □

Theorem 7.9 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_ℓ^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 7.8, 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 6.3. Note carefully that 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 6.3 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 7.7, 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.

As a final remark, we note that ZAr is in fact the same as the authenticated algorithm for atomic broadcasting under Byzantine failures proposed in [Cristian et al. 1985]. The latter was analyzed in a more abstract failure model, however, 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 Theorem 3.2 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 [Cristian et al. 1985] that it models link failures explicitly on a per-process-basis, rather than by their effect on the communication graph. In addition, whereas [Cristian et al. 1985] 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 5.4, 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 [Cristian et al. 1985] for tolerating f_o omission failures.

7.4 Broadcast Networks

Our system model in Definition 3.1 assumes a point-to-point network, which implies that the sender of a message is known even without authentication. This assumption (A2) is obviously not justified if oral messages algorithms like OMH were employed in systems with a broadcast network. Since faulty processes could impersonate non-faulty ones here, OMH would not work in this case. On the other hand, written messages algorithms should reasonably[19] work in broadcast networks because they prevent impersonation. In fact, even oral messages algorithm that achieve consensus under the system model of Definition 3.1 will achieve consensus in a broadcast network if authentication is added. After all, authentication ensures assumption (A2) even in such systems.

In fact, in the absence of link failures, written messages algorithms would benefit from a broadcast network, because neither arbitrary nor omission faulty processes are possible anymore. Since every process sends only one message, which is automatically broadcast to all processes, every receiver must get the same value. So we could in fact set $f_a = f_o = 0$ and count all arbitrary failures as symmetric failures and all omission failures as manifest failures for any written messages algorithm analyzed under the hybrid failure model.

If link failures are possible, however, we find that they have a lot more power in broadcast networks than before. Whereas they can simply be caught by adding an appropriate multiple of f_ℓ^r and f_ℓ^s to the number of processes in the point-to-point case, we experience the unpleasant effect that they make arbitrary (but not omission) failures possible in the broadcast case [Schmid 1995]. However, the behavior of arbitrary processes is restricted:

Consider a message from an arbitrary faulty process which is not received by f_ℓ^s receivers. If that process sends a second message containing a different value, which is not received by another f_ℓ^s receivers, then at most $2f_\ell^s$ receivers will only get one message and will assume that the message is valid. The other processes do detect the second message from the same sender and will use the value $\emptyset$ due to the manifest failure. So the obvious solution is either to count arbitrary failures again, or to count sender link failures twice, i.e., require $4f_\ell^s$ instead of $2f_\ell^s$ additional processes.

When analyzing OMHA in broadcast networks, we can exploit the restricted behavior of arbitrary processes. First, it is easily seen that the validity proof can

[19]Besides of the problem of jamming.

be taken over unchanged: Since the transmitter is not arbitrary faulty, and since arbitrary faulty processes do not occur in the last but one line of equations (13) and (14) in the proofs of Lemma 5.2 and Lemma 5.3, their different behavior in the broadcast network has no impact on validity and the proof of Lemma 5.3 still holds. As far as omission faulty processes are concerned, they either behave non-faulty or like manifest failures. In any case, all obedient receivers will deliver the same value for them.

THEOREM 7.10 VALIDITY & AGREEMENT. *For any f_a, f_s, f_o, f_m, f_ℓ^s, f_ℓ^r and any $m \geq \min\{1, f_\ell^s\}$, OMHA(m) satisfies agreement (B1) and validity (B2) if there are strictly more than $4f_\ell^s + f_\ell^r + 2(f_a + f_s) + f_o + f_m + m$ participating processes.*

Proof: Again, we only look at the case where the transmitter behaves arbitrary faulty as described above.

Let $m = 1$. Abbreviating the number of initially participating receivers with $n' \geq 4f_\ell^s + f_\ell^r + 2(f_a + f_s) + f_o + f_m + m$, the transmitter sends both ν and ν', which are received and turned into $\emptyset$ by at least $n' - 2f_\ell^s - (f_a - 1) - f_s - f_o' - f_m'$ non-faulty receivers, whereas at most f_ℓ^s processes receive only ν and f_ℓ^s only ν'. Here, $f_o' \leq f_o$ is the number of omission faulty receivers that will commit a manifest failure in OMHA(0) (all others will appear like non-faulty processes) and $f_m' \leq f_m$ is the actual number of manifest faulty processes.

In OMHA(0), a non-faulty receiver gets $R(\emptyset)$ from at least $n_q' = n' - 2f_\ell^s - (f_a - 1) - f_s - f_o' - f_m' - f_\ell^{r\prime}$ processes, with $f_\ell^{r\prime} \leq f_\ell^r$ link failures according to (A1^r), and it receives at most $n_q'' = n' - f_\ell^{r\prime} - f_o' - f_m'$ values different from $R(\emptyset)$. Therefore, we have

$$\begin{aligned} 2n_q' - n_q'' &= n' - 4f_\ell^s - 2f_a + 2 - 2f_s - f_o' - f_m' - f_\ell^{r\prime} \\ &\geq (f_\ell^r - f_\ell^{r\prime}) + (f_o - f_o') + (f_m - f_m') + m + 2 \\ &> 0 \end{aligned}$$

and $R(\emptyset)$ will win the hybrid-majority on every non-faulty process.

Let us now consider $m > \min\{1, f_\ell^s\}$. At least $\overline{n}_q' = n' - 2f_\ell^s - (f_a - 1) - f_s - f_o' - f_m'$ non-faulty receivers will get $\emptyset$ in OMHA(m) and thus will disseminate $R(\emptyset)$ in OMHA($m-1$). Validity ensures that every obedient process will deliver $R(\emptyset)$ for them. The f_o' omission faulty processes will appear crashed in OMHA($m-1$) and cause $\emptyset$. Therefore, every obedient process will deliver at most $\overline{n}_q'' = n' - f_o' - f_m'$ values different from $R(\emptyset)$. Since

$$\begin{aligned} 2\overline{n}_q' - \overline{n}_q'' &= n' - 4f_\ell^s - 2f_a - 2f_s - f_o' - f_m' \\ &\geq (f_o - f_o') + (f_m - f_m') + f_\ell^r + m \\ &> 0, \end{aligned}$$

$R(\emptyset)$ will again win the hybrid-majority on all obedient processes, hence all obedient processes will deliver $\emptyset$. □

The algorithm ZA, however, cannot exploit the different behavior of arbitrary failures so easily. Here, arbitrary faulty processes must still be counted in m. The reason is obvious from the proof of Theorem 7.10: We have utilized the fact that every non-faulty process receives enough $R(\emptyset)$ values to win the hybrid-majority.

This has saved us from using enough rounds to ensure that all processes get exactly the same input set for the hybrid-majority. In ZA, however, only $\emptyset$ does exist, which is not considered in hybrid-majority. Therefore, we will again have to ensure that all processes work with the same input set in ZA(0), effectively requiring $m \geq f_a + \min\{1, f_\ell^s\}$.

Remarks:

(1) Note that an arbitrary faulty process can do the worst damage by sending two messages. With a third message, again only f_ℓ^s receivers might not detect a manifest failure.

(2) When executing OMHA on a broadcast network, the tradeoff between the point-to-point algorithm of Theorem 7.3 and the broadcast version of Theorem 7.10 is $f_a + f_o$ vs. $2f_\ell^s$ additional processes and $f_a + f_o + 2$ vs. 2 rounds.

(3) Since ZA is already optimal and treats arbitrary failures like symmetric failures anyway, it will not really benefit from the broadcast network. There is, however, the slight tradeoff between the original version and the broadcast version with $2f_\ell^s$ more processes and f_o less rounds.

8. ASSUMPTION COVERAGE

To apply a deterministic failure model like the one of Definition 3.1 in practice, one has to address the question of assumption coverage: Given some particular implementation, what is the probability Q that the failure assumptions (f_a, f_s, f_o, f_m, f_ℓ^s, f_ℓ^r) are violated at runtime? In case of $f_a = 6$ and all other model parameters 0, for example, Q is the probability that more than 6 processes are faulty. Computing this probability of failure is a mandatory step for any algorithm where safety depends upon compliance to the failure model and is reasonably simple for process-centric failure models: Since the probability of a component failure is usually independent of the particular execution (except of its duration), it is not difficult to compute Q in this case.

Unfortunately, this is not true for our link failure assumptions. If individual link failures are independent, Q increases with every message broadcast during the execution of a distributed algorithm: According to ($A1^s$) resp. ($A1^r$), no message broadcast resp. reception may suffer from more than f_ℓ^s resp. f_ℓ^r link failures. Given the fact that typical consensus algorithms send many messages, the question arises whether Q can eventually be made as small as desired—by choosing suitable values of f_ℓ^s and f_ℓ^r—at all.

In this section, we will derive a bound on the probability of failure Q_m of the algorithm OMH(m)[20] of Section 5.1 in case of independent link failures. Since OMH has exponential message complexity, this may be seen as a worst case scenario for assumption coverage; polynomial algorithms like ZAr and the ones analyzed in [Biely and Schmid 2001] guarantee a considerably smaller probability of failure.

[20] Since OMH uses the same communications pattern as the authenticated algorithms OMHA of Section 7.1 and ZA of Section 7.2, Q_m is also valid for the latter exponential algorithms. The value of Q_m for the polynomial algorithm ZAr could be computed from the results developed in [Biely and Schmid 2001].

The expression for Q_m (see Theorem 8.3) will reveal that the probability of failure usually shrinks when f_ℓ^s, f_ℓ^r is increased, despite of the fact that n must be increased to maintain the required number of processes. Numerical results for a few parameter settings will show that Q_m can indeed be made arbitrarily small by sufficiently increasing n.

Our analysis is based upon a very simple probabilistic model of link failures, which assumes that the probability of losing or corrupting a single message on a single link or network interface is $0 < p < 1$, and that individual link failures occur independently of each other and of process failures. Note carefully, however, that we consider the assumption coverage with respect to link failures only: We do not consider possible violations of process failure assumptions in our analysis, i.e., we assume that the actual number of arbitrary, symmetric, omission and manifest faulty processes is always at most f_a, f_s, f_o and f_m, respectively. Link failures are hence viewed as additional incidents that happen to the messages sent on a link, irrespectively of whether they are correct messages from a non-faulty process or incorrect/missing ones from a faulty process.

Despite of its simplicity, this model is commonly used in practice, see e.g. [Eugster et al. 2001; Nikoletseas and Spirakis 1995], since it is analytically tractable and facilitates easy comparison of results. It is in fact a quite accurate and realistic model for uncorrelated transient channel/network interface failures in homogeneous system architectures. Persistent and, in particular, correlated failures are of course beyond its scope.

Our detailed analysis computes a bound on the probability of success $P_m = 1 - Q_m$ of the system during a single execution of OMH(m) as follows: We start with computing the binomial probability p_{n-k} of a successul (= model-conform) broadcast transmission/reception to/from $n-k$ peer processes. Next, we analyze the (exponential) communications requirements of OMH and compute the joint probability that all broadcasts/receptions during an execution are successful. This gives an expression for Q_m consisting of two sums involving p_{n-k}, which are evaluated in Lemma 8.1 and 8.2. This finally leads to the expression of Q_m given by Theorem 8.3.

Since $f_\ell^s = f_\ell^r = f_\ell$ is the only reasonable choice in presence of independent link failures, recall Remark 2 on Definition 3.1, the success probabilities for a single message broadcast/reception, namely,

$$\begin{aligned} p^s_{n-k} &= \mathbf{Prob}\{\leq f_\ell^s \text{ failures in a single broadcast to} \\ &\qquad\qquad n-k \text{ receivers}\} \\ p^r_{n-k} &= \mathbf{Prob}\{\leq f_\ell^r \text{ failures in a single reception from} \\ &\qquad\qquad n-k \text{ senders}\} \end{aligned}$$

for $0 \leq k \leq n-1$ are the same $p^s_{n-k} = p^r_{n-k} = p_{n-k}$ and follow a binomial distribution:

$$p_{n-k} = \sum_{l=0}^{f_\ell} \binom{n-k}{l} p^l (1-p)^{n-k-l} \tag{15}$$

The total *probability of success* $P_m = 1 - Q_m$ that there is no violation of our assumption of at most f_ℓ link failures in any message broadcast/reception during the execution of OMH(m) is given by

$$P_m = \textbf{Prob}\{\text{All broadcasts in OMH}(m),\ldots,\text{OMH}(1) \text{ have} \le f_\ell \text{ link failures} \\ \wedge \text{ all receptions in OMH}(0) \text{ have} \le f_\ell \text{ link failures each}\}. \tag{16}$$

Recall from the proof of Lemma 5.3 that ($A1^r$) in Definition 3.1 is required in the base case of the induction only, i.e., in OMH(0).

It is immediately apparent from step 1 of Definition 5.1 that the execution of OMH(m) evolves as shown in Table I.

OMH(m)	**# concurrent instances**	**# receivers**
m	1	$n-1$
$m-1$	$n-1$	$n-2$
$m-2$	$(n-1)(n-2)$	$n-3$
$\vdots$	$\vdots$	$\vdots$
1	$(n-1)\cdots(n-m+1)$	$n-m$
0	$(n-1)(n-2)\cdots(n-m)$	$n-m-1$

Table I. *Recursive instances in the execution of algorithm OMH.*

With the notation

$$\begin{aligned} [n]_k &= n(n-1)\ldots(n-k+1) \qquad \text{for } k > 0 \\ [n]_0 &= 1 \end{aligned}$$

it is apparent that, for $k < m$, there are $[n-1]_k$ instances of OMH($m-k$) that each issue a single broadcast [where ($A1^s$) applies] to $n-k-1$ receivers. For $k = m$, on the other hand, we have to consider message receptions [where ($A1^r$) applies] only: There are $[n-1]_m$ instances of OMH(0), and every receiver of a particular instance of OMH(0) should receive a message from all $n-m$ recipients in the prior instance OMH(1). Assuming that the "self-reception" by the transmitter of OMH(0) is always failure-free, there remain $n-m-1$ "true" message receptions by any receiver of OMH(0).

Abbreviating $n_k = [n-1]_k$, (16) translates to

$$\begin{aligned} P_m &= \prod_{k=0}^{m-1} p_{n-k-1}^{n_k} \cdot p_{n-m-1}^{n_m} = \prod_{k=0}^{m} p_{n-k-1}^{n_k} \\ &= \prod_{k=0}^{m} (1 - q_{n-k-1})^{n_k} \quad \text{with } q_{n-k} = 1 - p_{n-k} \\ &= \prod_{k=0}^{m} \Big(1 - \frac{n_k q_{n-k-1}}{n_k}\Big)^{n_k} \\ &\ge e^{-\sum\limits_{k=0}^{m} n_k q_{n-k-1}} \prod_{k=0}^{m} \Big(1 - \frac{(n_k q_{n-k-1})^2}{n_k}\Big), \end{aligned} \tag{17}$$

where we used the relation [Whittaker and Watson 1927, p. 242]

$$e^{-t} \geq (1-t/n)^n \geq e^{-t}(1-t^2/n) \tag{18}$$

valid for $t < n$; since q_{n-k-1} is some probability < 1, this condition is of course satisfied. Assuming

$$\sum_{k=0}^{m} n_k q_{n-k-1} < 1, \tag{19}$$

the application of the well-known facts (1) $\log(1-x) = -\sum_{j\geq 1} x^j/j$ for $|x| < 1$, (2) $\sum_{i\in I} a_i^j \leq \left(\sum_{i\in I} a_i\right)^j$ for $a_i \geq 0$ and integer $j \geq 1$, and (3) $e^{-x} \geq 1-x$ for $0 \leq x < 1$ yields

$$\begin{aligned}
P_m &\geq e^{-\sum_{k=0}^{m} n_k q_{n-k-1} + \sum_{k=0}^{m} \log\left(1 - \frac{\left(n_k q_{n-k-1}\right)^2}{n_k}\right)} \\
&\geq e^{-\sum_{k=0}^{m} n_k q_{n-k-1} - \sum_{k=0}^{m}\sum_{j\geq 1} \frac{(\sqrt{n_k} q_{n-k-1})^{2j}}{j}} \\
&\geq e^{-\sum_{k=0}^{m} n_k q_{n-k-1} - \sum_{j\geq 1} \frac{\left(\sum_{k=0}^{m} \sqrt{n_k} q_{n-k-1}\right)^{2j}}{j}} \\
&\geq \left(1 - \sum_{k=0}^{m} n_k q_{n-k-1}\right)\left(1 - \left(\sum_{k=0}^{m} \sqrt{n_k} q_{n-k-1}\right)^2\right) \\
&\geq 1 - \sum_{k=0}^{m} n_k q_{n-k-1} - \left(\sum_{k=0}^{m} \sqrt{n_k} q_{n-k-1}\right)^2 && (20) \\
&\geq 1 - \sum_{k=0}^{m} n_k q_{n-k-1} - \left(\sum_{k=0}^{m} n_k q_{n-k-1}\right)^2. && (21)
\end{aligned}$$

To obtain an upper bound on the overall probability of failure $Q_m = 1 - P_m$, we hence need an upper bound on

$$\sum_{k=0}^{m} [n-1]_k q_{n-k-1} = (n-1)! \sum_{k=0}^{m} \frac{q_{n-k-1}}{(n-k-1)!} \tag{22}$$

and, if the more accurate lower bound (20) is addressed,

$$\sum_{k=0}^{m} \sqrt{[n-1]_k} q_{n-k-1} = \sqrt{(n-1)!} \sum_{k=0}^{m} \frac{q_{n-k-1}}{\sqrt{(n-k-1)!}}. \tag{23}$$

The required bound for the dominating term (22) follows from the following Lemma 8.1.

LEMMA 8.1 UPPER BOUND. *For $n - m - f_\ell - 2 \geq 1$,*

$$G_m = \sum_{k=0}^{m} \frac{q_{n-k-1}}{(n-k-1)!}$$

$$\leq \left(1+\frac{1}{n-m-f_\ell-2}\right)\cdot\frac{q_{n-m-1}}{(n-m-1)!} \tag{24}$$

$$\leq \left(1+\frac{1}{n-m-f_\ell-2}\right)\cdot\frac{1}{(n-m-f_\ell-2)!}\cdot\frac{p^{f_\ell+1}}{(f_\ell+1)!}. \tag{25}$$

Proof: According to [Abramowitz and Stegun 1970, Eq. 26.5.24], q_{n-k} equals the incomplete Beta function $I_p(f_\ell+1, n-k-f_\ell)$, i.e.,

$$q_{n-k} = \sum_{l=f_\ell+1}^{n-k}\binom{n-k}{l}p^l(1-p)^{n-k-l} \tag{26}$$

$$= \frac{(n-k)!}{(f_\ell)!\,(n-k-f_\ell-1)!}\cdot\int_0^p t^{f_\ell}(1-t)^{n-k-f_\ell-1}\,dt. \tag{27}$$

Hence,

$$G_m = \frac{1}{(f_\ell)!}\int_0^p t^{f_\ell}\sum_{k=0}^{m}\frac{(1-t)^{n-k-f_\ell-2}}{(n-k-f_\ell-2)!}\,dt, \tag{28}$$

which involves

$$\begin{aligned}
S &= \sum_{k=0}^{m}\frac{(1-t)^{n-k-f_\ell-2}}{(n-k-f_\ell-2)!}\\
&= \frac{(1-t)^{n-m-f_\ell-2}}{(n-m-f_\ell-2)!}\Bigg(1+\frac{1-t}{n-m-f_\ell-1}+\\
&\qquad\qquad +\cdots+\frac{(1-t)^m}{(n-m-f_\ell-1)\cdots(n-f_\ell-2)}\Bigg)\\
&\leq \frac{(1-t)^{n-m-f_\ell-2}}{(n-m-f_\ell-2)!}\sum_{j=0}^{m}\left(\frac{1-t}{n-m-f_\ell-1}\right)^j\\
&\leq \frac{(1-t)^{n-m-f_\ell-2}}{(n-m-f_\ell-2)!}\cdot\frac{1}{1-\frac{1-t}{n-m-f_\ell-1}}\\
&\leq \frac{(1-t)^{n-m-f_\ell-2}}{(n-m-f_\ell-2)!}\cdot\frac{n-m-f_\ell-1}{n-m-f_\ell-2+t}\\
&\leq \frac{(1-t)^{n-m-f_\ell-2}}{(n-m-f_\ell-2)!}\cdot\left(1+\frac{1}{n-m-f_\ell-2}\right)
\end{aligned}$$

since $0 \leq t \leq p$. Plugging the above expression into (28), we obtain

$$G_m \leq \frac{1+\frac{1}{n-m-f_\ell-2}}{(f_\ell)!\,(n-m-f_\ell-2)!}\int_0^p t^{f_\ell}(1-t)^{n-m-f_\ell-2}\,dt \tag{29}$$

from where the major result (24) of our theorem follows by recalling (27).

To establish (25), we use the definition (26) of q_{n-k-1} to find

$$\begin{aligned}
g &= \frac{q_{n-m-1}}{(n-m-1)!}\\
&= \sum_{l=f_\ell+1}^{n-m-1}\frac{p^l}{l!}\cdot\frac{(1-p)^{n-m-l-1}}{(n-m-l-1)!}
\end{aligned} \tag{30}$$

$$
\begin{aligned}
&= \sum_{j=0}^{n-m-f_\ell-2} \frac{p^{j+f_\ell+1}}{(j+f_\ell+1)!} \cdot \frac{(1-p)^{n-m-f_\ell-2-j}}{(n-m-f_\ell-2-j)!} \\
&\le \frac{p^{f_\ell+1}}{(f_\ell+1)!} \sum_{j=0}^{n-m-f_\ell-2} \frac{p^j}{j!} \cdot \frac{(1-p)^{n-m-f_\ell-2-j}}{(n-m-f_\ell-2-j)!}.
\end{aligned}
$$

Applying the binomial theorem $(p+1-p)^{n-m-f_\ell-2} = 1$, we finally get

$$
\frac{q_{n-m-1}}{(n-m-1)!} \le \frac{1}{(n-m-f_\ell-2)!} \cdot \frac{p^{f_\ell+1}}{(f_\ell+1)!} \tag{31}
$$

which completes the proof of our lemma. □

Remarks:

(1) Lemma 8.1 reveals that the sum (22) is dominated by the term $k = m$, which just reflects the intuitively clear fact that the many messages from OMH(0) in the last round determine OMH(m)'s overall probability of failure.

(2) By subtracting $q_{n-m-1}/(n-m-1)!$ from both sides of (24), and multiplying by $(n-1)!$ according to (22), it is easy to see that (24) also implies monotonicity of

$$
\frac{q_{n-k-1}}{(n-k-1)!} \le \frac{q_{n-m-1}}{(n-m-1)!} \tag{32}
$$

$$
[n-1]_k q_{n-k-1} \le [n-1]_m q_{n-m-1} \tag{33}
$$

for any $0 \le k \le m$.

(3) The bound given by (25) is reasonably small—and also accurate, cp. the derivation starting with (30)— only if $np < 1$ is sufficiently small, since the ultimately required quantity $(n-1)!G_m$ that must be < 1 according to (19) has order $\mathcal{O}\big(n^m (np)^{f_\ell+1}/(f_\ell+1)!\big)$.

By a very similar proof, it is not difficult to prove a similar Lemma 8.2 related to the square-rooted sum (23). Since it is only used to improve the remainder $\mathcal{O}$-term in Theorem 8.3 below, its proof will be left as an exercise to the reader.

Lemma 8.2 Upper Bound $\sqrt{\ }$. *For $n-m-f_\ell-2 \ge 1$,*

$$
\begin{aligned}
H_m &= \sum_{k=0}^{m} \frac{q_{n-k-1}}{\sqrt{(n-k-1)!}} \\
&\le \Big(1 + \frac{1}{\sqrt{n-m-f_\ell-2}}\Big) \cdot \sqrt{\frac{[n-1]_{f_\ell+1}}{[n-m-1]_{f_\ell+1}}} \cdot \frac{q_{n-m-1}}{\sqrt{(n-m-1)!}} \\
&\le \Big(1 + \frac{1}{\sqrt{n-m-f_\ell-2}}\Big) \cdot \sqrt{\frac{[n-1]_{f_\ell+1}}{(n-m-f_\ell-2)!} \cdot \frac{p^{f_\ell+1}}{(f_\ell+1)!}}.
\end{aligned}
$$

□

By virtue of those results, we can establish the following Theorem 8.3.

THEOREM 8.3 ASSUMPTION COVERAGE OMH. *For $n - m - f_\ell - 2 \geq 1$ and $np < 1$ sufficiently small, the probability of failure Q_m of OMH(m) satisfies*

$$Q_m \leq Q'_m + \mathcal{O}\Big(\frac{(Q'_m)^2}{[n-f_\ell-2]_m}\Big) = \mathcal{O}\Big(n^m \frac{(np)^{f_\ell+1}}{(f_\ell+1)!}\Big), \tag{34}$$

where

$$Q'_m = \Big(1 + \frac{1}{n-m-f_\ell-2}\Big)[n-1]_{m+f_\ell+1}\frac{p^{f_\ell+1}}{(f_\ell+1)!}.$$

Proof: Recalling (22) resp. (23), the result of Lemma 8.1 resp. 8.2 immediately yields $(n-1)!G_m \leq Q'_m$ resp.

$$\begin{aligned} R''_m &= (n-1)!H_m^2 \\ &\leq \Big(1+\frac{1}{\sqrt{n-m-f_\ell-2}}\Big)^2 \cdot [n-1]_{f_\ell+1} \cdot [n-1]_{m+f_\ell+1} \cdot \Big(\frac{p^{f_\ell+1}}{(f_\ell+1)!}\Big)^2 \\ &\leq \mathcal{O}\Big(\frac{(Q'_m)^2}{[n-f_\ell-2]_m}\Big), \end{aligned} \tag{35}$$

where the last bound is easily confirmed by comparing R''_m with $(Q'_m)^2$. Recalling the lower bound (20) on the probability of success, (34) is established by straightforward upper bounding. Note that (20) is only guaranteed to hold when (19) holds, which is secured by $np < 1$ sufficiently small according to Remark 3 on Lemma 8.1.
□

In order to assess the dependency of the probability of failure Q_m upon the model parameters n, m, f_ℓ, we substitute $n = n_0 + c\ell$ in (34), where $c \cdot \ell$ gives the number of processes that must be added to cope with a (sufficiently small) number ℓ of additional link failures per process:

$$\begin{aligned} Q_m &= \mathcal{O}\left(n_0^m \frac{(n_0 p)^{f_\ell+\ell+1}}{(f_\ell+\ell+1)!}\big(1+\frac{c\ell}{n_0}\big)^{m+f_\ell+\ell+1}\right) \\ &= \mathcal{O}\Big(n_0^m \frac{(n_0 p)^{f_\ell+1}}{(f_\ell+1)!} \cdot \frac{(n_0 p \cdot e^c)^\ell}{[f_\ell+\ell+1]_\ell}\Big) \end{aligned} \tag{36}$$

In the last step, we employed the well-known relation $\big(1 + t/(k+j)\big)^k \leq e^t$ for $t \geq 0$, $j \geq 0$: Since we must have $n_0 - m - f_\ell - 2 \geq 1$ for Theorem 8.3, it follows that $k + j = n_0 \geq m + f_\ell + \ell + 1 = k$.

It is hence apparent from (36) that, as long as $np < 1$ sufficiently small, the probability of failure Q_m

—rapidly grows with m and hence with the number $f_a + f_o$ of arbitrary and omission failures,

—marginally grows with n and hence with the number of any kind of failures,

—decreases with the number of tolerated link failures f_ℓ (and with decreasing p, of course), since the last factor in (36) is < 1 for any suitably chosen ℓ.

Note carefully that the latter implies that increasing f_ℓ is always beneficial for reasonable parameter settings, which actually justifies our whole approach.

In Tables III–VI, we give numerical values for Q'_m for different values of m and f_ℓ in case of $n = 4f_\ell + 3m + 1$, which allows e.g. $f_\ell^s = f_\ell^r = f_\ell$, $f_a = m - 1$, $f_o = 0$, and $f_s = f_m = 1$ by Theorem 5.4.

f_ℓ	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$
1	8	11	14	17	20	23
2	12	15	18	21	24	27
3	16	19	22	25	28	31
5	24	27	30	33	36	39
7	32	35	38	41	44	47
10	44	47	50	53	56	59
15	64	67	70	73	76	79
20	84	87	90	93	96	99

Table II. *Value of $n = 4f_\ell + 3m + 1$ for different m, f_ℓ.*

f_ℓ	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$
1	0.64	1.	1.	1.	1.	1.
2	0.59	1.	1.	1.	1.	1.
3	0.52	1.	1.	1.	1.	1.
5	0.36	1.	1.	1.	1.	1.
7	0.22	1.	1.	1.	1.	1.
10	0.095	1.0	1.	1.	1.	1.
15	0.019	0.86	1.	1.	1.	1.
20	0.0036	0.37	1.	1.	1.	1.

Table III. *Value of (exact) probability of failure Q_m for $p = 0.1$.*

f_ℓ	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$
1	0.01	0.3	1	1	1	1
2	0.002	0.04	1	1	1	1
3	0.0002	0.006	0.3	1	1	1
5	2.10^{-6}	0.00009	0.005	0.3	1	1
7	2.10^{-8}	1.10^{-6}	0.00009	0.007	0.6	1
10	2.10^{-11}	2.10^{-9}	2.10^{-7}	0.00002	0.002	0.2
15	2.10^{-16}	2.10^{-14}	3.10^{-12}	4.10^{-10}	5.10^{-8}	8.10^{-6}
20	2.10^{-21}	2.10^{-19}	4.10^{-17}	7.10^{-15}	1.10^{-12}	2.10^{-10}

Table IV. *Value of (approximate) probability of failure Q'_m for $p = 0.01$.*

Whereas the probability of failure of OMH(m) given in Tables III–VI is not bad, even in case of a typical "wireless" loss probability $p = 0.01$, it is nevertheless clear that an algorithm that uses less messages is preferable with respect to our failure model. As an example, we consider the algorithm $\overline{\text{OMH}}$ that results from combining all messages that a process sends during OMH in a round into a single message. According to Table I, such a combined message consists of exactly $[n-1]_k/(n-k) =$

f_ℓ	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$
1	1.10^{-6}	0.00003	0.0009	0.03	1	1
2	2.10^{-9}	4.10^{-8}	2.10^{-6}	0.00007	0.004	0.2
3	2.10^{-12}	6.10^{-11}	3.10^{-9}	1.10^{-7}	7.10^{-6}	0.0004
5	2.10^{-18}	9.10^{-17}	5.10^{-15}	3.10^{-13}	2.10^{-11}	2.10^{-9}
7	2.10^{-24}	1.10^{-22}	9.10^{-21}	7.10^{-19}	6.10^{-17}	5.10^{-15}
10	2.10^{-33}	2.10^{-31}	2.10^{-29}	2.10^{-27}	2.10^{-25}	2.10^{-23}
15	2.10^{-48}	2.10^{-46}	3.10^{-44}	4.10^{-42}	5.10^{-40}	8.10^{-38}
20	2.10^{-63}	2.10^{-61}	4.10^{-59}	7.10^{-57}	1.10^{-54}	2.10^{-52}

Table V. *Value of (approximate) probability of failure Q'_m for $p = 0.0001$.*

f_ℓ	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$
1	1.10^{-10}	3.10^{-9}	9.10^{-8}	3.10^{-6}	0.0001	0.007
2	2.10^{-15}	4.10^{-14}	2.10^{-12}	7.10^{-11}	4.10^{-9}	2.10^{-7}
3	2.10^{-20}	6.10^{-19}	3.10^{-17}	1.10^{-15}	7.10^{-14}	5.10^{-12}
5	2.10^{-30}	9.10^{-29}	5.10^{-27}	3.10^{-25}	2.10^{-23}	2.10^{-21}
7	2.10^{-40}	1.10^{-38}	9.10^{-37}	7.10^{-35}	6.10^{-33}	5.10^{-31}
10	2.10^{-55}	2.10^{-53}	2.10^{-51}	2.10^{-49}	2.10^{-47}	2.10^{-45}
15	2.10^{-80}	2.10^{-78}	3.10^{-76}	4.10^{-74}	5.10^{-72}	8.10^{-70}
20	2.10^{-105}	2.10^{-103}	4.10^{-101}	7.10^{-99}	1.10^{-96}	2.10^{-94}

Table VI. *Value of (approximate) probability of failure Q'_m for $p = 0.000001$.*

$[n-1]_{k-1}$ single messages—corresponding to the instances of OMH$(m-k)$ at any of the $n-k$ originating processes—that are broadcast to $n-k-1$ receivers. Clearly, during the whole execution of $\overline{\text{OMH}}(m)$, any process broadcasts only m messages, except for the initial transmitter, which broadcasts only one message.

It is not difficult to show that the proofs of correctness for OMH are also valid for $\overline{\text{OMH}}$. In fact, the only difference lies in the fact that the receivers in $\overline{\text{OMH}}$ experience a link failure in a correlated fashion: If f_ℓ^s of the combined messages are lost in the broadcast of a single sender, any affected receiver loses the message for all instances of OMH$(m-k)$. This situation, however, could also occur when link failures are independent for all instances of OMH$(m-k)$.

By the same devices as used before, the probability of success $\overline{P}_m$ for $\overline{\text{OMH}}(m)$ evaluates to

$$\overline{P}_m \;=\; p_{n-1}\prod_{k=1}^{m} p_{n-k-1}^{n-k} \geq \prod_{k=0}^{m}\Big(1-\frac{(n-k)q_{n-k-1}}{n-k}\Big)^{n-k}$$

where the bound is even valid if all processes (and not only the initial transmitter) send an initial message in $\overline{\text{OMH}}(m)$. Due to that simplification, we just have to substitute $n_k = n-k$ in (21) and use the same line of reasoning as before to show the following Theorem 8.4.

Theorem 8.4 Assumption Coverage $\overline{\text{OMH}}$. *For $n-m-f_\ell-2 \geq 1$ and $np < 1$ sufficiently small, the probability of failure $\overline{Q}_m$ of $\overline{OMH}(m)$ satisfies*

$$\overline{Q}_m \leq \overline{Q}'_m + \mathcal{O}\big((\overline{Q}'_m)^2\big) = \mathcal{O}\Big(\frac{n}{f_\ell+3}\cdot\frac{(np)^{f_\ell+1}}{(f_\ell+1)!}\Big), \tag{37}$$

where

$$\overline{Q}'_m = \frac{[n+1]_{f_\ell+3} - [n-m]_{f_\ell+3}}{f_\ell+3} \cdot \frac{p^{f_\ell+1}}{(f_\ell+1)!}. \tag{38}$$

Proof: Applying (20) with $n_k = n-k$ reveals that $\overline{P}_m$ and hence the probability of failure $\overline{Q}_m$ is dominated by

$$\overline{Q}''_m = \sum_{k=0}^{m} (n-k) q_{n-k-1} = \sum_{k=0}^{m} (n-k)! \frac{q_{n-k-1}}{(n-k-1)!}, \tag{39}$$

Using the upper bound (31) with $m = k$ established in the proof of Lemma 8.1, we find

$$\begin{aligned}
\overline{Q}''_m &\le \sum_{k=0}^{m} \frac{(n-k)!}{(f_\ell+1)!\,(n-k-f_\ell-2)!} \cdot p^{f_\ell+1} \\
&\le (f_\ell+2) p^{f_\ell+1} \sum_{k=0}^{m} \binom{n-k}{f_\ell+2} \\
&\le (f_\ell+2) p^{f_\ell+1} \sum_{k=n-m}^{n} \binom{k}{f_\ell+2} \\
&\le (f_\ell+2) \left[\binom{n+1}{f_\ell+3} - \binom{n-m}{f_\ell+3} \right] p^{f_\ell+1} \\
&\le \frac{[n+1]_{f_\ell+3} - [n-m]_{f_\ell+3}}{f_\ell+3} \cdot \frac{p^{f_\ell+1}}{(f_\ell+1)!},
\end{aligned}$$

where we employed the well-known identity [Knuth 1973, p.54.(11)] $\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$. Recalling (21), which is again valid for $np < 1$ sufficiently small, and applying some simple majorizations on (38) that consider the fact that the coefficient of $n^{f_\ell+3}$ in both $[n+1]_{f_\ell+3}$ and $[n-m]_{f_\ell+3}$ is 1 and hence cancels out, (37) follows. □

Comparing (34) and (37) clearly shows that the probability of failure $\overline{Q}_m$ no longer grows with m. Tables VII and VIII contain a few numerical values for $\overline{Q}'_m$ for different values of m and f_ℓ and the same $n = 4f_\ell + 3m + 1$ used before, which ensures compatibility with Tables III and IV. We should note, however, that the messages sent by $\overline{\text{OMH}}$ are much larger than the ones of OMH—it is not really fair to consider the same values for the loss probability p here.

9. CONCLUSIONS

9.1 Accomplishments

In this paper, we showed that deterministic consensus in presence of link failures is possible—despite the impossibility result of [Gray 1978]—if the number of link failures in a broadcast resp. reception of any process is moderately restricted. We introduced a novel perception-based hybrid failure model for this purpose, which grants every process at most f_ℓ^r independent receive link failures (with at most f_ℓ^{ra} non-omission failures among those) and f_ℓ^s send link failures in each round,

f_ℓ	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$
1	0.88	1.	1.	1.	1.	1.
2	0.85	1.	1.	1.	1.	1.
3	0.80	0.99	1.	1.	1.	1.
5	0.62	0.93	1.	1.	1.	1.
7	0.42	0.77	0.96	1.	1.	1.
10	0.20	0.43	0.71	0.91	0.99	1.
15	0.041	0.10	0.21	0.37	0.58	0.78
20	0.0078	0.019	0.041	0.08	0.14	0.24

Table VII. *Value of (exact) probability of failure* $\overline{Q}_m$ *for* $p = 0.1$.

f_ℓ	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$
1	0.04	0.1	0.4	0.9	1.	1.
2	0.004	0.02	0.04	0.1	0.2	0.4
3	0.0004	0.002	0.004	0.01	0.02	0.04
5	5.10^{-6}	0.00002	0.00005	0.0001	0.0002	0.0005
7	5.10^{-8}	2.10^{-7}	5.10^{-7}	1.10^{-6}	3.10^{-6}	5.10^{-6}
10	5.10^{-11}	2.10^{-10}	4.10^{-10}	1.10^{-9}	3.10^{-9}	6.10^{-9}
15	4.10^{-16}	1.10^{-15}	4.10^{-15}	1.10^{-14}	2.10^{-14}	5.10^{-14}
20	4.10^{-21}	1.10^{-20}	3.10^{-20}	9.10^{-20}	2.10^{-19}	5.10^{-19}

Table VIII. *Value of (approximate) probability of failure* $\overline{Q}'_m$ *for* $p = 0.01$.

in addition to at most f_a, f_s, f_o, f_m arbitrary, symmetric, omission, and manifest process failures. Apart from dynamic link failures, our model can also be applied to incomplete communication graphs.

For $m \geq f_a + f_o + 1$, we analyzed three existing $m+1$-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 number of processes was shown to be

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

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

Algorithms specifically designed for written messages are hence clearly superior over algorithms adapted from an oral messages solution: Whereas OMHA did not profit much from authentication, ZA and ZAr benefit considerably — but also depend critically upon its strength. It turned out, however, that both algorithms can withstand intrusions to some extent: In case of broken signatures, OMHA degrades to OMH and hence requires an additional f_ℓ^{ra} in the number of processes. For ZA, a process with a compromised signature must be considered as arbitrary faulty and therefore counted in f_a. As far as link failures are concerned, authentication effectively prohibits arbitrary failures. Moreover, authentication is the only means to (more or less) safely employ algorithms like OMHA on top of broadcast networks.

We also provided a reasonably complete framework of impossibility results and lower bounds, which rests upon the necessity of unimpaired bidirectional communication for solving consensus. The number of required processes n and rounds $m+1$ in presence of both omission and arbitrary link failures were found to be

$$\begin{aligned} n &\geq f_\ell^s + f_\ell^{sa} + f_\ell^r + f_\ell^{ra} + 1, \\ m+1 &\geq f+2, \end{aligned} \tag{40}$$

where $f = f_o$ is the number of processes exhibiting crash failures. Our results also imply that a process must be counted as arbitrary (resp. omission) faulty only if it can disseminate incorrect information to a majority (resp. all) of the correct processes in the system.

Last but not least, we conducted an analysis of the assumption coverage of OMH, OMHA and ZA under a simple probabilistic model, where link failures occur with a fixed probability p independently of each other. We computed the probability Q of violating the link failure assumption $f_\ell^r = f_\ell^s = f_\ell$, which shows that our approach of adding processes in order to tolerate additional link failures per process always decreases Q as long as $np < 1$ sufficiently small. Consequently, for reasonably small m, our algorithms can be used even in wireless systems, where link failures with loss probabilities up to $p = 10^{-2}$ are the dominating source of errors. Given the limited bandwidth usually available in wireless systems, the excessive communication requirements of exponential algorithms may be prohibitive, though.

9.2 Costs of Tolerating Link Failures

Our results allow a comparison of the costs for tolerating link failures of our algorithms. Table IX summarizes the relevant figures for $f_\ell^s = f_\ell^r = f_\ell$: The second resp. third column gives the number of processes required for tolerating f_ℓ omission resp. f_ℓ arbitrary link failures; 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 $f_\ell = 0$.

Algorithm	**omissions**	**arbitrary**	**# rounds**
OMH	$n_0 + 3f_\ell$	$n_0 + 4f_\ell$	$m_0 + 1$
OMHA	$n_0 + 3f_\ell$	$n_0 + 3f_\ell$	$m_0 + 1$
ZA, ZAr	$n_0 + 2f_\ell$	$n_0 + 2f_\ell$	$m_0 + 1$

Table IX. *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.*

All our algorithms match the lower bound for the required number of rounds and are hence optimal in this respect. As far as the required number of processes is concerned, the best algorithms ZA and ZArneed only $2f_\ell$ additional processes to cope with $f_\ell \cdot n$ link failures per process in each round; for $f_\ell = 1$, only $n = 4$ processes are required in the absence of process failures, for example. Both algorithms thus match the lower bound $n > 2f_\ell$ for omission link failures[21] and are hence optimal. 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 not for $f_\ell^{sa} < f_\ell^s$, cf. Remark 6 on Theorem 5.4.

Since f_ℓ could be as much as $\mathcal{O}(n)$, our algorithms tolerate up to $\mathcal{O}\big((m+1)n^2\big)$ link failures during the whole execution. 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 note, though, that this does not mean that our algorithms 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.

Moreover, the bound (36) on the probability Q_m of violating the failure model reveals that adding sufficiently many processes is always beneficial as long as $np < 1$ is sufficiently small. In this case, the disadvantage of increasing the number of links that could be faulty is more than compensated by the ability to mask additional link failures per process. This ultimately confirms that

(1) limiting the power of link failures according to our failure model is not an undue restriction,

(2) our algorithms can even be employed in wireless systems, where link failure probabilities p up to 10^{-2} are common,

which was the ultimate goal for starting this research at all.

9.3 Further Research

There are several directions of current/future research in this area: For example, we analyzed several less message-costly consensus algorithms under our perception-based hybrid failure model in [Biely and Schmid 2001]. Since those provide sub-optimal link failure tolerance, we are currently working on a polynomial algorithm that is optimal in this respect. A more theoretical direction of future work would be an extension of our lower bound results to both process and link failures. Another important part of our current research, where we already made some progress [Schmid and Fetzer 2002], are consensus algorithms for asynchronous systems in presence of link failures. Last but not least, there are many applications like intrusion-tolerant admission control, on-line diagnosis, distributed database commitment, etc. that could benefit from the possibility to achieve consensus in systems with link failures. Part of our future research will hence be devoted to the exploration of such applications.

ACKNOWLEDGMENTS

We are grateful to Martin Biely for his comments on an earlier version of this paper.

[21] Note that the transmitter in a Byzantine agreement algorithm should be neglected when comparing the number of processes with the consensus lower bound, cf. Remark 1 on Definition 5.1.

REFERENCES

ABRAMOWITZ, M. AND STEGUN, I. A. 1970. *Handbook of Mathematical Functions*. Dover Publications, Inc., New York.

ABU-AMARA, H. AND LOKRE, J. 1994. Election in asynchronous complete networks with intermittent link failures. *IEEE Transactions on Computers 43,* 7 (July), 778–788.

AFEK, Y., ATTIYA, H., FEKETE, A., FISCHER, M., LYNCH, N., MANSOUR, Y., WANG, D.-W., AND ZUCK, L. 1994. Reliable communication over unreliable channels. *Journal of the ACM (JACM) 41,* 6, 1267–1297.

AGUILERA, M. K., CHEN, W., AND TOUEG, S. 2000. Failure detection and consensus in the crash-recovery model. *Distributed Computing 13,* 2 (Apr.), 99–125.

AGUILERA, M. K. AND TOUEG, S. 1998. A simple bivalency proof that t-resilient consensus requires t+1 rounds. Tech. Rep. TR98-1701, Department of Computer Science, Cornell University. September.

ATTIYA, H. AND WELCH, J. 1998. *Distributed Computing.* McGraw-Hill.

AZADMANESH, M. AND KIECKHAFER, R. M. 2000. Exploiting omissive faults in synchronous approximate agreement. *IEEE Transactions on Computers 49,* 10 (Oct.), 1031–1042.

AZADMANESH, M. H. AND KIECKHAFER, R. M. 1996. New hybrid fault models for asynchronous approximate agreement. *IEEE Transactions on Computers 45,* 4, 439–449.

BASU, A., CHARRON-BOST, B., AND TOUEG, S. 1996. Crash failures vs. crash + link failures. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing.* ACM Press, 246.

BIELY, M. AND SCHMID, U. 2001. Message-efficient consensus in presence of hybrid node and link faults. Tech. Rep. 183/1-116, Department of Automation, Technische Universität Wien. August. (submitted).

CHARRON-BOST, B. AND SCHIPER, A. 2000. Uniform consensus harder than consensus. Tech. Rep. DSC/2000/028, École Polytechnique Fédérale de Lausanne, Switzerland. May.

CRISTIAN, F., AGHILI, H., STRONG, R., AND DOLEV, D. 1985. Atomic broadcast: From simple message diffusion to byzantine agreement. In *Proceedings 15th Int. Conf. on Fault-Tolerant Computing (FTCS-15).* Ann Arbor, Michigan, USA, 200–206.

CRISTIAN, F. AND FETZER, C. 1994. Fault-tolerant internal clock synchronization. In *Proceedings of the Thirteenth Symposium on Reliable Distributed Systems.* Dana Point, Ca., 22–31.

EUGSTER, P. T., GUERRAOUI, R., HANDURUKANDE, S., KERMARREC, A.-M., AND KOUZNETSOV, P. 2001. Lightweight probabilistic broadcast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01).* Göteborg, Sweden, 443–452.

FISCHER, M., LYNCH, N., AND MERRITT, M. 1986. Easy impossibility proofs for the distributed consensus problem. *Distributed Computing 1,* 1, 26–39.

FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. 1985. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM 32,* 2 (Apr.), 374–382.

GAFNI, E. 1998. 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, 143–152.

GONG, L., LINCOLN, P., AND RUSHBY, J. 1995. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Proceedings Dependable Computing for Critical Applications-5.* Champaign, IL, 139–157.

GRAY, J. N. 1978. Notes on data base operating systems. In *Operating Systems: An Advanced Course*, G. S. R. Bayer, R.M. Graham, Ed. Lecture Notes in Computer Science, vol. 60. Springer, New York, Chapter 3.F, 465.

GRIDLING, G. 2002. An algorithm for three-process consensus under restricted link failures. Tech. Rep. 183/1-123, Department of Automation, Technische Universität Wien. Oct.

HADZILACOS, V. 1987. Connectivity requirements for Byzantine agreement under restricted types of failures. *Distributed Computing 2*, 95–103.

HADZILACOS, V. AND TOUEG, S. 1993. Fault-tolerant broadcasts and related problems. In *Distributed Systems*, 2nd ed., S. Mullender, Ed. Addison-Wesley, Chapter 5, 97–145.

Knuth, D. E. 1973. *Fundamental Algorithms*, 2nd ed. The Art of Computer Programming, vol. 1. Addison-Wesley, Reading, Massachusetts.

Lamport, L., Shostak, R., and Pease, M. 1982. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems 4,* 3 (July), 382–401.

Lincoln, P. and Rushby, J. 1993. A formally verified algorithm for interactive consistency under a hybrid fault model. In *Proceedings Fault Tolerant Computing Symposium 23.* Toulouse, France, 402–411.

Lynch, N. 1996. *Distributed Algorithms.* Morgan Kaufman.

Meyer, F. J. and Pradhan, D. K. 1987. Consensus with dual failure modes. In *In Digest of Papers of the 17th International Symposium on Fault-Tolerant Computing.* Pittsburgh, 48–54.

Nikoletseas, S. E. and Spirakis, P. G. 1995. Expander properties in random regular graphs with edge faults. In *12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95).* Munich, Germany, 421 – 432.

Perry, K. J. and Toueg, S. 1986. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering SE-12,* 3 (March), 477–482.

Pinter, S. S. and Shinahr, I. 1985. Distributed agreement in the presence of communication and process failures. In *Proceedings of the 14th IEEE Convention of Electrical & Electronics Engineers in Israel.* IEEE.

Powell, D. 1992. Failure mode assumptions and assumption coverage. In *Proc. 22nd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-22).* Boston, MA, USA, 386–395. (Revised version available as LAAS-CNRS Research Report 91462, 1995).

Reischuk, R. 1985. A new solution for the Byzantine generals problem. *Information and Control 64,* 1–3 (January–March), 23–42.

Rushby, J. 1994. A formally verified algorithm for clock sychronization under a hybrid fault model. In *Proceedings ACM Principles of Distributed Computing (PODC'94).* Los Angeles, CA, 304–313.

Rushby, J. 2001. Formal verification of hybrid Byzantine agreement under link faults. Tech. rep., Computer Science Laboratory, SRI International, Menlo Park, CA. Available at http://www.csl.sri.com/~rushby/abstracts/byzlinks01.html.

Santoro, N. and Widmayer, P. 1989. Time is not a healer. In *Proc. 6th Annual Symposium on Theor. Aspects of Computer Science (STACS'89).* LNCS 349. Springer-Verlag, Paderborn, Germany, 304–313.

Sayeed, H. M., Abu-Amara, M., and Abu-Amara, H. 1995. Optimal asynchronous agreement and leader election algorithm for complete networks with Byzantine faulty links. *Distributed Computing 9,* 3, 147–156.

Schmid, U. 1994. Synchronized UTC for distributed real-time systems. In *Proceedings 19th IFAC/IFIP Workshop on Real-Time Programming (WRTP'94).* Lake Reichenau, Germany, 101–107.

Schmid, U. 1995. Synchronized Universal Time Coordinated for distributed real-time systems. *Control Engineering Practice 3,* 6, 877–884. (Reprint from [Schmid 1994]).

Schmid, U. 2000. Orthogonal accuracy clock synchronization. *Chicago Journal of Theoretical Computer Science 2000,* 3, 3–77.

Schmid, U. 2001. 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, 57–66.

Schmid, U. and Fetzer, C. 2002. Randomized asynchronous consensus with imperfect communications. Tech. Rep. 183/1-120, Department of Automation, Technische Universität Wien. January. (submitted).

Schmid, U. and Schossmaier, K. 2001. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing 14,* 2 (May), 101 – 111.

Schmid, U. and Weiss, B. 2001. Consensus with oral/written messages: Link faults revisited. Tech. Rep. 183/1-110, Department of Automation, Technische Universität Wien. Feb. (obsolete, replaced by TR 183/1-124).

Schmid, U., Weiss, B., and Rushby, J. 2002. Formally verified byzantine agreement in presence of link faults. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*. 608–616.

Singh, G. 1996. Leader election in the presence of link failures. *IEEE Transactions on Parallel and Distributed Systems 7,* 3 (Mar.), 231–236.

Siu, H.-S., Chin, Y.-H., and Yang, W.-P. 1998. Byzantine agreement in the presence of mixed faults on processors and links. *IEEE Transactions on Parallel and Distributed Systems 9,* 4 (Apr.), 335–345.

Srikanth, T. K. and Toueg, S. 1987. Optimal clock synchronization. *Journal of the ACM 34,* 3 (July), 626–645.

Tel, G. 1994. *Introduction to Distributed Algorithms.* Cambridge University Press.

Thambidurai, P. M. and Park, Y. K. 1988. Interactive consistency with multiple failure modes. In *Proceedings 7th Reliable Distributed Systems Symposium.*

Varghese, G. and Lynch, N. A. 1992. A tradeoff between safety and liveness for randomized coordinated attack protocols. In *Proceedings of the 11th Annual ACM Symposium on Pprinciples of Distributed Computing.* Vancouver, British Columbia, Canada, 241–250.

Walter, C. J., Lincoln, P., and Suri, N. 1997. Formally verified on-line diagnosis. *IEEE Transactions on Software Engineering 23,* 11 (Nov.), 684–721.

Walter, C. J. and Suri, N. 2002. The customizable fault/error model for dependable distributed systems. *Theoretical Computer Science 290*, 1223–1251.

Walter, C. J., Suri, N., and Hugue, M. M. 1994. Continual on-line diagnosis of hybrid faults. In *Proceedings DCCA-4.*

Weiss, B. and Schmid, U. 2001. Consensus with written messages under link faults. In *20th Symposium on Reliable Distributed Systems (SRDS'01).* 194–197.

Whittaker, E. and Watson, G. 1927. *A Course of Modern Analysis.* Cambridge University Press, Cambridge.

Submitted November 2002