Randomized Asynchronous Consensus with Imperfect Communications

Ulrich Schmid*
Technische Universität Wien
Embedded Computing Systems Group (E182/2)
Treitlstraße 3, A-1040 Vienna
s@auto.tuwien.ac.at

Christof Fetzer
AT&T Labs-Research
Dependable Distributed Computing Department
180 Park Ave, Florham Park, NJ07932
christof@research.att.com

Abstract

We introduce a novel hybrid failure model, which facilitates an accurate and detailed analysis of round-based synchronous, partially synchronous and asynchronous distributed algorithms under both process and link failures. Its utility and expressiveness is demonstrated by means of a complete analysis of the well-known randomized Byzantine agreement algorithm of (Srikanth & Toueg 1987). Granting every process in the system up to f_ℓ link failures (with f_ℓ^a arbitrary faulty ones among those) in every round, without being considered faulty, we show that this algorithm needs just $n \geq 4f_\ell + 2f_\ell^a + 3f_a + 1$ processes for tolerating f_a Byzantine process failures. The probability of disagreement after R iterations is only 2^{-R}, which is the same as in the FLP model and thus much smaller than the lower bound $\mathcal{O}(1/R)$ known for synchronous systems with lossy links. We also provide a detailed analysis of the algorithm's running time, as well as an evaluation of the model's assumption coverage in systems with transient link failures. Finally, stubborn links are shown to be sufficient for this algorithm. In accordance with our findings for synchronous systems obtained elsewhere, our results reveal that randomized Byzantine agreement can be solved efficiently even in asynchronous systems with imperfect communications. Contrasting widespread believe, there is no need to employ a perfect communications subsystem even in case of excessive link failure rates.

Keywords: *Fault-tolerant distributed systems, partially synchronous systems, failure models, link failures, randomized Byzantine agreement, consensus, stubborn links.*

*Supported by the Austrian START programme Y41-MAT, in the context of our W2F-project *http://www.auto.tuwien.ac.at/Projects/W2F/*

1 Motivation

Traditionally, most asynchronous fault-tolerant algorithms have been designed for process failures only. A typical assumption is that at most f of the n processes in a distributed system may be Byzantine faulty during the entire execution. Link failures can also be handled within such models as long as link failure rates are low. If link failure rates are high, however, one cannot count link-related failures as sender or receiver process failures without quickly running out of non-faulty processes [22, 23].

Most failure models for asynchronous systems hence stipulate [12] or simulate perfect [1, 2, 4, 5] links, i.e., can assume that any message sent by a correct process to any other correct process will eventually be delivered. Although many distributed protocols have been designed for reliable channels, this abstraction inevitably needs unbounded memory and stable storage or, equivalently, UIDs in presence of crashes [11, 19]. Consequently, in wide area and wireless networks, where long lasting link breaks and non-FIFO behavior are common, the perfect communications assumption of the Fischer, Lynch & Paterson (FLP) model [12] is difficult to justify.

As the first contribution of this paper, we introduce[1] a hybrid failure model that incorporates both process and link failures in both time and value domain. It is applicable to any round-based distributed algorithm and consists of a basic *physical failure model* facilitating assumption coverage and timing analysis, and a more abstract round-by-round *perception failure model* facilitating accurate fault-tolerance analysis. Formalized for partially synchronous systems [10] where delays are possibly unknown, our model covers synchronous and asynchronous systems as limiting cases and is hence widely applicable. Unlike existing approaches based upon stabilization [8] or round-by-round fault detectors [13], it can handle Byzantine failures as well.

[1] This paper contains only a brief overview of our failure model; a forthcoming journal paper will provide all the details.

We will demonstrate the utility of our failure model by analyzing a hybrid version of the randomized Byzantine agreement algorithm of [25, 26] under Byzantine process and link failures. For synchronous systems, it has long been known that randomized algorithms—unlike deterministic ones, see [22, 23]—can solve consensus in presence of (unconstrained) lossy links [19]. There is a fairly large lower bound $1/(R+1)$ for the probability of disagreement after R rounds [19, Thm. 5.5], however. For asynchronous systems under the FLP model, the algorithm of [26] has a probability of disagreement of at most 2^{-R}. The question is: Does this algorithm still work if one drops the perfect link assumption and, if so, is there a penalty?

In this paper, we will show the following:

(1) *Every* process may suffer from f_ℓ additional lost (faulty) messages sent and received via arbitrary links per round, without being considered faulty, provided that n is increased by $4f_\ell$ ($6f_\ell$). Time redundancy typically used for implementing perfect communication can hence be replaced by resource redundancy (more processes).

(2) The lower bound $\mathcal{O}(1/R)$ on the probability of disagreement of randomized consensus with unconstrained lossy links after R rounds does not hold in our model. Our algorithm actually provides the same probability 2^{-R} as in the FLP model.

(3) Tolerating link failures considerably decreases the algorithm's running time, since tighter end-to-end delay bounds for communication between correct processes can be used.

(4) With very reasonable assumption coverage, unreliable datagram communication can be employed in systems with transient link failure rates up to 10^{-2}.

(5) For even higher link failure rates, 2-stubborn links [15] (guaranteeing reliable delivery of the last two messages) can be used. Unlike reliable communication channels, they do not suffer from the unbounded memory requirements of ever growing "unacknowledged message" queues.

The remaining sections of our paper are organized as follows: In Section 2, we will very briefly introduce our failure model. Section 3 contains the analysis of hybrid versions of two well-known reliable broadcast primitives, which are the major building blocks of the randomized consensus algorithm of [25, 26]. Its analysis in Section 4 consists of a proof of correctness (Section 4.1), a detailed runtime analysis (Section 4.2), a coverage analysis (Section 4.3), and the justification of using stubborn channels (Section 4.4). A short summary of our accomplishments in Section 5 concludes the paper.

2 Failure Model

This section contains a very brief overview of our failure model. It consists of an execution model, a basic physical failure model, and a more abstract perception failure model. Both the physical and the perception failure model are *hybrid* ones [3, 27], i.e., distinguish several classes of failures. The advantage of a hybrid failure model is its improved resilience: Less severe failures can usually be handled with fewer processes than more severe ones. Obviously, an algorithm's resilience in a standard model (like all-Byzantine) is easily obtained by setting some model parameters to 0.

Due to lack of space, we will entirely omit the description of the *physical failure model*, which is an extension of the model of [21]. It distinguishes several classes of time and value failures for both processes and links, and uses assertions like "at most ϕ_{av} processes may behave Byzantine". Due to the exploding number of possible combinations of time and value failures, it is not used for analyzing fault-tolerant algorithms, however. Its primary purpose is the analysis of the assumption coverage [17] in real systems, cp. Section 4.3.

The physical failure model can be reduced to a more abstract (and vastly simpler) *perception failure model*, which is similar in spirit to the round-by-round fault detector approach of [13]. It is a generalization of the synchronous model of [22, 23], and is solely based upon the local view (= perception of failures) of every process in any round. The perception failure model is particularly well-suited for analyzing the fault-tolerance properties of distributed algorithms.

2.1 Execution Model

We consider a distributed system of n *processors* connected by a fully or partially connected point-to-point network. All links between processors are bidirectional, consisting of two unidirectional channels that may be hit by failures independently. The system will execute a distributed round-based algorithm made up of one or more concurrent *processes* at every processor. Any two processes at different processors can communicate bidirectionally with each other via the interconnecting links. Every processor is identified by a unique *processor id* $p \in \{1, \ldots, n\}$; every process is uniquely identified system-wide by the tuple (processor id, process name), where the *process name* N is chosen from a suitable name space. Since a process will usually communicate with processes of the same name, we will distinguish processes primarily by their processor ids and suppress process names when they are clear from the context. Note carefully, however, that our model allows sender and receiver process to have different names.

Since we restrict our attention to round-based algorithms, all processes execute a finite or infinite sequence of consecutive *rounds* $R = 0, 1, \ldots$. In every round except the initial one $R = 0$, which is slightly different, a single process p may broadcast (= successively send) a single message—containing the current *round number* R and a *value* V_p^R depending upon its local computation—to all processes contained in p's current *receiver set* $\mathcal{R}_p^R \subseteq \{1, \ldots, n\}$.[2] We assume that every (non-faulty) receiver q knows its current *sender set* $\mathcal{S}_q^R = \{p : q \in \mathcal{R}_p^R\}$ containing all the processes that should have sent a message to it, and that a process satisfying $p \in \mathcal{R}_p^R$ (and hence $p \in \mathcal{S}_p^R$) sends a message to itself as well. Note that this convention does not prohibit an efficient direct implementation of self-reception, provided that the resulting end-to-end transmission delay is taken into account properly.

Concurrently, for every round number S, process p receives incoming round S messages from the processes $\in \mathcal{S}_p^S$ and collects their values in a local array (subsequently called *perception vector*) $\mathcal{V}_p^S = \{V_p^{1,S}, \ldots, V_p^{n,S}\}$. Note that $\mathcal{V}_p^S = \mathcal{V}_p^S(t)$ as well as its individual entries $V_p^{i,S} = V_p^{n,S}(t)$ are actually time-dependent; we will usually suppress t, however, in order not to overload our notation. Storing a single value for each peer process in the perception vector is sufficient, since any receiver may get at most one round S message from any non-faulty sender. The entry $V_p^{q,S} \in \mathcal{V}_p^S$ (subsequently called *perception*) is either $\emptyset$ if no round S message from process q came in yet (or if $q \notin \mathcal{S}_p^S$), or it contains the received value from the first round S message from process q. In case of multiple messages from the same sender q, which must be faulty, the receiver could also drop all messages and set $V_p^{q,S}$ to some obviously faulty value, instead of retaining the value from the first message.

Process p's current round R is eventually terminated at the *round switching time* σ_p^R, which is the real-time when process p switches from round R to the next round $R+1$. Note that round switching is event-based—and part of the particular algorithm—in case of asynchronous systems but enforced externally in case of synchronous systems. At the round switching time, the value $V_p^{R+1} = F_p\big(\mathcal{V}_p^R(\sigma_p^R), \Sigma_p\big)$ to be broadcast by process p in the next round $R+1$ is computed as a function F_p of the round R perceptions available in $\mathcal{V}_p^R = \mathcal{V}_p^R(\sigma_p^R)$ and p's local state Σ_p at time σ_p^R.

Formally, the essentials of the above execution pattern are captured by two specific events: $be_p^R = V_p^R[t_p^R]$ is process p's round R *broadcast event*, whereas $pe_q^{p,R} = V_q^{p,R}[t_q^{p,R}]$ denotes process q's *perception event* of process p's broadcast event. Those events are related via their parameter values $V_q^{p,R} = V_p^R$ (which are equal if there is no failure) and their occurrence times $t_q^{p,R} = t_p^R + \delta_q^{p,R}$, where $\delta_q^{p,R}$ is the *end-to-end computational + transmission delay* between sender p and receiver q in round R. Note that $\delta_q^{p,R}$ includes any round R computation at the sender and receiver process, in particular, $F_p\big(\mathcal{V}_p^R(\sigma_p^R), \Sigma_p\big)$.

Our model stipulates lower and upper bounds $\tau^- \geq 0$ and $\tau^+ \leq \infty$[3], not necessarily known to the algorithm, such that

$$\tau^- \leq \delta_q^{p,R} \leq \tau^+ \tag{1}$$

for any two well-behaved processes p, q connected by a non-faulty link. Note that this relation must be valid for any round R and $p = q$ as well.[4] Introducing the interval $\boldsymbol{\tau} = [\tau^-, \tau^+]$, the above relation (1) can be written concisely as $\delta_q^{p,R} \in \boldsymbol{\tau}$. The resulting bound for $\delta_q^{p,R}$'s *uncertainty*, which will play a central role in our perception failure model in Subsection 2.2, is given by $\varepsilon = \tau^+ - \tau^-$.

2.2 Perception Failure Model

Consider the round R perception vector $\mathcal{V}_q^R(t)$—observed at some real-time t— of a well-behaved process q. First of all, our execution model implies that $\mathcal{V}_q^R(t)$ is monotonic in time, in the sense that $|\mathcal{V}_q^R(t+\Delta t)| \geq |\mathcal{V}_q^R(t)|$ for any $\Delta t \geq 0$, since perceptions are only added. Moreover, since the value V_q^{R+1} to be broadcast in the next round $R+1$ is computed solely from $\mathcal{V}_q^R := \mathcal{V}_q^R(\sigma_q^R)$ and q's local state at the round switching time σ_q^R, it is obvious that, ultimately, only the failures in the perceptions present at the respective round switching times count. Timing failures are no longer visible here (but will probably affect σ_q^R, recall Section 2.1), since a message that did not drop in by σ_q^R at process q just results in $V_q^{p,R} = \emptyset$. Consequently, the resulting *perception failure model* is much simpler than the physical one and therefore more suitable for analyzing an algorithm's fault-tolerance properties.

Our formalization solely rests upon the $n \times n$ matrix $\mathcal{V}^R(t)$[5] of round R perceptions observed at the same arbitrary time t—typically, some process's round switching

[2] Throughout the paper, we use the following notation: Single lower-case letters p, q, … denote "anonymous" processes; in most cases, only the processor ids are used here. Process names and round numbers are denoted by single upper-case letters P, R, S, …. Process subscripts denote the process where a quantity like $V_q^{p,R}$ is locally available, process superscripts denote the remote source of a quantity. Calligraphic variables like $\mathcal{V}_p^R$ denote sets or vectors, bold variables like $\boldsymbol{\tau}$ denote intervals.

[3] Note that $\leq$ must be replaced by $<$ in case of $\tau^+ = \infty$. This conveniently models "totally" asynchronous systems, where only assertions like "eventually, …" are possible: $\delta_q^{p,R} < \tau^+ = \infty$ allows messages that travel arbitrarily slow but are still distinguishable from lost ones. Since a finite τ^+ is the more common case, we will use $\leq$ and closed intervals in the description of our model — but bear in mind that upper bound results must be interpreted as strict in case of $\tau^+ = \infty$.

[4] Those assumptions could be somewhat relaxed. In case of time-varying delays, for example, we could allow τ^- and τ^+ to be different in different rounds; this is exploited in the Θ-model of [18], for example. The very small self-reception delay could be considered as an (early) link timing failure and hence be masked by increasing f_ℓ^{ra} and f_ℓ^{sa} by 1.

[5] We will subsequently suppress the round number R in quantities like $\mathcal{V}^R(t)$ for brevity.

time—at all processes:

$$\mathcal{V}(t) = \begin{pmatrix} \mathcal{V}_1(t) \\ \mathcal{V}_2(t) \\ \vdots \\ \mathcal{V}_n(t) \end{pmatrix} = \begin{pmatrix} V_1^1 & V_1^2 & \dots & V_1^n \\ V_2^1 & V_2^2 & \dots & V_2^n \\ \vdots & \vdots & \vdots & \vdots \\ V_n^1 & V_n^2 & \dots & V_n^n \end{pmatrix}_t \quad (2)$$

Note that $\mathcal{V}(t)$ is in fact a quite flexible basis for our failure model, since different "views" of the state of the distributed computation can be produced easily by choosing a suitable time of observation t.

We distinguish the following failure modes for single perceptions in $\mathcal{V}(t)$ in our perception failure model:

Definition 1 (Perception Failures) *Process q's perception V_q^p of process p's broadcast value V_p can be classified according to the following mutually exclusive failure mode predicates:*

- $\text{correct}(V_q^p)$: $V_q^p = V_p$,
- $\text{omission}(V_q^p)$: $V_q^p = \emptyset$,
- $\text{value}(V_q^p)$: $V_q^p \neq \emptyset$ *and* $V_q^p \neq V_p$.

Next, we have to classify sender[6] process failures. This requires the important notion of *obedient* processes: An obedient process is an alive process that faithfully executes the particular algorithm. It gets its inputs and performs its computations exactly as a non-faulty process, but it might fail in specific ways to communicate its value to the outside world. We will subsequently use this term instead of non-faulty whenever a process acts as a receiver ("obedient receiver"), since this will allow us to reason about the behavior of (benign) faulty processes as well. If $\mathcal{R}_p$ denotes some sender p's receiver set, let $\overline{\mathcal{R}}_p \subseteq \mathcal{R}_p$ denote the set of obedient processes among those.

Whereas the physical failure model differentiates timing failures according to $\delta_q^{p,R} \in \boldsymbol{\tau}$ vs. $\delta_q^{p,R} \notin \boldsymbol{\tau}$ and hence incorporates those quantities explicitly, it is solely the choice of t that is used in Definition 2 for this purpose: It only depends upon t whether a non-faulty perception V_q^p represents a perception event pe_q^p from a non-faulty or rather a timing faulty process p. Hence, neither $\delta_q^{p,R}$ nor $\boldsymbol{\tau}$ will show up in the definitions of the perception failure model below.

Definition 2 (Perception Process Failures) *Let p be a (faulty) sender process and g be some obedient receiver with $\emptyset \neq V_g^p \in \mathcal{V}(t)$, if there is any such g. In the absence of link failures, process failures of p can be classified according to the perceptions $V_q^p \in \mathcal{V}(t+\varepsilon)$ at all obedient receivers $q \in \overline{\mathcal{R}}_p \subseteq \mathcal{R}_p$ as follows:*

- *Non-faulty:* $\forall q \in \overline{\mathcal{R}}_p : \text{correct}(V_q^p)$,
- *Manifest:* $\forall q, r \in \overline{\mathcal{R}}_p : V_q^p = V_r^p \neq V_p$ *detectably,*
- *Clean crash:* $\forall q \in \overline{\mathcal{R}}_p : \text{omission}(V_q^p)$,
- *Omission:* $\forall q \in \overline{\mathcal{R}}_p : \text{correct}(V_q^p) \vee \text{omission}(V_q^p)$,
- *Symmetric:* $\forall q, r \in \overline{\mathcal{R}}_p : V_q^p = V_r^p$,
- *Arbitrary: no constraints.*

A faulty process producing at most asymmetric omission failures is called benign faulty and is assumed to be obedient.

Bear in mind that both arbitrary and symmetric faulty processes, but not benign faulty ones, may also be faulty in the time domain.

The following Definition 3 specifies the possible failures in perceptions caused by link failures.

Definition 3 (Perception Link Failures) *In the absence of sender process failures, a failure of the link from sender p to an obedient receiver q can be classified according to its effect upon q's perception $V_q^p \in \mathcal{V}(t)$ as follows:*

- *Link non-faulty:* $V_q^p = V_p$,
- *Link omission:* $V_q^p = \emptyset$,
- *Link arbitrary: no constraint.*

The failure classes up to link omission failures are called benign.

To overcome the impossibility of consensus in presence of unrestricted link failures [14, 19], it turned out that send and receive link failures should be considered independently [22, 23]. The following link-failure-related parameters are hence incorporated in the final perception failure model of Definition 4 below:

($\mathrm{A1}^s$) *Broadcast link failures*: For any single sender s, there are at most f_ℓ^s receiver processes q with a perception vector $\mathcal{V}_q$ that contains a faulty perception V_q^s from s.

($\mathrm{A1}^r$) *Receive link failures*: In any single process q's perception vector $\mathcal{V}_q$, there are at most f_ℓ^r faulty perceptions V_q^p.

Separating broadcast and receive link failures as above 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 will consider ($\mathrm{A1}^s$) and ($\mathrm{A1}^r$) as independent of each other and of process failures, for any process in the system and any

[6]Receiver process failures will be considered below, when introducing link failures.

round. Only the model parameters f_ℓ^s and f_ℓ^r cannot be independently chosen (without restricting the link failure patterns), since the system-wide number of send and receive link failures must of course match. Hence, $f_\ell^s = f_\ell^r = f_\ell$ is the most natural choice, although other settings can also be considered [22].

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.

The following Definition 4 contains our complete perception-based failure model, which just specifies the properties of any round's perception matrix $\mathcal{V}^R(t)$. Note carefully that this definition is valid for arbitrary times t, including those where the perceptions from some senders did not yet arrive (and are hence $\emptyset$).

Definition 4 (Asynchronous Perception Failure Model) *Let $\mathcal{V}^R(t)$ be the round R perception matrix of an asynchronous system of processes running on different processors that comply to our execution model. For any obedient receiver q, it is guaranteed that $V_q^{p,R} = \emptyset$ if $p \notin \mathcal{S}_q^R$ or if $V_q^{p,R}$ was not received by time t. Moreover:*

- *(P1) There are at most f_a, f_s, f_o, f_c, and f_m columns in $\mathcal{V}^R(t)$ that correspond to arbitrary, symmetric, omission, clean crash, and manifest faulty processes and may hence contain perceptions $V_q^{p,R}$ according to Definition 2.*
- *(A1^s) In every single column p, at most f_ℓ^s perceptions $V_q^{p,R} \in \mathcal{V}^R(t)$ corresponding to obedient receivers $q \in \overline{\mathcal{R}}_p^R \subseteq \mathcal{R}_p^R$ may differ from the ones obtained in the absence of broadcast link failures. At most $f_\ell^{sa} \leq f_\ell^s$ of those perceptions may be link arbitrary faulty.*
- *(A1^r) In every single row q corresponding to an obedient receiver, at most f_ℓ^r of the perceptions $V_q^{p,R} \in \mathcal{V}^R(t)$ corresponding to senders $p \in \mathcal{S}_q^R$ may differ from the ones obtained in the absence of receive link failures. At most $f_\ell^{ra} \leq f_\ell^r$ of those may be link arbitrary faulty.*
- *(A2) Process q can be sure about the origin p of $V_q^{p,R} \in \mathcal{V}^R(t)$.*
- *(A3) $\emptyset \neq V_q^{p,R} \in \mathcal{V}^R(t) \Rightarrow \emptyset \neq V_r^{p,R} \in \mathcal{V}^R(t+\varepsilon)$ for every non-faulty sender p connected to obedient receivers q and r via non-faulty links.*

Remarks:

1. The effects of process failures (P1) and link failures (A1^s), (A1^r) are considered orthogonal; it can hence happen that a link failure hits a perception originating from a faulty sender process. This is also true for manifest and clean crash failures, where link arbitrary failures could create non-empty perceptions at some receivers.
2. Our model also allows to model "totally" asynchronous systems by setting $\tau^+ = \infty$, which implies $\varepsilon = \infty$. Although the distinction between symmetric and asymmetric failures is void here, it still makes sense to distinguish arbitrary, (early) time, omission and manifest failures.

The primary way of using our failure model in the analysis of agreement-type algorithms is the following: Given the perception vector $\mathcal{V}_q^R(\sigma_q^R)$ of some specific obedient receiver process q at its round switching time σ_q^R, it allows to determine how many perceptions will at least be present in any other process r's perception vector $\mathcal{V}_r^R(\sigma_q^R + \varepsilon)$ shortly thereafter. The following Lemma 1 formalizes this fact.

Lemma 1 (Difference in Perceptions) *At any time t, the perception vector $\mathcal{V}_q(t)$ of any process at an obedient receiver q may contain at most $f_\ell^{ra} + f_a + f_s$ timing/value-faulty perceptions $V_q^p \neq \emptyset$. Moreover, at most $\Delta f = f_\ell^{ra} + f_\ell^r + f_a + f_o$ perceptions V_r^p corresponding to $V_q^p \neq \emptyset$ may be missing in any other obedient receiver's $\mathcal{V}_r(t + \Delta t)$ for any $\Delta t \geq \varepsilon$.*

Proof: The first statement of our lemma is an obvious consequence of Definition 4. To prove the second one, we note that at most $f_\ell^{ra} + f_a + f_o$ perceptions may have been available (partly too early) at q without being avaliable yet at r, additional $f_\ell^{ra\prime} \leq f_\ell^{ra}$ perceptions may be late at r, and $f_\ell^r - f_\ell^{ra\prime}$ ones could suffer from an omission at r. All symmetric faulty perceptions present in $\mathcal{V}_q(t)$ must also be present in $\mathcal{V}_r(t + \Delta t)$, however. Summing up all the differences, the expression for Δf given in Lemma 1 follows. □

3 Elementary Broadcast Primitives

In Section 4, we will provide the complete analysis of an advanced asynchronous algorithm, namely, the randomized Byzantine agreement algorithm of [25, 26], under our perception failure model. It relies upon two well-known broadcast primitives, *echo broadcast* [7, 26] and *simulated authenticated broadcast* [25], which have to be adapted and analyzed first. To keep the notation simple, we will use the "all-Byzantine" restriction of the perception failure model in Definition 4 ($f_s = f_o = f_c = f_m = 0$, which also implies obedient = non-faulty) throughout this section.

3.1 Echo Broadcasting

The *echo broadcast* primitive of [7, 26] implements crusader's agreement [9], which limits the power of faulty processes during a broadcast. Its interface consists of two functions *echo-broadcast*(V_p) and *echo-deliver*(V_p), which allow a process p to broadcast some value V_p to all processes in the system. The semantics of the echo broadcast primitive ensures that any two obedient processes that ever *echo-deliver* get the same value, and that all obedient processes will *echo-deliver* if the broadcaster is non-faulty, see Theorem 1 below.

Note that we will only consider a single instance of echo broadcasting in this section. Typical applications like the consensus algorithm of Figure 3 require multiple instances, however, which can be distinguished by their process names. We use the round number R of the application process that calls *echo-broadcast* as the process name; it is of course fixed and should not be confused with the round numbers of the echo broadcast implementation.

```
  Implementation of echo-broadcast(V_p):
send (bcast, R, V_p, p) to all;

  Process for echo-deliver(V_p):
cobegin
/* Concurrent block for R */
if received (bcast, R, V_p, p) from p
   → send (echo, R', V_p, p) to all [once]; /* rebroadcast */
fi

/* Concurrent block for R' */
if received (echo, R', V_p, p) with identical V_p = V from
n − f_ℓ^s − f_ℓ^r − f_a distinct processes
   → unblock echo-deliver(V_p); /* ready for echo-deliver at rec. */
fi
coend
```

Figure 1. *Echo broadcast primitive for the "all-Byzantine" hybrid failure model of Definition 4*

Figure 1 shows the pseudo code of our hybrid version of the original algorithm. It needs two concurrent single-round processes named R and R' on each processor, which send messages consisting of a type (*bcast* resp. *echo*), the process name (R resp. R'), the broadcast value V_p, and the originator of the broadcast p to each other (including itself). In the implementation of *echo-broadcast*, the broadcaster p just disseminates its value V_p to all peers. V_p will be received in process R at every processor and echoed to all. The echo messages are collected by process R', which eventually unblocks *echo-deliver* when sufficiently many of of those dropped in. The numbers f_ℓ^s, f_ℓ^r, and f_a denote the maximum tolerated number of failures as specified in Definition 4. Note that f_a gives the maximum number of *processors* that may execute a faulty process R, R', or application process (calling *echo-broadcast* or *echo-deliver*). Since "at most f faulty processors" implies "at most f faulty processes (of the same name)", however, we will use the phrases "faulty processors" and "faulty processes" synonymously.

The following Theorem 1 proves that the algorithm of Figure 1 satisfies the properties of echo broadcasting:

Theorem 1 (Properties Echo Broadcasting) *In a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 1$ processors satisfying the failure model of Definition 4, where $f_a \geq 0$ gives the maximum number of Byzantine faulty processors during the entire execution, the echo broadcast primitive of Figure 1 guarantees:*

- *(UC)* Uniform Correctness*: If non-faulty processor p executes* echo-broadcast(V_p) *at time t, then* echo-deliver(V_p) *at every obedient processor is unblocked within $[t + 2\tau^-, t + 2\tau^+]$.*
- *(UU)* Uniform Unforgeability*: If processor p is obedient and does not execute* echo-broadcast(V_p) *by time t, then* echo-deliver(V_p) *cannot unblock at any obedient processor by $t + 2\tau^-$ or earlier.*
- *(UA)* Uniform Agreement*: If* echo-deliver(V_q^p) *and* echo-deliver(V_r^p) *both return a value broadcast by processor p at two obedient processes q and r, respectively, then $V_q^p = V_r^p$.*

System-wide, at most $n + 1$ broadcasts of $(\log_2 n + \log_2 C_R + \log_2 C_V + 1)$-bit messages are performed by obedient processes, where C_R resp. C_V give the cardinality of the process name space resp. the set of broadcast values.

Proof: (*Uniform Correctness*.) Since the broadcaster p is non-faulty, at least $n - f_\ell^s - f_a$ non-faulty receivers get (*bcast*, R, V_p, p) in process R and emit (*echo*, R', V_p, p) within time $[t + \tau^-, t + \tau^+]$, according to (A1^s) in Definition 4 and the definition of τ^-, τ^+. Consequently, any obedient receiver gets at least $n' = n - f_\ell^s - f_\ell^r - f_a$ correct (*echo*, R', V_p, p) from different processes in process R' within another $\boldsymbol{\tau} = [\tau^-, \tau^+]$, by (A1^r) and (P1) in Definition 4. According to Figure 1, *echo-deliver*(V_p) at any obedient receiver is hence unblocked as asserted. Note that *echo-deliver*(V_p) cannot succeed before $2\tau^-$, since this could only happen if p was faulty, cp. the unforgeability proof below.

(*Uniform Unforgeability*.) The proof is by contradiction: Assume that there is an obedient process q that unblocks *echo-deliver* by $t + 2\tau^-$, which implies

$$|\mathcal{V}_q^{R'}(t+2\tau^-)| \geq n - f_\ell^s - f_\ell^r - f_a \geq f_\ell^s + 2f_\ell^{ra} + f_\ell^r + 2f_a + 1$$

according to the second **if** in Figure 1. Since only at most $f_\ell^{ra} + f_a$ of the corresponding messages $(echo, R', V_p, p)$ might originate from arbitrary receive link failures[7] and Byzantine faulty processors, and at most f_ℓ^{sa} obedient processes could have sent $(echo, R', V_p, p)$ in response to some spurious $(bcast, R, V_p, p)$ messages caused by broadcast link failures in process R, at least one obedient processor r not affected by a broadcast link failure in process R must have sent $(echo, R', V_p, p)$, by time $t + \tau^-$. This can only happen if r got a true $(bcast, R, V_p, p)$—not a spurious one caused by an arbitrary broadcast link failure—in the first **if** sent by time t. This contradicts the assumption of the unforgeability property, however.

(*Uniform Agreement.*) If two different obedient processes q and r *echo-deliver* two different values, they must have got sufficiently many same echo-messages with different values $V_q^p \neq V_r^p$ each. We use a simple pigeonhole principle argument to show that this is impossible, given that there are only n processes that could have sent such messages: Obviously, $g_q \geq n - f_\ell^s - f_\ell^r - f_a - f_\ell^{ra} - f_a$ resp. $g_r \geq n - f_\ell^s - f_\ell^r - f_a - f_\ell^{ra} - f_a$ of the messages received at q resp. r must originate from non-faulty processes. Since there are at most $g \leq n - f_a$ such processes, $X = g_q + g_r - g$ must satisfy

$$X \geq n - 2(f_\ell^s - f_\ell^r - 2f_a - f_\ell^{ra}) + f_a \geq 1.$$

Consequently, at least one non-faulty process must have sent different messages to q and r, which is impossible.

As far as the claimed message complexity of our algorithm is concerned, it is of course impossible to bound the number of message broadcasts by non-obedient processes. Every of the at most n processes that faithfully executes the algorithm of Figure 1, however, performs at most one broadcast of $(echo, R', V_p, p)$ in direct or indirect response to the initial one, where only process p broadcasts the message $(bcast, R, V_p, p)$. This completes the proof of Theorem 1. $\square$

3.2 Simulated Authenticated Broadcasting

We now turn to the more involved *simulated authenticated broadcast* primitive of [25], which implements authenticated reliable broadcasts without cryptography. Note that we are dealing with the asynchronous version here; its synchronous counterpart has been analyzed in the context of the consensus algorithm of Srikanth & Toueg in [6].

[7] Link arbitrary failures could produce time faulty messages "out of thin air".

Simulated authenticated broadcasting is implemented by means of two functions, namely, *sa-broadcast*(V_p) and *sa-deliver*(V_p), which allow a process p to reliably broadcast some value V_p to all processes in the system. The semantics of simulated authenticated broadcasting is captured by three properties, namely, correctness, unforgeability, and relay, defined in Theorem 2 below.

As in Section 3.1, we will again consider only a single instance of echo broadcasting in this section. Typical applications like the consensus algorithm of Figure 3 require multiple instances, however, which can be distinguished by their process names. We again use the round number R of the application process that calls *sa-broadcast* as the process name; it is of course fixed and should not be confused with the round numbers of the broadcast implementation.

```
  Implementation of sa-broadcast(V_p):
send (bcast, R, V_p, p) to all;

  Process for sa-deliver(V_p):
cobegin
/* Concurrent block for R */
if received (bcast, R, V_p, p) from p
   → send (echo, R', V_p, p) to all [once]; /* rebroadcast */
fi

/* Concurrent block for R' */
if received (echo, R', V_p, p) with identical V_p = V from
f_ℓ^sa + f_ℓ^ra + f_a + 1 distinct processes
   → send (echo, R', V_p, p) to all [once]; /* sufficient evidence */
fi
if received (echo, R', V_p, p) with identical V_p = V from
f_ℓ^sa + 2f_ℓ^ra + f_ℓ^r + 2f_a + 1 distinct processes
   → unblock sa-deliver(V_p); /* ready for sa-deliver at receiver */
fi
coend
```

Figure 2. *Simulated authenticated broadcast primitive for the "all-Byzantine" hybrid failure model of Definition 4*

Figure 2 shows the pseudo code of our hybrid simulated authenticated broadcast primitive. It consists of two concurrent single-round processes named R and R' on each processor, which send messages consisting of a type (*bcast* resp. *echo*), the process name (R resp. R'), the broadcast value V_p, and the originator of the broadcast p to each other (including itself). The initial message $(bcast, R, V_p, p)$ is used by the broadcaster p's process R to signal that the function *sa-broadcast*(V_p) has been called, $(echo, R', V_p, p)$ is emitted (at most once) either by process R upon reception of (*bcast*, R, V_p, p) from p, or when process R' got (*echo*, R', V_p, p) from at least one non-faulty process ("sufficient evidence"). The figures f_a, f_ℓ^r, and f_ℓ^{ra} give the maximum tolerated number of failures as specified in Definition 4. As in Section 3.1, those numbers give the maximum number of *processors* that may execute a faulty process R, R', or application process (calling *sa-broadcast* or *sa-deliver*). Since "at most f faulty processors" implies "at most f faulty processes (of the same name)", however, we will use the phrases "faulty processors" and "faulty processes" synonymously.

Theorem 2 below proves that the algorithm of Figure 2 satisfies the properties of authenticated broadcasting. It uses the properties of perception vectors at two different obedient receivers established in Lemma 1 in Section 2.2.

Theorem 2 (Properties Simulated Auth. Broadcasting) *In a system with $n \geq f_\ell^{sa} + f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 1$ processors satisfying the failure model of Definition 4, where $f_a \geq 0$ gives the maximum number of Byzantine faulty processors during the entire execution, the simulated authenticated broadcast primitive of Figure 2 guarantees:*

(UC) Uniform Correctness*: If non-faulty processor p calls* sa-broadcast(V_p) *at time t, then* sa-deliver(V_p) *at every obedient processor is unblocked within* $[t + 2\tau^-, t + 2\tau^+]$.

(UU) Uniform Unforgeability*: If processor p is obedient and does not execute* sa-broadcast(V_p) *by time t, then* sa-deliver(V_p) *cannot unblock at any obedient processor by* $t + 2\tau^-$ *or earlier.*

(UR) Uniform Relay*: If* sa-deliver(V_p) *at an obedient processor is unblocked at time t, then every obedient processor does so by time* $t + \tau_\Delta$*, where* $\tau_\Delta = \varepsilon + \tau^+$.

System-wide, at most $n + 1$ broadcasts of $(\log_2 n + \log_2 C_R + \log_2 C_V + 1)$-bit messages are performed by obedient processes, where C_R resp. C_V give the cardinality of the process name space resp. the set of broadcast values.

Proof: (*Uniform Correctness*.) Here we can assume that the broadcaster p is non-faulty. Hence, according to (A1^s) in Definition 4 and the definition of τ^-, τ^+, at least $n - f_\ell^s - f_a$ non-faulty receivers get (*bcast*, R, V_p, p) and emit (*echo*, R', V_p, p) in process R within time $[t + \tau^-, t + \tau^+]$. Consequently, any obedient receiver gets at least

$$n' = n - f_\ell^s - f_\ell^r - f_a \geq f_\ell^{sa} + 2f_\ell^{ra} + f_\ell^r + 2f_a + 1$$

correct (*echo*, R', V_p, p) from different processes in process R' within another $\boldsymbol{\tau}$, by (A1^r) and (P1) in Definition 4. According to Figure 2, *sa-deliver*(V_p) at any obedient receiver is hence unblocked as asserted. Note that *sa-deliver*(V_p) cannot erroneously succeed before $t + 2\tau^-$, according to the uniform unforgeability property (UU).

(*Uniform Unforgeability*.) The proof is by contradiction: Assume that there is an obedient process q that unblocks *sa-deliver* by $t + 2\tau^-$, which implies $|\mathcal{V}_q^{R'}(t + 2\tau^-)| \geq f_\ell^{sa} + 2f_\ell^{ra} + f_\ell^r + 2f_a + 1$ according to the third **if** in Figure 2. Since only at most $f_\ell^{ra} + f_a$ of the corresponding (*echo*, R', V_p, p) may be due to messages produced by arbitrary receive link failures[8] and timing/value-faulty processors, and at most f_ℓ^{sa} obedient processes could have sent (*echo*, R', V_p, p) in response to spurious (*bcast*, R, V_p, p) caused by broadcast link failures in process R, at least one obedient processor r not affected by a broadcast link failure in process R must have sent (*echo*, R', V_p, p), by time $t + \tau^-$.

This happens if either (1) r got a true (*bcast*, R, V_p, p)—not a spurious one caused by an arbitrary broadcast link failure—in the first **if** sent by time t, or (2) r achieved sufficient evidence in the second **if**, that is, $|\mathcal{V}_r^{R'}(t + \tau^-)| \geq f_\ell^{sa} + f_\ell^{ra} + f_a + 1$. By the same argument as before, (2) requires that at least one obedient processor q sent (*echo*, R', V_p, p) by time t, which in turn can only happen if (1) applies to q (for time $t - \tau^- \leq t$). Case (1), however, contradicts the assumption of the unforgeability property.

(*Uniform Relay*.) Since some obedient process q unblocks *sa-deliver* at time t, $|\mathcal{V}_q^{R'}(t)| \geq f_\ell^{sa} + 2f_\ell^{ra} + f_\ell^r + 2f_a + 1$ according to the delivery criterion in Figure 2. Hence, the perception vector at any obedient process r must satisfy

$$|\mathcal{V}_r^{R'}(t + \varepsilon)| \geq f_\ell^{sa} + f_\ell^{ra} + f_a + 1$$

according to Lemma 1. It follows that all non-faulty processes achieve sufficient evidence in the second **if** of Figure 2 and send (*echo*, R', V_p, p) to all processes by this time. As in the proof of correctness, this implies

$$|\mathcal{V}_r^{R'}(t + \varepsilon + \tau^+)| \geq f_\ell^{sa} + 2f_\ell^{ra} + f_\ell^r + 2f_a + 1$$

for any obedient process r, which causes r to unblock *sa-deliver* by $t + \varepsilon + \tau^+ = t + \tau_\Delta$ as asserted.

As far as the claimed message complexity of our algorithm is concerned, it is of course impossible to bound the number of message broadcasts by not obedient processes. Every of the at most n processes that faithfully executes the algorithm of Figure 2, however, perform at most one broadcast of (*echo*, R', V_p, p) in direct or indirect response to the initial one, where only process p broadcasts (*bcast*, R, V_p, p). This completes the proof of Theorem 2. $\square$

It is important to note that the simulated authenticated broadcast primitive of Figure 2—unlike echo broadcasting—does *not* provide (uniform) *uniqueness* defined as: If a non-faulty (obedient) process unblocks

[8] Link arbitrary failures could produce time faulty messages "out of thin air".

sa-deliver(V_p) for a value V_p broadcast in round R, then no non-faulty (obedient) process unblocks *sa-deliver*(V'_p) for a value $V'_p \neq V_p$ broadcast in round R. In the algorithm of Figure 2, a faulty broadcaster could hence "inject" multiple values in the same round, simply by inconsistently sending those to different echoing processes. Uniform relay (UR) guarantees, however, that every obedient process eventually gets every value. This fact will allow us to get rid of the costly "proof-concept" in the original randomized Byzantine agreement algorithm of [26].

4 Randomized Byzantine Agreement

Enabled by the results of the previous subsection, we are ready for investigating the randomized consensus algorithm of [25] under our perception failure model. A (randomized) consensus—also called Byzantine agreement—algorithm computes (with high probability) a common decision value V based upon initial values v_p provided locally at every process p. Our algorithm will be based upon Toueg's improvement [26] of the algorithm proposed by Rabin in [20], which uses authenticated broadcasts and Shamir's secret sharing scheme [24]. By plugging in the hybrid broadcast primitives from Section 3.1 and 3.2, a hybrid version of this algorithm is easily derived.

The secret sharing scheme of [24] assumes a non-faulty *dealer* D, which generates a sequence of random bits $s_1, s_2, \ldots$ and, for each bit s_k, n *pieces* s_k^i, $1 \leq i \leq n$. Those pieces are such that the knowledge of $t+1$ of those is necessary and sufficient for computing s_k. The dealer signs all pieces with its signature σ_D to prevent forgery and distributes to each process i the sequence of its pieces $\sigma_D(s_1^i), \sigma_D(s_2^i), \ldots$. Note that this happens off-line, *prior* to the execution(s) of the actual consensus algorithm, and is the only place where authentication will be required in our hybrid algorithm. This secret sharing scheme makes it impossible for t faulty processes to compute the secret solely from their pieces at runtime — at least one non-faulty process's piece must be available for this purpose as well.

The original algorithm of [26] (cp. Figure 3) computes, with probability at least $1 - (1/2)^K$, a common consensus value V from all the processes' input values $v_p \in \{0, 1\}$, $1 \leq p \leq n$, by means of K iterations with three phases (which may consist of multiple rounds each). In phase 1, all processes broadcast their current consensus value V_p (initially $V_p = v_p$) and wait for the arrival of $n - f$ authenticated messages from different processes (including itself). Every process then computes a new value for V_p based upon the values in the received messages and saves the latter as a *proof* for its choice.

In phase 2, every process *echo-broadcasts* its new value V_p along with its proof to all processes in the system and waits for $n - f$ such messages from different processes. It computes the number of messages containing $V_q = 1$ among those and saves them in the variable *count*.

In phase 3, every process p in iteration k discloses its piece of the secret s_k^p by broadcasting it to all processes, and waits for the arrival of $f + 1$ pieces—with a correct dealer's signature—from different processes. When they arrive, process p can compute the shared secret s_k. Note that it does not matter whether a piece comes from a faulty or non-faulty process since forging a piece is impossible due to authentication.

Finally, a new consensus value V_p is computed by suitably combining the random bit s_k with the value of *count* obtained in phase 2: The algorithm ensures that, with probability at least 1/2, all non-faulty processes achieve the same new consensus value at the end of an iteration, even if there was arbitrary disagreement before.

4.1 The Hybrid Algorithm

We will now develop a hybrid variant of the above algorithm for the "all-Byzantine" setting of our perception failure model. It is assumed here that there are $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 1$ processors in the system, with at most $f_a \geq 0$ arbitrary faulty ones[9] among those. Faulty processes may omit to send any message or even disseminate faulty values in a colluded attempt to fail the system. Note that the "all-Byzantine" setting is appropriate here, since crash failures are as severe as arbitrary failures for our algorithm.

The pseudo code of our hybrid randomized consensus algorithm is shown in Figure 3. Note that link failures show up explicitly only in phase 3 of the hybrid algorithm, since they are completely hidden by the broadcast primitives in phases 1 and 2.

Figure 3 reveals that we only replaced the authenticated broadcast in phase 1 of the original algorithm by the simulated broadcast primitive of Section 3.2. The latter does not employ signatures, however, so we could not retain the *proof* concept. Instead, we exploit the fact that simulated authenticated broadcasting satisfies the uniform relay property (UR) according to Theorem 2: It guarantees that, eventually, every non-faulty process must get all messages seen by any other non-faulty process (although not necessarily in the same sequence). As as consequence, our algorithm needs to *echo-broadcast* the single value V_p only, which considerably reduces the communication costs.

Receiver q can verify in phase 2 whether the value V_p disseminated by some process p via *echo-broadcast* is legitimate as follows: It just looks whether there are $n - f_a$

[9] We will again use the terms "faulty process" and "faulty processor" synonymously in this section. This implies, in particular, that the processes executing at a non-faulty processor are all non-faulty.

```
Code for process p:
V_p = v_p; /* Initial value */
for k = 0 to K − 1 do

  /* Phase 1 (round R = 3k) */
  sa-broadcast(V_p);
  wait for sa-deliver(V_q) from n − f_a processes;
  count := number of q's with V_q = 1;
  if count ≥ n − 2f_a then V_p := 1    else V_p := 0;
  fi

  /* Phase 2 (round 3k + 1) */
  echo-broadcast(V_p);
  wait for echo-deliver(V_q) with acceptable(V_q) from n − f_a distinct
  processes;
  count := number of q's with V_q = 1;

  /* Phase 3 (round 3k + 2) */
  send(σ_D(s_k^p)) to all; /* disclose piece */
  wait for receive(σ_D(s_k^q)) with correct σ_D from f_ℓ^ra + f_a + 1 procs;
  compute s_k from the received pieces;
  if (s_k = 0 and count ≥ 1) or
     (s_k = 1 and count ≥ 2f_a + 1) then V_p = 1;
     else V_p := 0;
  fi
od
```

Figure 3. *Randomized binary consensus algorithm for the hybrid failure model of Definition 4*

phase 1 messages among its—possibly larger—set of received ones, such that V_p satisfies the condition of the **if** of phase 1 if applied to those; we express this via the predicate *acceptable*(V_p). It is important to note, however, that q must wait until $n - f_a$ phase 2 messages from different p's have passed this test — after all, it may be the case that the phase 1 messages that were used by p to compute its V_p in phase 2 did not yet arrive at q.

By means of a proof that almost literally follows the original one in [26], it is not difficult to establish the following major Theorem 3:

Theorem 3 (Properties Randomized Consensus) *In a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 1$ processors according to Definition 4, where $f_a \geq 0$ denotes the maximum number of arbitrary faulty processors during the whole execution, the randomized consensus algorithm of Figure 3 satisfies:*

P1. Termination*: All non-faulty processes terminate the algorithm.*

P2. Validity*: If all non-faulty processes p start with the initial value $v_p = m$, then every non-faulty process terminates the algorithm with $V = m$.*

P3. Randomized Agreement*: With probability at least $1 - (1/2)^K$, every non-faulty process terminates the algorithm with the same value V.*

Proof: (*Termination.*) We have to show that the wait statements executed by non-faulty processes always terminate. A tedious but easy proof by induction is based on the following remark: Suppose all the non-faulty processes reach the beginning of a given iteration. Each one will *broadcast* the message required by that iteration to all the processes in phase 1. By the uniform correctness property (UC) in Theorem 2, every non-faulty process will eventually *sa-deliver* the message broadcast by a non-faulty process. Therefore, every non-faulty process *sa-delivers* a message from at least $n - f_a$ distinct processes.

The same is true for *echo-broadcast* in phase 2, where the uniform correctness property (UC) of Theorem 1 applies: Every non-faulty process p *echo-delivers* a message from at least $n - f_a$ distinct non-faulty processes. We have to verify, however, that *acceptable*(V_q) is eventually true for all of those: Since all the messages *sa-delivered* at process q in phase 1 must also arrive within τ_Δ at process p according to the relay property (R) in Lemma 2, this is obviously the case.

Last but not least, termination of the wait in phase 3 is obvious, since at least $n - f_a - f_\ell^r \geq f_\ell^{ra} + f_a + 1$ pieces from different non-faulty processes must arrive at any non-faulty process. Consequently, every non-faulty process terminates its iteration and arrives at the start of the next one.

(*Validity.*) Suppose $n - f_a$ (or more) non-faulty processes p have the same value $V_p = m$ at the beginning of iteration k. We show that all the non-faulty processes will have $V = m$ at the end of this iteration. Consequently, once an agreement on a value m is reached by the non-faulty processes, this agreement on m will hold at the end of each subsequent iteration.

We first claim that, in phase 1 of iteration k, every non-faulty process p has *count* $\geq n - 2f_a$ if and only if $m = 1$: Since at most f_a processes *broadcast* a value different from m, at least $n - 2f_a$ of the $n - f_a$ messages received by p have the value m and at most f_a have a value different from m. So, if $m = 1$ then *count* $\geq n - 2f_a$, and if $m = 0$ then *count* $\leq f_a$. Note that $f_a < n - 2f_a$, and the claim is proved.

From this claim, we conclude that every non-faulty process sets $V := m$ at the end of phase 1 of iteration k. We now show that, at the beginning of phase 2, there are no *echo-delivered* messages that satisfy *acceptable*(V_q) with $V_q \neq m$: Suppose $m = 0$, so at least $n - f_a$ non-faulty processes p have $V_p = 0$ at the beginning of iteration k. However, *acceptable*(V_q) with $V_q = 1$ would require at least $n - 2f_a > f_a$ messages with value 1 *sa-delivered* in phase 1, which is impossible. If, on the other hand, $m = 1$, then *acceptable*(V_q) with $V_q = 0$ would require $\geq f_a + 1$ messages with value 0 *sa-delivered* in phase 1, which is also impossible.

This implies that every non-faulty process considers only messages with $V_q = m$ *acceptable* in phase 2, so *count* is set to either 0 if $m = 0$ or $n - f_a \geq 2f_a + 1$ if $m = 1$. Consequently, $V_p := m$, independently from the value of the bit s_k.

(*Randomized Agreement*.) Assume that there is no agreement in the system at the beginning of iteration k, in the sense that that not all non-faulty processes p have the same value $V_p = m$. We show that, with probability at least 1/2, we have agreement at the end of iteration k.

Consider phase 2 of iteration k and let p be the *first* non-faulty process that terminates the appropriate wait by having *echo-delivered*(V_q) satisfying *acceptable*(V_q) (we call this *accepted*) from $n - f_a$ distinct processes $p_1, p_2, \ldots, p_{n-f_a}$. There are two possibilities for p's variable *count*:

(i) *count* $\geq f_a + 1$. In this case, we claim that all the non-faulty processes will have *count* ≥ 1 in phase 2. Without loss of generality, p accepted a message with value 1 from processes $\{p_1, \ldots, p_{f_a+1}\}$. In the same phase, every non-faulty process q accepts messages from all but f_a processes. Therefore, q accepts a message from at least one process $p_j \in \{p_1, \ldots, p_{f_a+1}\}$. By the uniform agreement property (UA) in Theorem 1, the messages *echo-delivered* by p and q from p_j must be identical, so q has *count* ≥ 1 as asserted.

(ii) *count* $< f_a + 1$. In this case, we claim that all non-faulty processes will have *count* $< 2f_a + 1$ in phase 2. Any non-faulty process q accepts messages with $V_p = 1$ from at most f_a processes in $\{p_1, \ldots, p_{n-f_a}\}$, and from at most f_a processes in $\{p_{n-f_a+1}, \ldots, p_n\}$. Therefore, q has *count* $< 2f_a + 1$ at the end of this phase 2 as asserted.

Clearly, event (i) vs. (ii) is established before any non-faulty process reveals its piece of the shared secret bit s_k. Since $f_a + 1$ pieces are required to compute s_k, this event is hence established before s_k is known by any process and is hence independent of s_k's value. Let α_k be the probability that (i) applies. If $s_k = 0$ and *count* $\geq f_a + 1$, then all non-faulty processes p have *count* ≥ 1, and therefore they all set $V_p = 1$ at the end of iteration k. This happens with probability $\alpha_k/2$.

If $s_k = 1$ and *count* $< f_a + 1$, then all the non-faulty processes p have *count* $< 2f_a + 1$, and therefore they all set $V_p := 0$ at the end of iteration k. This happens with probability $(1 - \alpha_k)/2$. Putting everything together, disagreement is transformed into systemwide agreement among all non-faulty processes with probability at least $\alpha_k/2 + (1 - \alpha_k)/2 = 1/2$. This eventually completes the proof of Theorem 3. □

4.2 Running Time Analysis

We now turn our attention to the analysis of the running time of the randomized Byzantine agreement algorithm of Figure 3, which will be based upon the lower and upper bounds τ^-, τ^+ on the end-to-end computational and transmission delays. Note carefully that the algorithm does not know anything about those bounds, and that we assume $\tau^+ = \infty$ in case of purely asynchronous systems. For performance analysis purposes, however, we just stipulate that any two non-faulty processes communicate to each other over a non-faulty link within $[\tau^-, \tau^+]$. Relying upon this assumption, we can compute the best case and worst case running time as an expression involving τ^+, τ^- and $\varepsilon = \tau^+ - \tau^-$.

In sharp contrast to the existing approaches for tolerating link failures in asynchronous system, which are all based upon time redundancy, our algorithm tolerates link failures by means of resource redundancy only: Eventually, other processors assist—implicitly via the consensus algorithm—any given processor in getting all the required information, despite of link failures. Consequently, τ^+ needs to cover only a single message transmission (without retransmissions) between two correct processes via a correct link here. Since excessive delays can be considered as (early) link timing failures in our model, we can usually stipulate a much smaller τ^+. By contrast, approaches that explicitly or implicitly assume perfect communication must choose τ^+ according to the maximum message delay among all transmissions. Consequently, the concurrency offered by the assisting processors leads to a considerably smaller execution time and hence better real-time capabilities.

The algorithm of Figure 3 is essentially sequential, in the sense that its iterations and phases cannot be executed concurrently. The processes required for simulated authenticated broadcasting and echo broadcasting run concurrently with the main process, however, and must all be started at boot time in order not to lose messages sent by fast processes. Note that our implementations of echo broadcasting and simulated authenticated broadcasting employ $2n$ dedicated processes per iteration each; in practice, however, their responsibilities can be taken over by two generic processes with slightly extended capabilities.

The algorithm's running time will be computed by tracking the maximum differences $\tau_\sigma^R = \max_{nf.\ p,q} |\sigma_p^R - \sigma_q^R|$ of the round switching times of non-faulty processes during the iterations. Let us first stipulate a process p that is fast enough so that *sa-deliver* and *echo-deliver* block upon invocation. Clearly, p calls its instance of simulated authenticated broadcasting for iteration k at time σ_p^{R-1} when round $R = 3k$ commences (phase 1). Next it will call iteration k's *echo-broadcast* at round switching time σ_p^R, which is determined by the $n - f_a$-th *sa-deliver*. Finally, at time σ_p^{R+1}

defined by the $n - f_a$-th *echo-deliver*, it discloses its piece and computes a new value V_p. This round is terminated upon reception of the $f_\ell^{ra} + f_a + 1$-st correct piece at the round switching time σ_p^{R+2}, which starts the next iteration $k + 1$.

For a slower process p, the above description must be modified in order to account for the fact that p might call e.g. *sa-deliver* late, i.e., at some time where it has already been unblocked by the simulated authenticated broadcasting process and hence returns immediately. In this case, σ_p^R is equal to the maximum of the actual unblocking time of *sa-deliver* and σ_p^{R-1} (recall that all pseudo code statements, except wait, are executed in zero time).

We start our treatment with Lemma 2, which bounds the maximum difference of the times when two non-faulty processes switch from phase 1 to phase 2 resp. phase 2 to phase 3 in the algorithm of Figure 3. This lemma is valid for both echo broadcasting and simulated authenticated broadcasting, since its proof depends only upon the uniform correctness property (UC) that is the same for both broadcast primitives.

Lemma 2 (Running Time Phase 1 & 2) *For $R = 3k$ resp. $R = 3k + 1$, let $\tau_\sigma^{R-1} = \max_{nf.\ p,q} |\sigma_p^{R-1} - \sigma_q^{R-1}|$ be the maximum difference of the round switching times of all non-faulty processes in the execution of the algorithm in Figure 3, where phase 1 resp. 2 of some iteration k is entered. If the first non-faulty process f enters at time $t_f = \sigma_f^{R-1}$, then*

$$t_f + 2\tau^- \le \sigma_p^R \le t_f + \tau_\sigma^{R-1} + 2\tau^+ \tag{3}$$

for any non-faulty process p. Moreover,

$$\tau_\sigma^R = \max_{nf.\ p,q} |\sigma_p^R - \sigma_q^R| \le 2\varepsilon + \tau_\sigma^{R-1}. \tag{4}$$

Proof: Let $t_p = \sigma_p^{R-1}$ be the time when (non-faulty) process p executes *broadcast*, and t_x^p be the time when the corresponding *deliver*(V_p) returns at non-faulty process x. If $\overline{t}_x^p \le t_x^p$ denotes the time when the broadcasting process unblocks *deliver*, then

$$t_x^p = \max\{\overline{t}_x^p, \sigma_x^{R-1}\} = \max\{\overline{t}_x^p, t_x\}$$

and hence $t_x^p - t_f = \max\{\overline{t}_x^p - t_p + t_p - t_f, t_x - t_f\}$. Since the correctness property in both Theorem 2 and 1 guarantees $2\tau^- \le \overline{t}_x^p - t_p \le 2\tau^+$ for any two non-faulty processors p and x, we obtain

$$2\tau^- \le t_x^p - t_f \le \tau_\sigma^{R-1} + 2\tau^+ \tag{5}$$

for any two non-faulty processors p and x. We now have to show that this extends also to faulty p's, since we cannot be sure that the $n - f_a$-th *deliver* is from a non-faulty process.

With s_x resp. s_y denoting the time when the $n - f_a$-th distinct *deliver* returns at non-faulty processes x resp. y, we argue that there must be non-faulty processes p and q_x resp. q_y such that

$$t_x^p \le s_x \le t_x^{q_x} \qquad \text{resp.} \qquad t_y^p \le s_y \le t_y^{q_y}. \tag{6}$$

The existence of q_x and q_y follows immediately from taking the $n - f_a$-th delivery, since only $n - f_a - 1$ *delivers* occur earlier than s_x and s_y, respectively; at least one of the $f_a + 1$ remaining ones must be non-faulty. Moreover, in presence of $f'_a \le f_a$ faulty processes, at least $n - f_a - f'_a$ *delivers* among the $n - f_a$ ones must originate from non-faulty processes at x and y. Since there are only $n - f'_a$ non-faulty processes in the system, at least $2(n - f_a - f'_a) - (n - f'_a) = n - 2f_a - f'_a > 1$ must be the same. This also confirms the existence of p.

Since (6) implies

$$t_x^p - t_y^{q_y} \le s_x - s_y \le t_x^{q_x} - t_y^p, \tag{7}$$

we obtain

$$\begin{aligned}
s_x - s_y &\ge \max\{\overline{t}_x^p, t_x\} - \max\{\overline{t}_y^{q_y}, t_y\} \\
&\ge \max\{\overline{t}_x^p - t_f, t_x - t_f\} - \\
&\qquad \max\{\overline{t}_y^{q_y} - t_f, t_y - t_f\} \\
&\ge \max\{\overline{t}_x^p - t_p + t_p - t_f, t_x - t_f\} - \\
&\qquad \max\{\overline{t}_y^{q_y} - t_{q_y} + t_{q_y} - t_f, t_y - t_f\} \\
&\ge 2\tau^- - \max\{2\tau^+ + \tau_\sigma^{R-1}, \tau_\sigma^{R-1}\}
\end{aligned} \tag{8}$$

and analogously, by upper bounding,

$$s_x - s_y \quad \le \quad \max\{2\tau^+ + \tau_\sigma^{R-1}, \tau_\sigma^{R-1}\} - 2\tau^-. \tag{9}$$

It only remains to show that there cannot be any further delay due to the acceptance test in phase 2 of Figure 3, i.e., that actually $\sigma_x^R = s_x$. This is evident, though, since the above time bounds ensure that the round $R - 1$ messages of all non-faulty senders are available at any receiver x, y: After all, τ_σ^{R-1} is incorporated in (5). Combining (6) with (5) hence justifies (3), and (8) combined with (9) confirms (4). This eventually completes the proof of Lemma 2. □

Next we will improve the result of Lemma 2 for the first phase in each iteration, where simulated authenticated broadcasting is employed. We need a technical lemma on the m-th largest elements of two corresponding sets of real values for this purpose. It simply says that if any two corresponding elements differ at most by some y, then this is also true for the m-th largest elements:

Lemma 3 (Uniform Bounds m-Maxima) *Let $\mathcal{S} = \{z_i\}_{1 \le i \le n}$ with $z_i = x_i + y_i$ and y be given, such that $x_i \le x_j$ for $i < j$ and $y_i \le y$ for*

$1 \leq i, j \leq n$. *If* $\mathcal{S}' = \{w_i\}_{1\leq i\leq n}$ *with* $w_i = x_i + y$ *for* $1 \leq i \leq n$*, then the respective* m*-th largest elements satisfy* $\max_m \mathcal{S} \leq \max_m \mathcal{S}'$*.*

Proof: If $\max_m \mathcal{S} = z_v$ for some $1 \leq v \leq n$, we claim that there exists some index $p \leq n - m + 1$ such that $z_v \leq z_p$, since otherwise $z_1 < z_v, z_2 < z_v, \dots, z_{n-m+1} < z_v$. However, z_v is the m-largest element in $\mathcal{S}$, which means that there are only $n-m$ z_i's that could possible satisfy $z_i < z_v$, providing the required contradiction. Since of course $w_i \leq w_j$ for $i < j$, we have $\max_m \mathcal{S} = z_v \leq z_p \leq w_p \leq w_{n-m+1} = \max_m \mathcal{S}'$. □

Using this lemma, a bound on the maximum difference of the times when two non-faulty processes switch from phase 1 to phase 2 in the algorithm of Figure 3 can be deduced from the relay property of simulated authenticated broadcasting. It will turn out, however, that this bound is only slightly better than the one already established in Lemma 2. The reason for this is the sequential nature of the algorithm of Figure 3, which does not allow slow processes to speed up if sufficiently many messages dropped in already.

Lemma 4 (Running Time Phase 1) *For* $R = 3k$*, let* $\tau_\sigma^{R-1} = \max_{\textit{nf. } p,q} |\sigma_p^{R-1} - \sigma_q^{R-1}|$ *be the maximum difference of the round switching times of all non-faulty processes in the execution of the algorithm in Figure 3, where phase 1 of some iteration* k *is entered. Then,*

$$\begin{aligned} \tau_\sigma^R &= \max_{\textit{nf. } p,q} |\sigma_p^R - \sigma_q^R| \\ &\leq \varepsilon + \tau^+ + \max\{0, \tau_\sigma^{R-1} - 2\tau^-\}. \end{aligned} \quad (10)$$

Proof: Since the relay property of simulated authenticated broadcasting in Theorem 2 guarantees that any two non-faulty processes unblock *sa-deliver* for the same set $\mathcal{M}$ of messages within τ_Δ, although perhaps in different order, it follows that $\max_{\textit{nf. } p,q} |\sigma_p^R - \sigma_q^R|$ is just the maximum difference of the $n - f_a$-largest elements in the sets of delivery times $\mathcal{T}_p = \{t_p^i : i \in \mathcal{M}\}$ and $\mathcal{T}_q = \{t_q^i : i \in \mathcal{M}\}$ over any pair of non-faulty processes p, q.

With $\overline{\sigma}_p^R$, $\overline{\sigma}_q^R$ denoting the $n - f_a$-largest elements in the corresponding sets of the times $\overline{\mathcal{T}}_p = \{\overline{t}_p^i : i \in \mathcal{M}\}$ and $\overline{\mathcal{T}}_q = \{\overline{t}_q^i : i \in \mathcal{M}\}$ where simulated authenticated broadcasting unblocks *sa-deliver*, choosing $y = \tau_\Delta$, $z_i = \overline{t}_p^i$, $\mathcal{S} = \overline{\mathcal{T}}_p$, $x_i = \overline{t}_q^i$ and $\mathcal{S}' = \overline{\mathcal{T}}_q + \tau_\Delta = \{\overline{t}_q^i + \tau_\Delta : i \in \mathcal{M}\}$ in Lemma 3 reveals $\overline{\sigma}_p^R \leq \overline{\sigma}_q^R + \tau_\Delta$.

Since $t_p^j = \max\{\overline{t}_p^j, \sigma_p^{R-1}\}$ and $x \leq y \Rightarrow \max\{x, z\} \leq \max\{y, z\}$ for any real x, y, z, it follows from the resulting identical ordering of $\overline{\mathcal{T}}_p$ and $\mathcal{T}_p$ that if $\overline{\sigma}_p^R = \overline{t}_p^j$ with index j, then also $\sigma_p^R = \max\{\overline{\sigma}_p^R, \sigma_p^{R-1}\} = t_p^j$. Hence,

$$\begin{aligned} \sigma_p^R - \sigma_q^R &= \overline{\sigma}_p^R - \overline{\sigma}_q^R + \max\{0, \sigma_p^{R-1} - \overline{\sigma}_p^R\} - \\ &\qquad \max\{0, \sigma_q^{R-1} - \overline{\sigma}_q^R\} \\ &\leq \tau_\Delta + \max\{0, \sigma_p^{R-1} - t_p^j\} \\ &\leq \tau_\Delta + \max\{0, \sigma_p^{R-1} - \sigma_j^{R-1} - (t_p^j - \sigma_j^{R-1})\} \\ &\leq \tau_\Delta + \max\{0, \tau_\sigma^{R-1} - 2\tau^-\}; \end{aligned}$$

in the last step, we used the time bound from the correctness property in Theorem 2. Repeating the same argument with p and q exchanged produces the same result, which finally confirms (10) and completes the proof of Lemma 4. □

It only remains to determine the running time of the final phase 3 of an iteration. Using the same argument as in the proof of Lemma 2, it is not difficult to establish the following Lemma 5.

Lemma 5 (Running Time Phase 3) *For* $R = 3k + 2$*, let* $\tau_\sigma^{R-1} = \max_{\textit{nf. } p,q} |\sigma_p^{R-1} - \sigma_q^{R-1}|$ *be the maximum difference of the round switching times of all non-faulty processes in the execution of the algorithm in Figure 3, where phase 3 of some iteration* k *is entered. If the first non-faulty process* f *enters at time* $t_f = \sigma_f^{R-1}$*, then*

$$t_f + \tau^- \leq \sigma_p^R \leq t_f + \tau^+ + \tau_\sigma^{R-1} \quad (11)$$

for any non-faulty process p*, and*

$$\tau_\sigma^R = \max_{\textit{nf. } p,q} |\sigma_p^R - \sigma_q^R| \leq \varepsilon + \tau_\sigma^{R-1}. \quad (12)$$

Proof: Let $t_p = \sigma_p^{R-1}$ be the time when (non-faulty) process p broadcasts its piece, and t_x^p be the time when (non-faulty) process x delivers p's piece. If $\overline{t}_x^p \leq t_x^p$ denotes the time when the piece from p actually drops in, then

$$t_x^p = \max\{\overline{t}_x^p, \sigma_x^R\} = \max\{\overline{t}_x^p, t_x\}$$

and hence $t_x^p - t_f = \max\{\overline{t}_x^p - t_p + t_p - t_f, t_x - t_f\}$. Since the delivery of a non-faulty message over a non-faulty link occurs within τ^- and τ^+, it follows that $\tau^- \leq \overline{t}_x^p - t_p \leq \tau^+$. Plugging this into the above equation confirms (11).

Moreover, if $s_x = \sigma_x^R$ denotes the time when the $f_a + f_\ell^{ra} + 1$-th piece from a distinct sender is received in process x, it is evident that there must be two non-faulty processes p_x, q_x—not hit by a link (timing) failure—such that $t_x^{p_x} \leq s_x \leq t_x^{q_x}$: Since s_x is the time of reception of the $f_\ell^{ra} + f_a + 1$-st correct piece from a distinct process here, there is at least one reception from a non-faulty process not later than s_x, and one that is not earlier than s_x. Hence,

$$\begin{aligned} s_x - s_y &\geq \max\{\overline{t}_x^{p_x}, t_x\} - \max\{\overline{t}_y^{q_y}, t_y\} \\ &\geq \max\{\overline{t}_x^{p_x} - t_f, t_x - t_f\} - \\ &\qquad \max\{\overline{t}_y^{q_y} - t_f, t_y - t_f\} \\ &\geq \max\{\overline{t}_x^{p_x} - t_{p_x} + t_{p_x} - t_f, t_x - t_f\} - \\ &\qquad \max\{\overline{t}_y^{q_y} - t_{q_y} + t_{q_y} - t_f, t_y - t_f\} \\ &\geq \tau^- - \tau^+ - \tau_\sigma^{R-1} \end{aligned}$$

and analogously, by upper bounding, $s_x - s_y \leq \tau^+ + \tau_\sigma^{R-1} - \tau^-$. This also confirms (12) and completes the proof of Lemma 5. $\square$

Now we are ready for our final Theorem 4, which provides a lower and upper bound on the running time of our algorithm.

Theorem 4 (Running Time Randomized Consensus) *Assume a system with $n \geq 2f_\ell^s + 2f_\ell^{ra} + 2f_\ell^r + 3f_a + 1$ processors according to Definition 4, where $f_a \geq 0$ denotes the maximum number of arbitrary faulty processors during the whole execution and $\varepsilon \geq \tau^-$. If the first resp. last non-faulty process starts iteration $k = 0$ of the randomized consensus algorithm of Figure 3 at time t resp. $t + \tau_\sigma^{-1}$ for some $\tau_\sigma^{-1} \geq 0$, then any non-faulty process completes its K iterations within $[t + 5K\tau^-, t + \tau_\sigma^{-1} + 5K\tau^+]$. Moreover, all non-faulty processes complete iteration K within*

$$\tau_\sigma^{3K-1} \leq K(5\varepsilon - \tau^-) + \max\{2\tau^-, \tau_\sigma^{-1}\}. \tag{13}$$

During K iterations, at most Kn broadcasts of (signed) pieces and at most $2K(n+1)$ broadcasts of $(\log_2 n + \log_2 C_R + 2)$-bit messages are performed by obedient processes, where $C_R \geq 3K$ gives the cardinality of the process name space.

Proof: Plugging in the results (10), (4), and (12) of Lemma 4, Lemma 2 and Lemma 5, respectively, into each other shows that a single iteration increases τ_σ by $4\varepsilon + \tau^+ - 2\tau^- = 5\varepsilon - \tau^-$, where we exploited $\varepsilon \geq \tau^-$ to derive $\tau^+ + \varepsilon + \max\{0, \tau_\sigma^{R-1} - 2\tau^-\} = 2\varepsilon - \tau^- + \max\{2\tau^-, \tau_\sigma^{R-1}\}$. This confirms (13) in case of K iterations. Similarly, plugging in the result (3) of Lemma 2 for both phase 1 and 2, and (11) of Lemma 5 into each other reveals that each iteration's running time is within $[5\tau^-, 5\tau^+]$, which eventually justifies (13) as well.

The claimed message complexity is an immediate consequence of the fact that the algorithm calls simulated authenticated broadcast, echo broadcast, and broadcast of its piece once per iteration. Recalling the message complexities given in Theorem 1 and 2, Theorem 4 follows. $\square$

4.3 Assumption Coverage

In [6,22,23], we conducted an analysis of the assumption coverage in systems where individual links may fail independently with a fixed probability p in any round: Assuming that model parameters related to process failures (like f_a) are chosen conservatively enough to be never violated, we computed the *probability of failure* Q_m that the allowed maximum number of link failures $f_\ell^r = f_\ell^s = f_\ell$ is exceeded at least once during m rounds.

In [6], Q_m was computed for a "generic" algorithm, which executes γ full message exchanges (= broadcasts of all processes) per round, β additional single broadcasts per round, and α additional broadcasts. Hence, $B_{tot} = \alpha + m \cdot (\beta + \gamma n) = \mathcal{O}(mn)$ broadcasts are performed system-wide during an m-round execution. Each broadcast is a full one, i.e., involves all $n - 1$ remote processes (the transmission to itself is assumed to be fault-free). The following Theorem 5 from [6] revealed that adding processors for tolerating more link failures always decreases Q_m as long as $np < 1$ sufficiently small:

Theorem 5 (Assumption Coverage [6, Thm. 7]) *For $np < 1$ sufficiently small, the probability of failure Q_m of the generic algorithm with B_{tot} broadcasts during an m-round execution satisfies*

$$Q_m \leq Q'_m + \mathcal{O}\left(\frac{(Q'_m)^2}{B_{tot}}\right) = \mathcal{O}\left(\frac{nm \cdot (np)^{f_\ell+1}}{(f_\ell+1)!}\right), \tag{14}$$

where

$$Q'_m = B_{tot}\binom{n-1}{f_\ell+1}p^{f_\ell+1} \tag{15}$$

Using this generic result, we can compute the assumption coverage for our randomized consensus algorithm. According to Theorem 4, we just have to set $B_{tot} = 3Kn+2K$ with $n \geq 6f_\ell + 3f_a + 1$ here; note that this applies to the case where all link faults may be arbitrary. Table 1 provides the corresponding values of the (approximate) probability of failure Q'_m for $K = 8$, $p = 10^{-2}$ and the minimal number of processors $n = 6f_\ell + 3f_a + 1$. They reveal that our algorithm could reasonably be used even in bandwidth-limited wireless systems, where link faults with loss probabilities up to $p = 10^{-2}$ are common. Needless to say, much smaller values are obtained for smaller p, cp. [6, 22, 23].

f_ℓ	$f_a = 1$	$f_a = 2$	$f_a = 3$	$f_a = 4$	$f_a = 5$	$f_a = 6$
2	0.2	0.4	0.6	1	1	1
3	0.03	0.06	0.1	0.3	0.3	0.5
5	0.0009	0.002	0.003	0.005	0.009	0.01
7	2.10^{-5}	0.00004	0.00008	0.0001	0.0002	0.0004
10	1.10^{-7}	2.10^{-7}	3.10^{-7}	5.10^{-7}	9.10^{-7}	1.10^{-6}

Table 1. *Value of (approximate) probability of failure Q'_m for $p = 0.01$ for the hybrid randomized Byzantine agreement algorithm for $K = 8$ with minimal number of processes.*

4.4 Stubborn Channels

The previous analysis showed that our algorithm tolerates a considerable number of link faults without additional measures, provided that sufficiently many processors are available. For moderate link failure rates, pure UDP datagram communication can hence be used instead of TCP, for example. For high link failure rates, however, it is quite likely that our link failure bounds f_ℓ^s and f_ℓ^r are too restrictive, cf. our assumption coverage analysis in Section 4.3. Resorting to some kind of reliable communication is inevitable here. It is well-known, however, that perfect communication requires unbounded memory space [11].

Stubborn channels [15] have been proposed as an alternative. A *k-stubborn channel* is a point-to-point communication link that reliably delivers the last k messages submitted to it for transmission, provided that both sender and receiver are non-faulty and the sender eventually stops submitting messages (such that "last k messages" makes sense). Obviously, a 1-stubborn channel can easily be implemented atop of datagrams by using a single buffer: The message in the buffer is periodically retransmitted until an acknowledgment is received. If a new message is submitted for transmission before the previous one has been acknowledged, it just overwrites the previous message in the buffer. A k-stubborn channel can be implemented by using k 1-stubborn channels operating on a circular buffer.

In spite of being powerful enough for solving consensus in asynchronous systems [15], a k-stubborn channel needs only bounded memory space. Our algorithm reconfirms this fact, since it is easily modified to work with 2-stubborn channels. The modification required is *forcing* the execution of the algorithm to some future iteration: A process currently in iteration k is forced to iteration $k' > k$ by interrupting the current execution, setting the iteration loop counter to k', and resuming execution in the wait of phase 1 in iteration k'. Note, however, that *sa-broadcast*($V_p^{k'}$) is not called when forcing process p.

The following Lemma 6 shows that forcing does not affect the consensus result, provided that at least one non-faulty process is ahead by two iterations:

Lemma 6 (Forcing Rounds) *Suppose some non-faulty process p_l is currently in iteration $k' \leq k-2$ at time t, when some other non-faulty process p_a is already in iteration $k \geq 2$. Then, all non-faulty processes will complete iteration $k-1$ and thus enter iteration k within the time bounds given by Theorem 4 also when p_l is forced to iteration $k-1$. The consensus value resp. the probability of reaching agreement is not affected by forcing process p_l.*

Proof: Consider the algorithm in Figure 3. Since p_a is non-faulty and in iteration k, it must have completed iteration $k-1$ by t. Since process p_l did not enter iteration $k-1$ by t, p_a did not see a non-faulty value $V_{p_l}^{k-1}$ from p_l by the time when it terminates any phase of iteration $k-1$. Since p_a terminated iteration $k-1$, however, it must have got $n - f_a$ *sa-delivers* from distinct $P_{p_a} = \{p_1, \ldots, p_{n-f_a}\}$ in phase 1 that satisfy $p_l \notin P_{p_a}$.

Nevertheless, even when p_l is forced to phase 1 of iteration $k-1$, all non-faulty processes—including p_l—eventually *sa-deliver* the messages from all processes in P_{p_a}, by the relay property of Theorem 2. It follows that all non-faulty processes can complete phase 1 and hence, eventually, iteration $k-1$ as asserted, according to the proof of termination in Theorem 3. Since the execution time bounds of Lemma 2 and 4 are based upon the maximum acceptance delay τ_Δ, termination occurs in accordance with the results of Theorem 4.

As far as agreement is concerned, we have to distinguish 2 situations: (a) all non-faulty processes, including p_l, (would) have agreed on the same value m in iteration $k-2$, and (b) there is no agreement among those. In case (a), the validity proof in Theorem 3 ensures that p_a and hence all other non-faulty processes decide upon m in iteration $k-1$. For case (b), we note that since p_a did not see a non-faulty $V_{p_l}^{k-1}$ from p_l by the time when it terminates phase 1 in iteration $k-1$, the same must be true for the first non-faulty process p that terminates phase 1 of iteration $k-1$. This implies $p_l \notin P_p$, such that p's variable *count* cannot depend upon V_{p_l}. Since the event (i) vs. (ii) defined in the proof of randomized agreement in Theorem 3 is solely determined by *count*, probability α must be independent of V_{p_l}. As the shared secret bit is obviously independent of all values, the probability of disagreement is the same as for the non-forced operation. This completes the proof of Lemma 6. □

The result of Lemma 6 implies that process p_l can be forced to iteration $k-1$ at time t, without changing the outcome of iteration $k-1$, when sufficient evidence for the existence of a non-faulty process in iteration k is obtained. This is the case when iteration k messages from $f_a + f_\ell^{ra} + 1$ distinct processes arrive, since only at most f_ℓ^{ra} of those could be spurious messages from arbitrary receive link faults, and f_a messages could originate from faulty processes.

Most importantly, if round forcing is employed, there is no need for a process in iteration k to support iteration k' for any $k' \leq k-2$. In particular, all echo broadcasting and simulated authenticated broadcasting processes belonging to iteration k' can be killed upon switching to round k. After all, it is guaranteed that all non-faulty processes will eventually complete iteration $k-1$ and enter iteration k, which means that all late processes—which might not terminate since some forced processes stopped their support of earlier rounds—must eventually get sufficient evidence and be forced to iteration $k-1$.

Since a process in iteration k needs to deal with messages belonging to iteration k or $k-1$ only, it is not difficult to show that it suffices to reliably transmit (and receive) only the two highest-round messages of a given type if forcing is employed, i.e., that 2-stubborn channels will be sufficient. This will be done in Theorem 6 below, where we assume that each processor executes $2n$ "generic" processes [*sa-R*] resp. [*sa-R'*], which implement all the iteration's (dedicated) processes R resp. R' of simulated authenticated broadcasting for the n peer processors; recall that one dedicated process per iteration and processor was assumed in Section 4.1. Similarly, we need $2n$ generic processes [*echo-R*] resp. [*echo-R'*] on each processor for echo broadcasting. Finally, one process per processor executes the randomized Byzantine agreement algorithm. Note that all those processes are always in the same round, and are all forced together.

Theorem 6 (Stubborn Channels) *Theorems 3 and 4 remain valid if every pair of instances of the algorithms of Figure 3, 2 and 1 employs a dedicated 2-stubborn channel for basic communication, provided that round forcing is applied when (1)* $f_a + f_\ell^{ra} + 1$ sa-broadcast *or* echo-broadcast *init-messages* $(\text{bcast}, R, V_p, p)$ *arrived from distinct processors, or (2)* $(f_a + f_\ell^{ra} + 1) \times (f_a + f_\ell^{ra} + 1)$ *distinct echo-messages* $(\text{echo}, R', V_p, p)$ *arrived, as witnesses of* $f_a + f_\ell^{ra} + 1$ *distinct* $(\text{bcast}, R, V_p, p)$.

Proof: (Sketch.) Stubborn channels imply that the reliable delivery of current messages is given up when a new message is submitted for transmission. Consequently, we must show that if a message belonging to some iteration k is handed over to *send*, it cannot be harmful to meaningful earlier iterations. This is guaranteed if either (a) the receiver process will eventually be forced to at least iteration $k - 1$, or (b) the content of the previous messages is void anyway.

For the phase 3-messages sent by the randomized Byzantine agreement algorithm of Figure 3 itself, (a) is implied by Lemma 6; (b) applies if the sender process is faulty. Recalling the operation of echo broadcasting and simulated authenticated broadcasting, we have exactly two situations where messages $(\mathit{echo}, R', V_p, p)$ are generated and submitted to the stubborn channels.

(1) In the algorithm of Figure 2, when $f_a + f_\ell^{ra} + 1$ $(\mathit{echo}, R', V_p, p)$ arrived from distinct processes, then $(\mathit{echo}, R', V_p, p)$ is submitted for transmission to the stubborn channel. However, this implies sufficient evidence for Lemma 6, so case (a) above applies.

(2) In the algorithms of Figure 1 and 2, when $(\mathit{bcast}, R, V_p, p)$ arrives, then $(\mathit{echo}, R', V_p, p)$ is submitted for transmission to the stubborn channel. Here we must distinguish 2 cases: If the sender of $(\mathit{bcast}, R, V_p, p)$ is non-faulty, Lemma 6 applies and overwriting the previous message is allowed, since round forcing will eventually happen. If the sender of $(\mathit{bcast}, R, V_p, p)$ is faulty, on the other hand, its value(s) and hence any echoing is void anyway. Therefore, case (b) applies here.

This confirms that 2-stubborn channels are sufficient for our algorithm and completes the proof of Theorem 6. □

Note that (the proof of) Theorem 6 also implies that any process q needs to store only the two perceptions $V_q^{p,R}$ with the highest round numbers received from any p, i.e., one does not need unbounded memory space for perception vectors.

5 Conclusions

We presented a brief overview of a novel hybrid failure model for round-based distributed algorithms in partially synchronous systems with possibly unknown delays. It accommodates both process and link failures and distinguishes asymmetric, symmetric, omission, clean crash, and manifest failures, both in the time and in the value domain. Our model considerably simplifies accurate fault-tolerance analysis and allows the evaluation of assumption coverage and running times as well. It is hence well-suited for both synchronous and asynchronous wireline and, in particular, wireless networked systems.

We analyzed a hybrid version of the randomized Byzantine agreement of Srikanth & Toueg, which is based upon suitably adopted variants of the asynchronous echo broadcast and simulated authenticated broadcast primitives. Its probability of disagreement was found to be only 2^{-K}, which is the same as for asynchronous systems without link failures. With respect to link failure tolerance, it turned out that resource redundancy (more processes) can be used instead of time redundancy (retransmissions) for this purpose: Tolerating f_ℓ^r resp. f_ℓ^s receive resp. send link failures at every node, in every round, with $f_\ell^{ra} \leq f_\ell^r$ arbitrary ones among the f_ℓ^r receive link failures, just needs $2f_\ell^s + 2f_\ell^r + 2f_\ell^{ra}$ additional nodes. Although a comparison with the lower bound $f_\ell^s + f_\ell^{sa} + f_\ell^r + f_\ell^{ra}$ from [22] reveals that this is sub-optimal (whereas the resilience with respect to process failures is optimal [16]), our algorithm can nevertheless cope with up to $\mathcal{O}(Kn^2)$ link failures system-wide during K iterations.

A detailed analysis of the running time of the algorithm under our system model revealed a considerably performance improvement over the FLP model. Since excessive delays can be interpreted as link failures here, we can stipulate a much smaller upper bound τ^+ on the delay between non-faulty processors when computing worst case execution times. Moreover, our analysis of the assumption coverage in systems with transient link failures revealed that unreliable datagram communication is sufficient here, even for typical wireless link failure rates up to 10^{-2}. For excessive link failure rates, 2-stubborn links, which avoid unbounded memory space required for implementing perfect communications, can also be used.

References

[1] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D.-W. Wang, and L. Zuck. Reliable communication over unreliable channels. *Journal of the ACM (JACM)*, 41(6):1267–1297, 1994.

[2] M. K. Aguilera, W. Chen, and S. Toueg. On quiescent reliable communication. *SIAM Journal of Computing*, 29(6):2040–2073, April 2000.

[3] M. Azadmanesh and R. M. Kieckhafer. Exploiting omissive faults in synchronous approximate agreement. *IEEE Transactions on Computers*, 49(10):1031–1042, Oct. 2000.

[4] K. Bartlett, R.A.Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM (JACM)*, 12(5):260–261, 1969.

[5] A. Basu, B. Charron-Bost, and S. Toueg. Crash failures vs. crash + link failures. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, page 246. ACM Press, 1996.

[6] M. Biely and U. Schmid. Message-efficient consensus in presence of hybrid node and link faults. Technical Report 183/1-116, Department of Automation, Technische Universität Wien, August 2001. (submitted).

[7] G. Bracha and S. Toueg. Resilient consensus protocols. In *Proceedings of the 2nd Symposium on the Principles of Distributed Computing (PODC'83)*, pages 12–26, Montreal, Canada, 1983.

[8] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, 1999.

[9] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[10] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, Apr. 1988.

[11] A. Fekete, N. Lynch, Y. Mansour, and J. Spinelli. The impossibility of implementing reliable communication in the face of crashes. *Journal of the ACM (JACM)*, 40(5):1087–1107, 1993.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, Apr. 1985.

[13] E. Gafni. Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 143–152. ACM Press, 1998.

[14] J. N. Gray. Notes on data base operating systems. In G. S. R. Bayer, R.M. Graham, editor, *Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, chapter 3.F, page 465. Springer, New York, 1978.

[15] R. Guerraoui, R. Oliveira, and A. Schiper. Stubborn communication channels. Technical Report 98-278, Dèpartment d'Informatique, Ècole Polytechnique Fédérale de Lausanne, 1998.

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

[17] G. Le Lann. Certifiable critical complex computing systems. In K. Duncan and K. Krueger, editors, *Proceedings 13th IFIP World Computer Congress 94*, volume 3, pages 287–294. Elsevier Science B.V. (North-Holland), 1994.

[18] G. Le Lann and U. Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003. (submitted).

[19] N. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[20] M. O. Rabin. Randomized Byzantine Generals. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 403–409, 1983.

[21] U. Schmid. How to model link failures: A perception-based fault model. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, pages 57–66, Göteborg, Sweden, July 1–4, 2001.

[22] U. Schmid and B. Weiss. Synchronous Byzantine agreement under hybrid process and link failures. Technical Report 183/1-124, Department of Automation, Technische Universität Wien, Nov. 2002. (submitted; replaces TR 183/1-110).

[23] U. Schmid, B. Weiss, and J. Rushby. Formally verified byzantine agreement in presence of link faults. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 608–616, July 2-5, 2002.

[24] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[25] T. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.

[26] S. Toueg. Randomized byzantine agreements. In *Proceedings of the 3rd Symposium on Principles of Distributed Computing (PODC'84)*, pages 163–178, 1984.

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