

This paper appears in *Proc. Intl. Conference on Dependable Systems and Networks (DSN-2003)*, San Francisco, CA, USA. IEEE, 2003.

Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries

Michael Backes Christian Cachin

IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{mbc,cca}@zurich.ibm.com

April 1, 2003

Abstract

This paper presents a formal model for asynchronous distributed systems with parties that exhibit Byzantine faults or that crash and subsequently recover. Motivated by practical considerations, it represents an intermediate step between crash-recovery models for distributed computing and proactive security methods for tolerating arbitrary faults. The model is computational and based on complexity-theoretic techniques from modern cryptography, which allows for reasoning about cryptographic protocols in a formal way. One of the most important problems in fault-tolerant distributed computing, reliable broadcast, is then investigated in this hybrid model. A definition of reliable broadcast is presented and an implementation is given based on the protocol of Bracha (PODC '84).

1 Introduction

Fault-tolerant protocols are important for networked distributed systems that must cope with unreliable components. A fundamental primitive for synchronization among a group of parties is *reliable broadcast*, where a distinguished party broadcasts a value m to the other parties; if the sender is correct, then all non-faulty parties should accept m , and if the sender is faulty, all non-faulty parties should decide for the same value or not terminate the protocol at all. In asynchronous networks, which are the focus of this paper, no assumptions are made about network delays and the relative execution speed of the parties.

There is a considerable literature devoted to the specification and implementation of reliable broadcast in environments where parties may fail by crashing silently [14, 19, 13] or by executing arbitrary operations [3, 4, 24, 22]. Faults of the latter type are usually called *Byzantine* [23] and strictly generalize crash failures. In all those works, faults are assumed to be permanent in the sense that faulty parties never recover.

Recently, a more general and more realistic approach has been suggested by a number of authors, where crashed parties may subsequently *recover* and crash again, perhaps repeating this cycle arbitrarily often [16, 1, 20, 2]; several protocols for reliable broadcast and consensus have been formulated in these models. This approach allows for an appropriate treatment of real-world systems that operate over an extended period of time.

Concurrently, interest in practical asynchronous systems that tolerate Byzantine faults has been rising as well [11, 8], prompted by the growing number of malicious attacks on the Internet. However, recovery in a model with Byzantine faults is much harder because cryptographic keys may have been exposed, resulting in a loss of confidentiality. This problem has been addressed by so-called *proactive security* techniques [9, 10, 12], which periodically refresh the cryptographic keys in the system. But these methods are rather expensive and, with the exception of [6, 25], restricted to synchronous systems.

In this paper, we address an intermediate case that is relevant in practice. We introduce an asynchronous model where some parties may crash and recover again, without otherwise deviating from their specified behavior, and where other parties may exhibit permanent Byzantine faults. Our approach leads to practical protocols that recover gracefully from transient system outages and also tolerate some malicious attacks, without incurring the cost of full proactive recovery operations.

The model is based on the computational model of Cachin et al. [7, 5], which presents a complexity-theoretic formal approach to distributed protocols in the tradition of modern cryptography [18]. This model places polynomial-time computational restrictions on all system components and does not allow for infinite runs as is otherwise customary in models of distributed systems; this is necessary for treating cryptographic protocols with computational assumptions appropriately. Termination of protocols is guaranteed by restricting the amount of “work” that they generate *independently* of the network scheduler (or the adversary, more generally).

The crash-recovery models cited above generally distinguish between parties that are always up, parties that are always down, and others. Most models, however, cannot rely on parties that crash and recover indefinitely, which are called *unstable*. In contrast, our approach allows *all* non-Byzantine parties to *crash and recover repeatedly*, as long as no more than a fraction of them is crashed at any particular point in time and the total number of crashes is polynomially bounded.

We remark that hybrid models treating Byzantine faults and crashes separately have been investigated before, starting with [17]; a recent example of a hybrid protocol tolerating Byzantine faults is [15]. However, the combination of (permanent) Byzantine faults and crashes with subsequent recovery has not been addressed so far.

Outline. The paper consists of two main parts. In the first part (Section 2), the hybrid model for Byzantine faults and crashes with recovery is introduced. As mentioned before, the hybrid model extends [7], so the main part of this section merely recalls their approach and presents the differences for including crashes and recoveries. In this work, only deterministic protocols are considered, but extension to the notion of probabilistic termination [7] can be achieved analogously. The second part (Section 3) focuses on the application of the hybrid model to reliable broadcast. A formal definition of reliable broadcast in the hybrid model is given and a protocol that implements it is presented. The protocol is a generalization of Bracha’s reliable broadcast protocol for Byzantine faults [3]. This protocol may serve as a starting point for adapting more complex protocols to the hybrid model, such as Byzantine agreement and atomic broadcast. Implementation aspects are covered in Section 4, and Section 5 concludes the paper.

2 Model

2.1 Formal System Model

We assume a collection of n parties (or servers) P_1, \dots, P_n , of which the adversary can corrupt up to t , and a trusted dealer. We adopt the static corruption model, wherein the adversary has to decide for the set of corrupted parties before the actual beginning of the attack. W.l.o.g. we assume that every adversary corrupts precisely t parties, otherwise we can construct an adversary which extends a corruption of $t' < t$ parties to t parties leaving the behavior of the remaining $t - t'$ parties unchanged. Corrupted parties are not regarded as system components and are absorbed into the adversary. Uncorrupted parties are called *honest*.

We assume that the adversary may *crash* parties such that no more than f are crashed at any point in time. A crashed party will stop reading its inputs, i.e., discard all incoming messages. A rigorous definition is given later on. The adversary can also allow a party to *recover* so that it may participate in the protocol again. However, since the recovering party does not know which messages it has missed during the time it was crashed, it will generally perform some action determined by the protocol after recovering, such as asking the other parties to re-send certain messages. We call an honest party *finally-up* if it is not crashed at the end of the protocol run, i.e., when the adversary halts; otherwise, we call it *finally-down*. This yields a partition on the set of honest parties. Trivially, at most f parties are finally-down, and at least $n - t - f$ are finally-up. In previous crash-recovery models, there is usually a third possibility called *unstable*, which denotes parties that crash and recover arbitrarily often; no statements about their behavior can be made. In our model, honest parties may crash and recover arbitrarily often within a general polynomial bound, and unstable behavior is not an issue.

In order to reason about complexity-theoretic aspects of protocols, we assume that every party P_i and the adversary are implemented as probabilistic interactive Turing machines. Complexity is measured in the size of the initial content of the input tape: a common security parameter k given in unary representation (or more generally, an arbitrary k -bit string, which might include information about the initial states, see below). In particular, polynomial time is defined in this sense. Now, the set of *negligible* functions contains those functions that decrease asymptotically faster than the inverse of any polynomial, measured in the security parameter k . Formally, a function $\epsilon(k)$ is called negligible if for all $c > 0$ there exists a k_0 such that $\epsilon(k) < \frac{1}{k^c}$ for all $k > k_0$.

There is also an initialization algorithm, which is run by an external trusted party called the *dealer*; on input k , n , t , and f , it generates the state information that is used to initialize each party. W.l.o.g. we assume that this initial state information can be encoded in the already mentioned k -bit string. We leave it to the adversary to choose n , t , and f , but a specific protocol might impose its own restrictions (e.g., $n > 3t + 2f$). The adversary receives the initial state of the corrupted parties as produced by the dealer.

After initialization, the adversary may repeatedly activate a party P_i with some input message. P_i will carry out some computation, update its state, possibly generate some output messages, and wait for the next activation. The output messages are given to the adversary, and we assume that every output message includes both its origin and its destination.

In principle, the network is insecure and the adversary may choose to deliver or to drop any message it wants. Liveness conditions in this model, however, are conditional on the adversary delivering all messages among honest parties. For simplicity, we assume that the adversary cannot modify messages or forge a message's origin. This can be implemented easily in the

given model by using a message authentication code with one symmetric key distributed by the dealer for every pair of servers. Similarly, the adversary may not duplicate messages; this is reasonable since it can easily be achieved by using message counters.

The interaction of the adversary with the honest parties defines a sequence of events, which we view as logical time. This justifies notations like “at one particular point in time”, “after P_i has crashed” and so on.

In our model, protocols are invoked by the adversary. For compatibility with [7], a protocol *instance* is identified by a unique string ID , also called the *tag*, which is chosen by the adversary when it invokes the instance.

After introducing the behavior of the adversary and the actual protocol execution, we now focus on the computation of the system components, i.e., on the honest parties. Recall that each party is always activated with an input message; this message is added to an internal input buffer upon activation. We distinguish two types of messages that protocols can process and generate: The first type contains *input actions*, which represent a local activation and carry input to a protocol, and *output actions*, which signal termination and potentially carry output of a protocol; such messages are called *local events*. The second message type is an ordinary point-to-point network message, which is to be delivered to the peer protocol instance running on another party; such messages are also called *protocol messages*.

All messages are denoted by a tuple (ID, \dots) ; the tag ID denotes the protocol instance to which this message is *associated*. Input actions are of the form $(ID, \mathbf{in}, type, \dots)$, and output actions are of the form $(ID, \mathbf{out}, type, \dots)$, with *type* defined by the protocol specification. All other messages of the form $(ID, type, \dots)$ are protocol messages, where *type* is defined by the protocol implementation.

2.2 Crashes and Recoveries

In order to include crashes in our model, we assume that every party maintains a binary flag *crashed* initialized to 0. Moreover, we consider a special input action **crash**, represented by

$$(ID, \mathbf{in}, \mathbf{crash}),$$

which causes the receiving party to *crash*. A crashed party sets *crashed* to 1 and discards incoming inputs as long as *crashed* = 1.

A crashed party can be *recovered* by an input action **recover**, represented by

$$(ID, \mathbf{in}, \mathbf{recover}).$$

Upon receiving this input, the party first checks if *crashed* = 0, doing nothing in this case. Otherwise, it sets *crashed* to 0 and continues with its recovery transition as specified by the protocol implementation.

Note that honest parties crash in the form of an atomic transition and that all transitions of honest parties are therefore atomic. Hence, honest parties recover from a well-defined state.

This convention is merely for syntactic convenience, since our protocol works without change if we adopt the following, perhaps more realistic model. Recall that parties are modeled as interactive Turing machines. Every input message of a party may be augmented by a special “crash” field that specifies where the party crashes during the activation. A crash may occur immediately *after* every operation in which the party updates its state (that is, after it writes some variable(s) to its work tape) or writes a message to its outgoing communication interface (i.e., its communication tape). At this point, *crashed* is set to 1 and control returns to the

adversary. In this way, it is possible to model a crash that occurs somewhere in a compound operation like sending a message to all other parties. The atomicity of the state updates and the messages transmissions can be guaranteed using standard techniques.

Furthermore, we may partition the state of every party into *persistent* memory, which remains unchanged at a crash and can be reused after recovery, and into *volatile* memory, which is changed in an arbitrary way or lost entirely. In practice, volatile memory is usually re-initialized after a crash, so we may assume it to be empty after a crash.

Formally, we assume that any party has an additional tape, called the *persistent* tape, apart from its usual work tape. If a party crashes, the content of its work tape is erased, i.e., the work tape is replaced with an empty tape and the persistent tape remains unchanged.

2.3 Termination

Since our model considers only computationally bounded components, the standard notion of “eventual” termination cannot be used; a system run as a whole must be polynomially bounded, so definitions based on infinite runs do not apply. We measure the efficiency of a protocol by quantifying the amount of work that an honest party performs on behalf of the protocol. In combination with a liveness property such as “validity”, restricting the amount of work implies termination in the conventional sense.

We measure the amount of work of the honest parties in terms of the *message complexity* of a protocol, defined as the number of associated messages generated by honest parties. It is a random variable that depends on the adversary and on k . Similarly, the *communication complexity* of a protocol is defined as the bit length of all associated messages (generated by honest parties). It is also a random variable that depends on the adversary and on k .

Recall that the adversary’s running time is polynomially bounded in k . For simplicity, we assume $n \leq k$ and let the length of each message in the protocol be bounded by a fixed polynomial Q in the security parameter k , thus, larger messages are ignored or cut off after $Q(k)$ bits.

Formally, the work performed by honest parties is measured by a *protocol statistic* X , which is a family of real-valued, non-negative random variables $\{X_A(k)\}$, parameterized by adversary A and security parameter k , where each $X_A(k)$ is a random variable induced by running the system with A . Message and communication complexity are examples of such statistics. We restrict ourselves to those protocol statistics that are bounded by a polynomial in the adversary’s running time, which we call *bounded protocol statistics*.

The key idea to defining efficient termination is to *uniformly* bound the statistic, i.e., to bound it by a fixed polynomial, which only depends on the actual implementation of the considered protocol — independent of the adversary.

We say that a protocol statistic X is *uniformly bounded* if there exists a fixed polynomial T in k such that for all adversaries A , there is a negligible function ϵ_A , such that for all $k \geq 0$,

$$\Pr[X_A(k) > T(k)] \leq \epsilon_A(k).$$

This is the notion used to define termination in the model with purely Byzantine corruptions and no recoveries [7].

However, this notion is not sufficient for defining termination in our hybrid model. Imagine an adversary that crashes certain honest parties again and again. Obviously, in order to ensure functional correctness of the considered protocol, these parties have to execute their recovery procedure again and again, and w.l.o.g. send some message, which increases the message com-

plexity in dependence of the adversary. Hence, the message complexity protocol statistic is no longer uniformly bounded.

In order to avoid this problem, we assume that every run of the system in the hybrid model is parameterized by a function $d: \mathbb{N} \rightarrow \mathbb{N}$ such that $d(k)$ denotes the maximum number of crashes that an adversary is allowed to perform in the considered protocol.

Based on this function, we now introduce our notion of a d -uniformly bounded statistic.

Definition 1 (d -uniformly bounded statistics). Let X be a bounded protocol statistic. We say that X is d -uniformly bounded (by T_1 and T_2) for a function $d: \mathbb{N} \rightarrow \mathbb{N}$ if there exist two fixed polynomials T_1 and T_2 such that for all adversaries A , there exists a negligible function $\epsilon_A(k)$ such that for all $k \geq 0$,

$$\Pr[X_A(k) > d(k) \cdot T_1(k) + T_2(k)] \leq \epsilon_A(k).$$

We will later apply this definition to the communication complexity of a protocol. Roughly, the definition states that the complexity of the protocol is uniformly bounded if no crash occurs (which is ensured by T_2), and the computational overhead caused by one crash is also uniformly bounded (ensured by T_1). Now a typical efficiency condition might require that the communication complexity of a protocol is d -uniformly bounded for some d .

We mention that our approach can be extended to the definition of probabilistically uniformly bounded protocol statistics, which is the way to ensure termination for *probabilistic* protocols such as randomized Byzantine agreement and atomic broadcast [7]. These definitions may be modified analogously for our hybrid model.

3 Reliable Broadcast

In this section, we focus on reliable broadcast in our hybrid model. Reliable broadcast provides a way for a party to send a message to all other parties. When used multiple times, it requires that all honest parties deliver the same set of messages and that this set includes all messages broadcast by honest parties, without guaranteeing anything about the order in which messages are delivered. In the context of arbitrary faults, reliable broadcast is also known as the *Byzantine generals problem* [21].

3.1 Review of Reliable Broadcast in the Standard Model

In the standard model with Byzantine faults and no recoveries, reliable broadcast (with a tag ID) ensures the following four properties for all polynomial-time adversaries, except with negligible probability.¹

Validity: If an honest sender broadcasts a message m then all honest parties will deliver this message, provided the adversary delivers all associated messages.

Consistency: If an honest party delivers a message m and another honest party delivers a message m' , then $m = m'$.

Totality: If some honest party delivers a message, then all honest parties deliver a message, provided the adversary delivers all associated messages.

¹Note that the standard *agreement* property is split into consistency and totality [7].

Authenticity: Every honest party delivers at most one message and, if the sender is honest, this message has been broadcast by this sender before.

Additionally, it must satisfy the following efficiency condition.

Efficiency: The communication complexity of the protocol instance is uniformly bounded.

The provision that the “adversary delivers all associated messages” is the quantitative counterpart to the traditional “eventual” delivery assumption. It can be ensured for an arbitrary adversary as follows. Suppose the adversary halts and there are yet undelivered protocol messages among honest parties (these can be inferred from a transcript of the adversary’s interactions). Then using a “benign” scheduler delivering all the undelivered messages and the newly generated ones, the protocol is run until no more undelivered protocol messages exist, whereby termination in polynomial time is guaranteed by efficiency and validity.

It is easy to see that the above definition of reliable broadcast is not adequate for the hybrid model. Obviously, we cannot make any statements about *all* honest parties in general (such as in *validity*) since they might crash and never recover, but only about those which are finally-up.

However, transposing these properties to the hybrid model can be done in various ways. One possibility is to restrict all properties to those honest parties that never crash; another would be to concentrate on those honest parties that are finally-up.

We now give our definition of reliable broadcast in a hybrid model, which we believe to be the most general definition possible. This claim is explained in the remarks following the definition.

3.2 Definition of Reliable Broadcast in the Hybrid Model

Broadcast instances are parameterized by a tag ID , and since the sender is a distinguished party and known to all others, we augment the tag w.l.o.g. by the identity j of the sender. Then, we restrict the adversary to submit a request for reliable broadcast tagged with $ID.j$ to P_i only if $i = j$.

A reliable broadcast protocol is activated when the adversary delivers an input action to P_j of the form

$$(ID.j, \text{in}, \text{r-broadcast}, m),$$

with $m \in \{0, 1\}^{Q(k)}$. When this occurs, we say P_j *reliably broadcasts m tagged with $ID.j$* , or simply P_j *r-broadcasts m* . Note that only P_j is activated like this, but the other parties must be ready to participate in the protocol as well; we assume that they are activated once with a special initialization input action. A party terminates a reliable broadcast with tag $ID.j$ by generating an output action of the form

$$(ID.j, \text{out}, \text{r-deliver}, m).$$

In this case, we say P_i *reliably delivers m tagged with $ID.j$* (or *r-delivers m* for brevity).

Recall that all protocol messages generated by honest parties with prefix $ID.j$ are *associated* to an instance with tag $ID.j$; this defines also the messages contributing to the communication complexity of the instance.

Definition 2 (Reliable broadcast in the hybrid model). Given a function $d: \mathbb{N} \rightarrow \mathbb{N}$, a protocol for *reliable broadcast* in the hybrid model satisfies the following conditions except with negligible probability:

Validity: If a finally-up party r -broadcasts m tagged with $ID.j$, then all finally-up parties r -deliver m tagged with $ID.j$, provided all honest parties have been activated on $ID.j$ and the adversary delivers all associated messages.

Consistency: If some honest party r -delivers m tagged with $ID.j$ and another honest party r -delivers m' tagged with $ID.j$, then $m = m'$.

Totality: If some honest party r -delivers a message tagged with $ID.j$, then all finally-up parties r -deliver some message tagged with $ID.j$, provided all honest parties have been activated on $ID.j$ and the adversary delivers all associated messages.

Authenticity: For all ID and all senders j , every honest party r -delivers at most one message m tagged with $ID.j$. Moreover, if P_j is honest, then m was previously r -broadcast by P_j .

Efficiency: For all ID and senders j , the communication complexity of instance $ID.j$ is d -uniformly bounded.

Remarks.

1. Validity ensures liveness of a protocol, and rules out trivial protocols that do not generate any messages. In contrast to the definition in the model without crashes, we restrict validity to senders and delivering parties that are finally-up. This restriction is essential for achieving validity. Obviously, we cannot guarantee that every honest party r -delivers a message since some of these parties might be crashed at the very start of the protocol run and never be recovered.

Similarly, if we consider a finally-down sender, then the adversary might permanently crash it right after its **r-send** transition, i.e., after it has r -broadcast the message. Recall that the whole transition is atomic, i.e., the sender is able to send **r-send** messages to all parties. Now, the adversary crashes the first honest party and delivers this **r-send** message (which will be discarded). After that, it recovers this party and applies the same procedure iteratively to all honest parties. Note that this behavior is valid, since at most one party (besides the sender) is crashed at every particular point in time, but no party will ever record the value of the **r-send** message, and the sender will not be able to answer any message from a recovering party since it is permanently crashed. Thus, one cannot state any validity for parties that are finally-down.

2. One can apply similar reasoning for demonstrating the optimality of the totality property. Evidently, parties which are finally-down cannot be forced to deliver a message. However, it is unnecessary to force the first delivering party to be finally-up in our definition of totality, or the sender to be finally-up in our definition of authenticity. These conditions turn out to be unnecessary for functional correctness, which will become clear when we present our protocol for reliable broadcast in the next section.
3. The consistency condition can be adopted directly from the standard model. Roughly, consistency does not have to be restricted to finally-up parties, because the considered parties deliver messages by assumption and it does not matter if they crash afterwards.
4. As mentioned above, we define efficiency in our hybrid model by using our new definition of d -uniformly bounded communication complexity where d corresponds to the protocol parameter bounding the number of crashes.

Summarizing the above remarks, we believe that our definition of reliable broadcast in hybrid models is the most general possible.

The only previous definition of reliable broadcast with crashes and recoveries has been given by Boichat and Guerraoui [2]. We argue now that our definition generalizes the strongest one of their notions to Byzantine faults.

Boichat and Guerraoui propose three modifications of the standard properties, depending on how the set of parties under consideration should be restricted. The first one, called *standard*, restricts a property to parties that never crash. The second one, called *uniform*, involves a restriction to parties that are finally-up. Finally, the third one, called *strongly uniform*, considers arbitrary parties. Strongly uniform totality (i.e., agreement) considers an arbitrary sender, but restricts to finally-up receivers.

As observed by Boichat and Guerraoui [2], strongly uniform validity is not a useful notion since it cannot be achieved by any algorithm (cf. the first remark). Thus, the strongest meaningful definition is to consider uniform validity and strongly uniform consistency, totality and authenticity properties. (The *termination* condition maps directly to *efficiency* in our model.) The resulting notion was called *strongly uniform reliable broadcast*.

Restricting our definition of reliable broadcast to honest parties corresponds precisely to strongly uniform reliable broadcast. Hence, Definition 2 generalizes the strongest definition of reliable broadcast in the crash-recovery model to Byzantine faults. Moreover, our definition is achievable as shown by the algorithm in the next section.

3.3 A Protocol for Reliable Broadcast

Throughout the following description, we assume that only persistent memory is used for storing data. In the subsequent section, we will investigate to what extent the use of persistent state can be reduced.

We demand that each party P_i stores all outgoing messages in a set \mathfrak{B} together with the intended recipient. Moreover, we assume that every message is stored at most once in \mathfrak{B} for every recipient. Let $\mathfrak{B}_l \subseteq \mathfrak{B}$ contain the messages of \mathfrak{B} intended for party P_l .

Protocol d -RBC for reliable broadcast is given in Figure 1. The pseudo-code notation used in the description lists the transitions taken by the party in dependence of the given input. If any of the *conditions* given in the clauses of the form **upon condition block** matches, the corresponding *block* is executed. We specify a *condition* in the form of *receiving messages* for protocol messages or *events* for input actions.

The protocol results from extending Bracha’s reliable broadcast protocol [3] by a recovery mechanism that works as follows. Whenever a party recovers, it sends a **help** message to all parties (such messages are *not* stored in \mathfrak{B}). When P_i receives such a message from P_j , it sends all messages in \mathfrak{B}_j to P_j , but P_i also keeps track of the number of received **help** messages and answers at most $d(k)$ times to every P_j .

Before proving that Protocol d -RBC implements reliable broadcast, we establish the following lemmas.

Lemma 1. *Let P_i be a finally-up party. Then every distinct message sent to P_i by another finally-up party P_l will be received by P_i in a non-crashed state, provided all associated messages are delivered.*

Proof. Assume that P_l sends a message m^* to P_i . Since P_l is permanently up after a particular point in time, it will send this message again at its last recovery. We now have to distinguish between two cases. If P_i is permanently up after this point in time, the message will be delivered

Protocol d -RBC for party P_i and tag $ID.j$

upon *initialization*:

$e_m \leftarrow 0; r_m \leftarrow 0 \quad (m \in \{0, 1\}^{Q(k)})$
 $c_l \leftarrow 0 \quad (l \in \{1, \dots, n\})$

upon $(ID.j, \text{in}, \text{r-broadcast}, m)$:

send $(ID.j, \text{r-send}, m)$ to all parties

upon *receiving* message $(ID.j, \text{r-send}, m)$ from P_l for the first time:

if $j = l$ **then**
padding-left: 4em;">send $(ID.j, \text{r-echo}, m)$ to all parties

upon *receiving* message $(ID.j, \text{r-echo}, m)$ from P_l for the first time:

$e_m \leftarrow e_m + 1$
if $e_m = \lceil \frac{n+t+1}{2} \rceil$ **and** $r_m \leq t$ **then**
padding-left: 4em;">send $(ID.j, \text{r-ready}, m)$ to all parties

upon *receiving* message $(ID.j, \text{r-ready}, m)$ from P_l for the first time:

$r_m \leftarrow r_m + 1$
if $r_m = t + 1$ **and** $e_m < \lceil \frac{n+t+1}{2} \rceil$ **then**
padding-left: 4em;">send $(ID.j, \text{r-ready}, m)$ to all parties
else if $r_m = 2t + f + 1$ **then**
padding-left: 4em;">output $(ID.j, \text{out}, \text{r-deliver}, m)$

upon $(ID.j, \text{in}, \text{recover})$:

send $(ID.j, \text{help})$ to all parties
padding-left: 2em;">send all messages in \mathfrak{B}

upon *receiving* message $(ID.j, \text{help})$ from P_l :

if $c_l \leq d(k)$ **then**
padding-left: 4em;"> $c_l \leftarrow c_l + 1$
padding-left: 4em;">send all messages of \mathfrak{B}_l

Figure 1: Protocol d -RBC for authenticated reliable broadcast (or the Byzantine generals problem) in our hybrid model.

by assumption and therefore also received in a non-crashed state. If P_i crashes again afterwards, consider now the recovery after its last crash. In this transition, it sends a **help** message to all participants, including P_i . This message will be delivered and P_i will not crash afterwards by assumption. Note that the check $c_i \leq d(k)$ of P_i must be successful since the adversary can crash P_i at most $d(k)$ times and it cannot duplicate **help** messages by assumption, so P_i will re-send m^* to P_i . Thus, m^* will be delivered and P_i can no longer be crashed afterwards, which concludes the proof. \square

Lemma 2. *Let $n, t, f \in \mathbb{N}$ be given such that $n > 3t + 2f$. Then $n - t - f \geq \lceil \frac{n+t+1}{2} \rceil$.*

Proof. Consider two numbers $a, b \in \mathbb{N}$ such that $a > \frac{b}{2}$. For odd b , this implies $a \geq \frac{b+1}{2} = \lceil \frac{b+1}{2} \rceil$, and for even b , this implies $a \geq \frac{b}{2} + 1 = \lceil \frac{b+2}{2} \rceil = \lceil \frac{b+1}{2} \rceil$. Now, $n - t - f = \frac{2n-2t-2f}{2} > \frac{n+t}{2}$ from the assumption, and since the left-hand side of this inequality is in \mathbb{N} , it follows $n - t - f \geq \lceil \frac{n+t+1}{2} \rceil$. \square

Lemma 3. *Let the sender P_j be finally-up. Then every finally-up party P_i will receive a message $(ID.j, \mathbf{r-ready}, m)$ for the value m proposed by the sender from at least $2t+f+1$ parties, provided that $n > 3t + 2f$ and all associated messages are delivered.*

Proof. According to Lemma 1, the initial message $(ID.j, \mathbf{r-send}, m)$ sent by P_j will eventually be received by every finally-up party, so each of them will output a message $(ID.j, \mathbf{r-echo}, m)$ to all parties. Thus, using Lemma 1, every finally-up party P_i will receive this message from all finally-up parties, i.e., from at least $n - t - f$ parties. Since $n - t - f \geq \lceil \frac{n+t+1}{2} \rceil$ by Lemma 2, it outputs (or has already output) a message $(ID.j, \mathbf{r-ready}, m)$ to all parties, either because of the above condition on the number of received **r-echo** messages, or because it has already received $t+1$ **r-ready** messages. All these **r-ready** messages will eventually be received by the finally-up parties according to Lemma 1, so P_i will receive at least $n - t - f \geq 3t + 2f + 1 - t - f = 2t + f + 1$ **r-ready** messages for the same value m . \square

Based on this, we can now formulate our main theorem.

Theorem 4. *Protocol RBC provides authenticated reliable broadcast in the hybrid model for $n > 3t + 2f$.*

Proof. We have to show validity, consistency, totality, authenticity, and efficiency.

Validity. Assume that a finally-up party has *r-broadcast* m tagged with $ID.j$. Now by Lemma 3, every finally-up party will receive at least $2t + f + 1$ **r-ready** messages for this value m , so it will *r-deliver* m by construction of the protocol.

Consistency. Towards a contradiction, assume that two honest parties *r-deliver* two messages m and m' . Thus, both parties must have received at least $2t + f + 1$ **r-ready** messages for m and m' , respectively. An honest party generates an **r-ready** message for m if and only if it has received $\lceil \frac{n+t+1}{2} \rceil$ **r-echo** messages or $t + 1$ **r-ready** messages already containing m . Thus, at least one honest party has sent an **r-ready** message containing m upon receiving $\lceil \frac{n+t+1}{2} \rceil$ **r-echo** messages; at most t of them are from corrupted parties. Similarly, some honest party must have received $\lceil \frac{n+t+1}{2} \rceil$ **r-echo** messages containing m' . Thus, there are at least $2\lceil \frac{n+t+1}{2} \rceil \geq n + t + 1$ **r-echo** messages with tag $ID.j$ and at least $n - t + 1$ among them from honest parties. But no honest party generates more than one distinct **r-echo** message by the protocol, yielding the desired contradiction.

Totality. When an honest party *r-delivers* a message m , then it has already received the message $(ID.j, \mathbf{r-ready}, m)$ from at least $2t + f + 1$ different parties. Therefore, at least $t + f + 1$ honest parties have already sent $\mathbf{r-ready}$ for m . Among these $t + f + 1$ parties, at least $t + 1$ parties are finally-up, so according to Lemma 1, every finally-up party will receive their $\mathbf{r-ready}$ messages provided all associated messages are delivered. Thus, every remaining finally-up party will also send an $\mathbf{r-ready}$ message for m , so every finally-up party will receive $n - t - f \geq 2t + f + 1$ $\mathbf{r-ready}$ messages for this m . Thus, every such party will *r-deliver* m .

Authenticity. The uniqueness of the *r-delivered* message is clear from the protocol. If a party delivers a message m , it must have received $2t + f + 1$ $\mathbf{r-ready}$ messages, i.e., at least $t + f + 1$ have been sent by honest parties. Since at most t parties might send an $\mathbf{r-echo}$ message for $m' \neq m$, no honest party will generate an $\mathbf{r-ready}$ message for m' ; thus, the message m must have been contained in the $\mathbf{r-send}$ message of P_j and was *r-broadcast* by P_j .

Efficiency. Let $d(k)$ denote the bound on the number of crashes. Then a protocol execution without any crashes has a message complexity of $O(n^2)$. If a recovery occurs, it causes computational overhead which can be bounded as follows. A recovering party sends at most $O(n^2)$ messages, and the parties receiving the \mathbf{help} message produce at most $O(n)$ reply messages each. Thus, the complexity is $O(n^2)$ for one crash and $O(dn^2)$ in total. However, there is one more problem we have to take care of. In contrast to the honest parties which can produce at most $d(k)$ \mathbf{help} messages in total, the corrupted parties are not restricted to this; they are only bounded by the running time of the adversary. However, only $d(k)$ \mathbf{help} messages of every corrupted party are answered by an honest party (the condition $c_l \leq d(k)$ in the last transition of Figure 1 ensures this). Thus, each corrupted party can cause an additional overhead of at most $d(k)n^2$. Putting it all together, we obtain a message complexity of $O(tdn^2)$. Finally, the length of each message is bounded by $Q(k)$, so we obtain a uniform polynomial bound on the communication complexity. □

4 Implementation Aspects

If we aim at actually implementing Protocol d -RBC, we can significantly speed up the algorithm by a suitable representation of the set \mathfrak{B} of every party. At first, note that if a message of the form $(ID.j, \cdot, m)$ is sent to an arbitrary participant for the first time, and therefore has to be stored in the set \mathfrak{B} , the message will additionally be output to every remaining participant. This holds by construction for all transitions except the transition that handles \mathbf{help} messages, which only sends messages to one party. However, these messages are already contained in \mathfrak{B} , so they do not have to be included again. Moreover, each and every honest party will only send $\mathbf{r-echo}$ messages for the same unique m , and the same holds for $\mathbf{r-ready}$ messages. Thus, we could efficiently represent the set \mathfrak{B} in persistent storage by specifying the message m , and whether an $\mathbf{r-echo}$ or an $\mathbf{r-ready}$ message has been sent so far.

However, we can do better. Note that the counters e_m and r_m can be used to remember whether an $\mathbf{r-echo}$ or an $\mathbf{r-ready}$ message has been sent so far. More precisely, we do not have to store the set \mathfrak{B} at all, but at executing a command of the form “re-send all messages of \mathfrak{B} ”, the party simply checks whether it has already received the initial $\mathbf{r-send}$ message for m . In this case, it outputs the $\mathbf{r-echo}$ message again to all parties for that m (or to the party P_l in case of a command “re-send \mathfrak{B}_l ”). Additionally, it checks if $e_m \geq \lceil \frac{n+t+1}{2} \rceil$ or $r_m \geq t + 1$

for some m . If this is the case, it outputs the corresponding **r-ready** message for this m . This implicit representation of \mathfrak{B} reduces the amount of persistent storage space even further. Since storing is generally slower than recomputing the needed messages, this results in significantly reduced overall running time.

More formally, each party has to persistently store at most $t + 1$ different values m together with e_m and r_m for each m . As both e_m and r_m range over $\{1, \dots, n\}$, the space complexity is bounded by $2(t + 1) \log n + (t + 1)Q(k)$. Moreover, the variables c_l counting the number of **help** messages received so far from party P_l have to be stored persistently. As the c_l range over $\{1, \dots, d(k)\}$ for all l , the induced space complexity is $n \log d(k)$. Finally, the implicit counters for checking the first-time delivery of every message type have to be stored. For each l we can implement this by a three-bit vector, which takes $3n$ bits of persistent storage space.

Altogether, the protocol needs $O(n \log d(k) + t \log n + tQ(k))$ persistent storage space for every honest party. When t is a constant fraction of n , this is $O(n \log(nd(k)) + nQ(k))$.

5 Conclusion

We have presented the first hybrid distributed system model that comprises both Byzantine faults and accidental crashes with subsequent recovery. The model is asynchronous and computational, i.e., all parties are constrained to perform only feasible, polynomial-time computations, which allows for reasoning about cryptography in a meaningful way. From a practitioner's point of view, the functionality of our model is both necessary and sufficient to capture a scenario that involves cryptography, malicious attacks, and processes that crash and recover later. Compared to the so-called proactive models that tolerate transient Byzantine faults, but require expensive cryptographic recovery protocols, our model is rich enough to allow for quite practical protocols.

As an illustration of the model, we have shown how to extend the definition of reliable broadcast to the hybrid model and presented a protocol that satisfies our definition. We have concluded with some considerations how to further improve the efficiency of the algorithm, which is important for practical purposes.

For future work, it will be interesting to consider other Byzantine fault-tolerant algorithms in our hybrid model, in particular randomized ones that guarantee probabilistic termination only. We already briefly sketched above how the definition of efficiency for randomized protocols can be formulated in analogy to the deterministic case. Adopting these protocols to the hybrid model is of high practical relevance.

Acknowledgments

This work was partially supported by the European IST Project MAFTIA (IST-1999-11583). However, it represents the view of the author(s). The MAFTIA project is partially funded by the European Commission and the Swiss Department for Education and Science.

References

- [1] M. K. Aguilera, W. Chen, and S. Toueg, "Failure detection and consensus in the crash-recovery model," *Distributed Computing*, vol. 13, no. 2, pp. 99–125, 2000.
- [2] R. Boichat and R. Guerraoui, "Reliable broadcast in crash-recovery model," in *Proc. 19th Symposium on Reliable Distributed Systems (SRDS 2000)*, pp. 32–40, 2000.

- [3] G. Bracha, “An asynchronous $[(n - 1)/3]$ -resilient consensus protocol,” in *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 154–162, 1984.
- [4] G. Bracha and S. Toueg, “Asynchronous consensus and broadcast protocols,” *Journal of the ACM*, vol. 32, pp. 824–840, Oct. 1985.
- [5] C. Cachin, “Modeling complexity in secure distributed computing.” International Workshop on Future Directions in Distributed Computing (FuDiCo), Bertinoro, Italy, June 2002.
- [6] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, “Asynchronous verifiable secret sharing and proactive cryptosystems,” in *Proc. 9th ACM Conference on Computer and Communications Security (CCS)*, 2002.
- [7] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, “Secure and efficient asynchronous broadcast protocols (extended abstract),” in *Advances in Cryptology: CRYPTO 2001* (J. Kilian, ed.), vol. 2139 of *Lecture Notes in Computer Science*, pp. 524–541, Springer, 2001. Extended version in Cryptology ePrint Archive, Report 2001/006, <http://eprint.iacr.org/>.
- [8] C. Cachin and J. A. Poritz, “Secure intrusion-tolerant replication on the Internet,” in *Proc. Intl. Conference on Dependable Systems and Networks (DSN-2002)*, pp. 167–176, June 2002.
- [9] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor, “Proactive security: Long-term protection against break-ins,” *RSA Laboratories’ CryptoBytes*, vol. 3, no. 1, 1997.
- [10] R. Canetti, S. Halevi, and A. Herzberg, “Maintaining authenticated communication in the presence of break-ins,” *Journal of Cryptology*, vol. 13, no. 1, pp. 61–106, 2000.
- [11] M. Castro and B. Liskov, “Practical Byzantine fault tolerance,” in *Proc. Third Symp. Operating Systems Design and Implementation (OSDI)*, 1999.
- [12] M. Castro and B. Liskov, “Proactive recovery in a Byzantine-fault-tolerant system,” in *Proc. Fourth Symp. Operating Systems Design and Implementation (OSDI)*, 2000.
- [13] T. D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.
- [14] J. Chang and N. F. Maxemchuck, “Reliable broadcast protocols,” *ACM Transactions on Computer Systems*, vol. 2, pp. 251–273, Aug. 1984.
- [15] M. Correia, L. C. Lung, N. F. Neves, and P. Veríssimo, “Efficient byzantine-resilient reliable multicast on a hybrid failure model,” in *Proc. 21st Symposium on Reliable Distributed Systems (SRDS 2002)*, pp. 2–11, Oct. 2002.
- [16] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi, “Failure detectors in omission failure environments,” Technical Report 96-1608, Department of Computer Science, Cornell University, Sept. 1996.
- [17] J. A. Garay and K. J. Perry, “A continuum of failure models for distributed computing,” in *Proc. 6th International Workshop on Distributed Algorithms (WDAG)*, vol. 647 of *Lecture Notes in Computer Science*, pp. 153–165, Springer, 1992.

- [18] O. Goldreich, *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.
- [19] V. Hadzilacos and S. Toueg, “Fault-tolerant broadcasts and related problems,” in *Distributed Systems* (S. J. Mullender, ed.), New York: ACM Press & Addison-Wesley, 1993. An expanded version appears as Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca NY, 1994.
- [20] M. Hurfin, A. Mostefaoui, and M. Raynal, “Consensus in asynchronous systems where processes can crash and recover,” in *Proc. 17th Symposium on Reliable Distributed Systems (SRDS’98)*, pp. 280–286, 1998.
- [21] L. Lamport, R. Shostak, and M. Pease, “The Byzantine generals problem,” *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 382–401, July 1982.
- [22] D. Malkhi, M. Merritt, and O. Rodeh, “Secure reliable multicast protocols in a WAN,” *Distributed Computing*, vol. 13, no. 1, pp. 19–28, 2000.
- [23] M. Pease, R. Shostak, and L. Lamport, “Reaching agreement in the presence of faults,” *Journal of the ACM*, vol. 27, pp. 228–234, Apr. 1980.
- [24] M. Reiter, “Secure agreement protocols: Reliable and atomic group multicast in Rampart,” in *Proc. 2nd ACM Conference on Computer and Communications Security*, pp. 68–80, 1994.
- [25] L. Zhou, F. B. Schneider, and R. van Renesse, “APSS: Proactive secret sharing in asynchronous systems.” Manuscript, Oct. 2002.