N-party BAR Transfer

(extended abstract of the MSc dissertation)

Xavier Vilaça
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisor: Professor Luís Rodrigues

***Abstract*—We introduce the N-party BAR transfer problem (NBART) that consists in reliably transferring arbitrarily large data from a set of $N_{\mathcal{P}}$ producers to a set of $N_{\mathcal{C}}$ consumers in the BAR model, i.e., in the presence of Byzantine, Altruistic, and Rational participants. We present an algorithm (ERA-NBART) that solves the problem for $N_{\mathcal{P}}, N_{\mathcal{C}} \geq 2f+1$ and assuming that there exists a trusted observer that gathers evidence to testify that the producers and consumers have participated in the transfer, where f is the maximum number of Byzantine processes in each of the producer and consumer sets. We do not impose limits on the number of Rational participants, although they can deviate from the algorithm to improve their utility. We show that ERA-NBART provides a Nash equilibrium. An alternative algorithm (LRA-NBART) is proposed that has a greater execution time and lower communication costs. We prove that this algorithm also provides a Nash equilibrium. We argue that a NBART algorithm is an important building block to support fault-tolerant distributed computations in peer-to-peer systems.**

I. Introduction

Peer-to-peer systems may be used to provide temporary or long-term storage services. Such services are useful in a number of settings. For instance, peer-to-peer systems can be used to process large volumes of data using volunteer computation, as illustrated by projects such as SETI@home [1] and, more recently, by the Boinc infrastructure that supports several computationally intensive research projects [2]. If such computations are performed using MapReduce [3], information produced by mappers needs to be transferred to the reducers or to intermediate storage. Volunteer storage nodes may not be willing to store data indefinitely, so they have to transfer data to other nodes after serving the system for some time. In any case, volunteers expect to be recognized for their contribution, for instance by being awarded credits that make them appear in a chart with the top contributors of the project.

In scenarios such as the ones listed above, a reliable protocol to transfer data from a set of producers to a set of consumers is an important building block. Any realistic service for this environment has to consider the existence of both Byzantine and Rational nodes, i.e., of nodes that deviate from the protocol, respectively, in an arbitrary way (Byzantine) and with the purpose of gaining some measurable benefit like being listed as top contributors without really executing jobs (Rational). A system model that captures the existence of these different kinds of participants is the Byzantine-Altruistic-Rational (BAR) model [4].

This work introduces the N-party BAR Transfer problem (NBART). This problem can be informally defined as follows. There are N producers and N consumers, which we generically call processes. Up to f processes of each of these sets can be Byzantine; the remainder are either Altruistic or Rational. All non-Byzantine producers have the same piece of arbitrarily large data that they have to transfer to all non-Byzantine consumers. Altruistic processes follow the protocol, Byzantine processes deviate arbitrarily from the protocol (e.g., omitting or sending modified messages), and Rational processes deviate from the protocol following a strategy to increase their utility. There is an abstract trusted observer that is not involved in the transfer, but that collects evidence about it. NBART is the problem of reliably transferring data from the producers to the consumers, while providing the trusted observer enough evidence to testify which processes participated in the transfer.

Systems not designed to cope with Rational behaviour may fall into the Tragedy of Commons [5]: the job is not done because all participants are Rational and aim for profit by not performing (part of) their role. To model Rational behaviour, we use an approach based on Game Theory [6]. The protocol executed by the processes is modelled as a game, in which each player (i.e., process) follows a strategy to increase its utility. To contradict this behaviour, an algorithm to solve NBART should provide a Nash equilibrium, so that no Rational process has an incentive to deviate from the protocol. We model the NBART problem as a strategic game, in which players choose a strategy simultaneously, once and for all [6], i.e., without knowledge of the others strategies and without the ability of changing it during the algorithm execution. This is not a restriction in our case as explained later. We do the usual assumption [7] that processes are risk-averse, i.e., that they do not follow a strategy that may put their profit at risk.

More precisely, the thesis makes the following contributions:

- It introduces the NBART problem, defining its main properties and challenges.
- It proposes two alternative synchronous algorithms that solve the NBART problem in an environment where processes are risk-averse:

  - The first algorithm, named Eager Risk-Averse NBART (ERA-NBART), has low execution time and high bit/message complexity.
  - The second algorithm, named Lazy Risk-Averse NBART (LRA-NBART), has low bit/message complexity and high time complexity.

The results of this work can be enumerated as follows:

- A proof that the ERA-NBART and LRA-NBART algorithms are correct when up to f processes of each of the sets of producers and consumers are Byzantine, as long as $N_P, N_C \geq 2f + 1$.
- A Game Theoretical analysis of ERA-NBART and LRA-NBART that proves that both algorithms provide a Nash equilibrium. We also claim that the algorithms provide a dominant strategy, therefore all Rational processes should follow them.
- A complexity analysis and a comparison between the proposed solutions.

The remaining of this document is organized as follows. Section II describes previous work related to the problems that were introduced. Section III gives a description of the NBART problem. The proposed solutions are presented in Section IV and V, and a comparison in terms of the complexity of the algorithms is performed in Section VI. We extend these solutions in Section VII. Finally, Section VIII concludes the extended abstract.

II. Related Work

The NBART problem is related to classical distributed systems problems such as Byzantine Agreement (BA), Reliable Broadcast (RB), Terminating Reliable Broadcast (TRB), and Interactive Consistency (IC) [8], [9], [10]. A first and major difference is that these algorithms are executed among a single set of processes, while NBART is about communication and agreement between two sets: producers and consumers. In that sense there is some resemblance with Paxos with its three process roles – proposers, acceptors, learners – but in NBART all producers are proposers of the same value, a notion that does not exist in Paxos [11]. An algorithm for solving NBART might be implemented by running N instances of algorithms that solved these problems, or even a single one in the case of IC. However, these solutions would be very inefficient in terms of message, time, and bit complexity because they would not exploit the fact that all (non-Byzantine) producers send the same data. For instance, in the case of IC the consumers would receive a vector with $N - f$ to N copies of the data sent by the producers, which we assume is large, so it would be very inefficient, at least in terms of bit complexity. Furthermore, these problems do not consider the BAR model, only Byzantine and Altruistic processes. If there were Rational processes, algorithms that solved these problems would not satisfy their properties. The same discussion applies to classic probabilistic reliable broadcast algorithms [12], [13]. Although they reduce the number of messages sent when compared with parallel executions of BA, RB or TRB, these works do not tolerate the BAR model. Besides, these algorithms only provide probabilistic guarantees.

There is a recent trend in Byzantine fault-tolerant algorithms that has also some relation to our work. Several papers presented implementations of registers with different concurrency semantics based on Byzantine quorum systems [14], [15]. Others presented algorithms to implement state machine replication, a generic solution to implement fault-tolerant distributed services [16], [17], [18]. In both cases the objective is to ensure that Byzantine nodes are unable to disrupt the consistency of the data stored in the servers or the service provided by the servers. In contrast, our work aims at ensuring the transference of a correct value from a set of nodes that produce the data independently, although following a deterministic function, to another set of nodes which have to determine which is the correct data. A third set of papers presented Byzantine fault-tolerant consensus algorithms for asynchronous systems, which might also be used as building blocks of less efficient solutions of NBART [19], [20], [21]. Again, none of these works considers the BAR model.

Some works applied Game Theory to problems involving both Rational and Byzantine players. Eliaz introduced the notion of k-Fault Tolerant Nash equilibrium (k-FNTE), as an equilibrium in which no Rational participant has any incentive to unilaterally deviate from the expected behaviour, with up to k players whose strategy is arbitrary [22]. This concept was applied to auctions. A Byzantine Nash equilibrium has also been studied in the context of a virus inoculation game [23]. Neither of these works were extended to more complex distributed games with communication between participants. Abraham et al. extended the work of [22] by introducing the notion of (k, t)-robustness, where k is the maximum number of colluding Rational participants and t is the upper limit to the number of Byzantine players [24]. The authors propose a solution for secret sharing that is (k, t)-robust. On the contrary to our work, they assumed that the utility of each player depends only on the output of the algorithm, i.e., on the successful delivery of the shared secret, therefore ignoring communication costs. It has been proved that no non-trivial distributed protocol for which Rational nodes take into consideration communication costs can be (k, t)-robust [7].

The Byzantine-Altruistic-Rational (BAR) model was proposed by Aiyer et al. as an abstraction for capturing these three distinct behaviours of processes [4]. The authors also proposed a general three-tied architecture for developing BAR-tolerant protocols, in cooperative distributed systems that span Multiple Administrative Domains (MAD). The first two levels of the proposed architecture implement a Replicated State Machine using a BAR-tolerant TRB protocol [7] and a mechanism than enforces periodic work and guarantees responses. Although this architecture might be used to solve the NBART problem, the use of the TRB protocol for transferring arbitrarily large data is too costly and the guaranteed response mechanism requires the active participation of a witness, which must be either a centralized entity or implemented through message broadcast to all the remaining nodes. Furthermore, the proposed mechanisms are based on the long-term cooperation between participants modelled as a repeated game [6], which is not the case of the NBART problem. Cost balancing mechanisms were used to deny cost savings to processes that fail to send expected messages, but the overhead of balancing the cost of transferring an arbitrarily large value is too high.

The same authors [7] have shown that the Dolev and Strong's TRB protocol [9] can be changed to provide a Nash equilibrium in the BAR model using ∞-tit-for-tat mechanisms [25]. The problem is modelled as a repeated game with an infinite number of rounds. Each round a different participant runs an instance of the protocol to broadcast its information to all the remaining non-Byzantine participants. They proved that Rational participants cannot expect any increase in their utility by omitting messages, even if a fraction of the participants is Byzantine. In this work we are interested in large peer-to-peer networks in which it is unlikely that the same participants interact more than once. For that reason we do not consider repeated executions, but model the algorithm instead in terms of a strategic game in which players interact only once. Therefore, in our case it is not possible to apply the incentive mechanisms of [4], [26], [27] based on tit-for-tat. Furthermore, none of these works addresses the problem of transferring an arbitrarily large value without using an active witness or direct reciprocity.

The BAR model has also been used with gossip peer-to-peer data dissemination algorithms [26], [27]. These algorithms are not directly applicable to solve NBART as they assume that the source of the information is trusted and provide no guarantee that the disseminated information reaches its destination. Furthermore, data transfer between each pair of nodes is performed using direct reciprocity in a fair exchange process. This requires that each Rational participant has incentives to transfer data if it expects to receive an equivalent contribution from its peer. In addition, the pestering mechanism of BAR Gossip [26] only provides a Nash equilibrium if a certain fraction of the participants are Altruistic [28]. In NBART, consumers do not possess any data that may serve as currency to pay the producers for the transfer, and no assumption is made about the presence of Altruistic participants.

To the best of our knowledge, no previous work in the literature addresses the problem of transferring an arbitrarily large data from a set of producers to a set of consumers in a non-repeated game, where processes can be Byzantine, Rational, or Altruistic.

III. System Model and Problem Statement

A. System Model

The NBART problem involves a set of *producers* $\mathcal{P}$ of cardinality $N_{\mathcal{P}}$ and a set of *consumers* $\mathcal{C}$ of cardinality N_c. To simplify the description of our algorithms, in this thesis, we start by assuming that the cardinality of both sets is the same, i.e., $N_{\mathcal{P}} = N_{\mathcal{C}} = N$. Then, in Section VII, we extend this work to the case where it may be true that $N_{\mathcal{P}} \neq N_{\mathcal{C}}$. We do not address the problem of forming these sets, in this paper. However, we assume that this mechanism ensures with high probability that the number of Byzantine processes is upper bounded and that processes cannot influence this mechanism. There is also a special abstract process called *trusted observer* (TO). We use the words *processes* or *participants* to designate these entities. Sometimes we use the word *players* to designate producers and consumers, when we model their interaction as a game.

We assume that the system is synchronous (there are maximum communication and processing delays) and that all processes are fully connected by authenticated reliable channels. This is a reasonable assumption as we require that the transfer may terminate after a finite period of time such that Rational processes may have some guarantees that they will be eventually rewarded. However, it is not strictly necessary for the communication and processing delays to have a known upper bound. Nevertheless, in order to simplify the description of our algorithms, we will make that assumption. We also assume that each process has a public-private key pair and that there is a public-key infrastructure in place, so every process has access to the public key of all others. Each process has access to a collision-resistant hash function (*hash*) and a signature function based on public-key cryptography (*sign*, *verifysig*).

Participants can be Byzantine, Altruistic, and Rational, in accordance with the BAR model. We assume that up to f elements of each of the $\mathcal{P}$ and $\mathcal{C}$ sets can be Byzantine. In Section VII, we will extend this assumption to the case where $f_{\mathcal{P}}$, the upper bound on the number of Byzantine producers, is different from $f_{\mathcal{C}}$, the upper bound on the number of Byzantine consumers. Any number of consumers and producers can be Altruistic or Rational. The trusted observer TO always follows its protocol.

An Altruistic process is one that follows the protocol. A Byzantine process can deviate arbitrarily from its behaviour, e.g., by sending or not sending certain messages, or by sending messages in a format or with content that is not according to the protocol. Byzantine processes however are not able to break the cryptographic mechanisms used in the algorithm (e.g., they are not able to generate signatures on behalf of Altruistic or Rational processes).

A Rational process is one that aims at maximizing a *utility function*, defined in terms of *benefits* and *costs*. A producer has a benefit by proving to the *TO* that it has contributed to the transfer; it incurs on the cost of sending the data. Consumers send to the *TO* acknowledgements of the reception of the data. A consumer benefits by obtaining the data and proving its reception to the *TO*; it incurs the costs of receiving and processing messages and sending the acknowledgements to the *TO*. We assume that there is no collusion among Rational processes.

B. The NBART Problem

The NBART problem can be defined as follows. Each producer p has a value (or data) of arbitrary size v_p such that, for any two non-Byzantine producers p_i and p_j, $v_{p_i} = v_{p_j} = \bar{v}$. Sometimes we refer to this value as the *correct value*, to denote that it is the value held by all non-Byzantine producers.

The algorithm terminates successfully when every non-Byzantine consumer consumes $\bar{v}$. A consumer c is said to *consume* value v_c when the primitive *consume*(c, v_c) is called. All non-Byzantine producers start the algorithm by producing value $\bar{v}$. A producer p is said to *produce* value v_p by calling the primitive *produce*(p, v_p). The *TO* is said to *produce evidence* about the transfer by calling primitive *certify*$($*TO*, *evidence*$)$. There are also two predicates *hasProduced(evidence,* p_i*)* and *hasAcknowledged(evidence,* c_j*)* that take as input the evidence produced by the *TO* to indicate, respectively, if producer p_i participated in the NBART and if consumer c_j notified the reception of the correct value. The problem consists informally in i) transferring the value from the producers to the consumers; and ii) providing evidence about the transfer. More formally the problem is defined in terms of the following properties:

- **NBART 1** *(Validity):* If a non-Byzantine consumer consumes v, then v was produced by some non-Byzantine producer.
- **NBART 2** *(Integrity):* No non-Byzantine consumer consumes more than once.
- **NBART 3** *(Agreement):* No two non-Byzantine consumers consume different values.
- **NBART 4** *(Termination):* Every non-Byzantine consumer consumes a value.
- **NBART 5** *(Evidence):* The trusted observer produces evidence about the transfer.
- **NBART 6** *(Producer Certification):* If producer p is non-Byzantine, then *hasProduced(evidence,* p*)* is *true*.
- **NBART 7** *(Consumer Certification):* If consumer c is non-Byzantine, then *hasAcknowledged(evidence,* c*)* is *true*.

With these definitions in mind, we can provide a more precise characterization of the *benefits* that Rational nodes aim to obtain. The benefit of a producer p is to have *hasProduced(evidence,* p*) true*. The benefit of a consumer c is twofold: i) to obtain the correct value and ii) to have *hasAcknowledged(evidence,* c*) true*.

IV. ERA-NBART

We now present ERA-NBART. The part of the algorithm executed by the producers, consumers, and trusted observer is presented respectively in Alg. 1, Alg. 2, and Alg. 3. The algorithm requires $N \geq 2f + 1$ producers and consumers.

The algorithm aims at ensuring that each consumer receives the value and can decide which is the correct value, in case it receives several different values (e.g., due to Byzantine producers). To satisfy this goal, each producer is not required to send a copy of the (possibly large) value to every consumer. In fact, it is enough that it sends the value to $f + 1$ consumers and a signed hash of the value to the remaining $N - f - 1$ consumers.

We define a deterministic function that returns the set of consumers that receive a copy of the value from producer p_i, denoted *consumerset*$_i$, as: *consumerset*$_i = \{c_j | j \in [i...(i + f) \ mod \ N]\}$. The intuition behind this function is that the consumers are seen as a circular space where each producer is responsible for sending the value it has computed to a set of consecutive consumers of cardinality $f + 1$, which are shifted from one another by one position. For instance, with $N = 3$, the *consumerset* of p_1, p_2 and p_3 are $\{c_1, c_2\}$, $\{c_2, c_3\}$, and $\{c_3, c_1\}$, respectively. It is also useful to define the inverse of this function as: *producerset*$_j = \{p_i | c_j \in$ *consumerset*$_i\}$.

A. Overview of the Algorithm

We model the operation of the algorithm in *rounds*. The round of a process is increased as result of a *nextRound* event. The system is synchronous, so non-Byzantine processes have their clocks synchronized and the *nextRound* event occurs simultaneously in all of them. The synchrony of the system and reliability of the channels ensure that if in response to event *nextRound*(n) a non-Byzantine process sends a message to another non-Byzantine process, that message is delivered to the destination before *nextRound*$(n+1)$ is triggered. This implies that *nextround* events are triggered periodically with a period greater than the worst case latency of communication channels.

The algorithm executes in three rounds. In the first round (round 0), the producers send values or hashes to consumers. In the second round, consumers send certificates of reception to the trusted observer. In the third round, the trusted observer produces the evidence.

B. Algorithm in Detail

In round 0, a producer computes the hash of the value and signs it (Alg. 1, lines 6-8). When the first round starts, it sends the value, its hash, and signature to the consumers in *consumerset*$_i$ (lines 11-13), but only the hash and signature to the remaining consumers (lines 14-16).

A consumer starts by waiting for signed values and hashes from producers during round 1 (Alg. 2, lines 9 and 15). Each value, hash, and signature received is stored in an array named *values* (lines 14 and 19). If a process does not send the message it was supposed to during this round, or if the hash or signature are not valid, the entry in the values set for that producer remains with the special value $\perp$, which will serve to build a proof of misbehaviour for the *TO* (if $f + 1$ consumers provide similar certificates).

When round 1 ends, the consumer picks the value v such that *hash*(v) appears in more than f positions of the array (lines 22-23). There are at most f faulty producers in the system, thus there is at most one value that matches this condition. Then, the consumer prepares the *confirm* array to serve as a *certificate* that vouches for the correct or incorrect behaviour of all producers, and that simultaneously

Algorithm 1: NBART Algorithm (producer p_i)

```
01 upon init do
02   myvalue := $\perp$;
03   myhash :=$\perp$;
04   myhashsig := $\perp$;
05   round := 0;

06 upon produce($p_i$,myvalue) $\wedge$ round = 0 do
07   myhash := hash(myvalue);
08   myhashsig := sign ($p_i$, myhash);

09 upon nextRound $\wedge$ round = 0 do // start of round 1
10   round := 1;
11   msgsig := sign ($p_i$, VALUE || myvalue || myhash || myhashsig);
12   forall $c_j \in consumerset_i$ do
13     send ($p_i$, $c_j$, [VALUE, myvalue, myhash, myhashsig, msgsig])
14   msgsig := sign ($p_i$, SUMMARY || myhash || myhashsig);
15   forall $c_j \in \mathcal{C} \backslash consumerset_i$ do
16     send ($p_i$, $c_j$, [SUMMARY, myhash, myhashsig, msgsig]);
```

Algorithm 2: NBART Algorithm (consumer c_j)

```
01 upon init do
02   myvalue :=$\perp$;
03   myhash:=$\perp$;
04   confirm := $[\perp]^N$;
05   values := $[\perp]^N$;
06   round := 0;

07 upon nextRound $\wedge$ round = 0 do // start of round 1
08   round := 1;

09 upon deliver ($p_i$, $c_j$, [VALUE, pvalue, phash, phashsig, msgsig]) $\wedge$ round = 1 do
10   if ($p_i \in producerset_j$)then
11     if verifysig($p_i$, VALUE || pvalue || phash || phashsig, msgsig)then
12       if verifysig($p_i$,phash, phashsig) then
13         if verifyhash(pvalue, phash) then
14           values[$p_i$] := $\langle$pvalue, phash, phashsig$\rangle$;

15 upon deliver ($p_i$, $c_j$, [SUMMARY, phash, phashsig, msgsig]) $\wedge$ round = 1 do
16   if ($p_i \notin producerset_j$)then
17     if verifysig($p_i$, SUMMARY ||phash || phashsig, msgsig) then
18       if verifysig($p_i$, phash, phashsig) then
19         values[$p_i$] := $\langle\perp$, phash, phashsig$\rangle$;

20 upon nextRound $\wedge$ round = 1 do // start of round 2
21   round := 2;
22   myhash := $h : \#(\{p|value[p] = \langle *, h, * \rangle\}) > f$.
23   myvalue := $v : \{p|value[p] = \langle v, myhash, * \rangle\}$.
24   forall $p_i$: values[$p_i$] = $\langle$*, myhash, *$\rangle$ do
25     confirm[$p_i$] := $\langle$values[$p_i$].hash, values[$p_i$].signature$\rangle$;
26   confsig := sign ($c_j$, confirm);
27   msgsig := sign ($c_j$, CERTIFICATE||confirm||confsig);
28   send ($c_j$, TO, [CERTIFICATE, confirm, confsig, msgsig]);
29   consume ($c_j$, myvalue);
```

proves that it has received and picked the correct value as described below (lines 24-25). For each producer p_i, the consumer either stores in *confirm*: i) the received hash and corresponding signature (extracted from the values set) or ii) the special value $\perp$ when no data, or incorrect data, was received from that producer. The consumer then signs this data structure with its private key and sends it as a proof of reception to the trusted observer (lines 26-28). The consumer terminates its local execution of the algorithm by outputting the value (line 29).

The trusted observer waits for a certificate from each consumer in round 2 (Alg. 3, line 6). The certificates are collected in an array called *evidence* (line 9). In the end, the trusted observer produces the array as evidence (line 11).

Algorithm 3: NBART Algorithm (trusted observer *TO*)

```
01 upon init do
02   evidence:= $[\perp]^C$;
03   round := 0;

04 upon nextRound $\wedge$ round $< 2$ do
05   round := round+1;

06 upon deliver ($c_j$, TO, [CERTIFICATE, confirm, confsig, msgsig]) $\wedge$ round = 2 do
07   if verifysig ($c_j$, CERTIFICATE||confirm||confsig, msgsig) then
08     if verifysig ($c_j$, confirm, confsig) then
09       evidence[$c_j$] := $\langle$confirm, confsig$\rangle$;

10 upon nextRound $\wedge$ round = 2 do // start of round 3
11   certify (TO, evidence);
```

Considering the data structure that is created by the trusted observer as evidence, we can now define with more detail the predicates *hasProduced* and *hasAcknowledged*. Let $h(v)$ denote the hash of the value v and let $s_{p_k}(h(v))$ denote the hash of v signed by the producer p_k:

- *hasProduced(evidence, p_i)* is true if the following condition holds: there are at least $N - f$ consumers $c_k \in \mathcal{C}$: *evidence*$[c_k][p_i] = \langle h(v), s_{p_i}(h(v)) \rangle$. It is false otherwise.
- *hasAcknowledged(evidence, c_j)* is true if the following conditions hold: it exists a set of producers, named *correctset*$_j$, such that $\#$*correctset*$_j \geq N - f$ and for $\forall p_k \in$ *correctset*$_j$ *hasProduced*(*evidence*, p_k) is true and *evidence*$[c_j][p_k] = \langle h(v), s_{p_k}(h(v)) \rangle$. It is false otherwise.

The algorithm does not require the observer to actively participate in the execution of the algorithm. Furthermore, the verification process performed by the trusted observer is independent for each transfer. Therefore many instances of NBART can be executed in parallel under the jurisdiction of one or more trusted observers, without the trusted entity being a single point of failure or a bottleneck.

C. Analysis

The analysis of the algorithm has three parts. First, we prove its correctness. Then, we demonstrate that it is a Nash equilibrium. Finally, we perform a complexity analysis in terms of communication costs.

1) Correctness: To prove the correctness of the algorithm, we start by assuming that all non-Byzantine processes follow the specified behaviour. Then, we prove that ERA-NBART fulfils each of the NBART properties in a sequence of lemmas. Then, we derive the following theorem from those lemmas, proving that ERA-NBART is correct in the presence of Byzantine and Altruistic behaviour, as long as $N \geq 2f + 1$:

Theorem 4.1: (Correctness) If all non-Byzantine participants follow the protocol, then the provided algorithm solves the NBART problem defined in terms of properties NBART 1-7.

2) Game Theoretical Analysis: To prove that the protocol provides a Nash equilibrium, we model the NBART problem as a strategic game [6], where each player (process) decides its *strategy* (or plan of action) once and it remains valid for all its actions during the execution of NBART. These decisions about the strategy are made simultaneously and, as Rational players do not collude among themselves, without knowledge of the strategies selected by other players. The profile of strategies, denoted by $\vec{\sigma}$, is the correspondence between each player and its strategy.

We considered the following possible behaviours. Altruistic producers send $\mathit{hash}(v)$ to all consumers and the value v to the consumers of $\mathit{consumerset}_i$. Rational producers send $\mathit{hash}(v)$ to any subset of $\mathcal{C}$ and the value to any subset $\mathcal{C}' \subseteq \mathcal{C}$. Altruistic consumers process all the information received from producers, send it to the *TO*, and consume one value. Rational consumers may or may not: consume a value, process all the values or hashes received from producers, and send the received information to the *TO*. Byzantine players follow an arbitrary strategy. Notice that these are pure strategies, that is, the decisions about which strategy to follow is deterministic.

The main goal of this analysis is to prove that ERA-NBART provides a Nash equilibrium. That is, for the profile of strategies $\vec{\sigma}$ where all Rational processes follow the algorithm, then no process can increase its expected benefit by unilaterally deviating from the strategy specified by $\vec{\sigma}$.

To provide a complete proof that neither the producers nor the consumers benefit from deviating from the protocol, we proved that the expected benefits obtained by each producer p_i are always equal to 0, whenever p_i does not send the signature of the correct value to all consumers and does not send the value to all consumers of $\mathit{consumerset}_i$ This allows us to prove the following theorem.

Theorem 4.2: No producer has any incentives to unilaterally deviate from the protocol.

We also prove that the expected benefit obtained by each consumer c_j is 0 whenever c_j: does not process and stores all the valid signatures sent by producers, and does not send those signatures to TO. This allows us to prove the following theorem.

Theorem 4.3: No consumer has any incentives to unilaterally deviate from the protocol.

The following Theorem concludes that the algorithm provides a Nash equilibrium.

Theorem 4.4: **(Nash equilibrium)** The profile of strategies $\vec{\sigma}_M$ where every player follows the protocol is a Nash equilibrium.

V. LRA-NBART

In this section, we present a variant of ERA-NBART that aims at minimizing the information exchanged in the network at the cost of increasing the latency of the algorithm. This algorithm is depicted for producers, consumers, and TO, respectively, in Alg. 4, Alg. 5, and Alg. 6.

ERA-NBART forces each producer to send the value to $f + 1$ consumers. One possible alternative consists in requiring only one transfer per consumer. This minimizes the overhead imposed by transferring the value. Unfortunately, this change requires the algorithm to execute additional rounds. Hence, there is a trade-off associated with this choice.

A. Algorithm

We change the algorithm such that the producer, in the first round, only sends the hash of the value and corresponding signature to all consumers (Alg. 4, lines 11-13). This allows all consumers to obtain the following information: i) which is the hash of the correct value (Alg. 5, line 16) and ii) which producers claim to have the correct value.

Then, the consumer selects a producer using a deterministic function and requests a copy of the value from that producer (Alg. 5, lines 17-20 and 30-33). If the producer is Rational, it should send the value to the consumer (Alg. 4, lines 16-21), otherwise the corresponding consumer will set the position of the array $hashes$, corresponding to the producer, to the value $\perp$ (Alg. 5, lines 28-29). However, if the producer is faulty, it may not reply to the request from the consumer. In the worst case, the consumer may have to request the value to $f + 1$ different producers, which delays the transfer. For load balancing, different consumers will ask the values from different producers in different orders. The sequences that map consumers to producers and producers to consumers, in each round, are named *producerseq* and *consumerseq* for consumers and producers, respectively. Their definitions are similar to the definitions of *consumerset* and *producerset* in ERA-NBART, with the only difference that in LRA-NBART these functions map each index between 0 and f to a process. With this, each consumer should request the value in round r to the producer identified by $\mathit{producerseq}[r-2]$, since round 2 corresponds to the index 0. Likewise, each producer should only reply to the consumer identified by $\mathit{consumerseq}[r-2]$, in round r. The definitions of these data structures are the following: $\mathit{producerseq}_j = [r \rightarrow p_i | r \in [0 \dots f], i = j{+}r \ \mathit{mod}\ N]$; and $\mathit{consumerseq}_i = [r \rightarrow c_j | r \in [0 \dots f], \mathit{producerseq}_j[r] = p_i]$

As a result of this strategy, after the round 1, all processes execute $f + 1$ rounds where values may be potentially transferred from producers to consumers. Consumers, stop requesting the value as soon as they received it from a non-Byzantine producer. However, they still wait for the round $f + 3$ to send a certificate to the trusted observer. At round $f{+}3$ all consumers send the certificates to TO and terminate the protocol (Alg. 5, lines 36-39). As with ERA-NBART, the trusted observer terminates the protocol one round later, after receiving all the certificates (Alg. 6, lines 6-9) and after certifying each participant (Alg. 6, line 11).

Note that a Byzantine consumer may pretend to not have received the value, and request the value from a different producer in all $f + 1$ rounds. Thus, in the worst case, a non-Byzantine producer may still need to send $f + 1$ copies of the data. Nevertheless, the total cost of transferring the value, even in face of a worst case adversary, is still smaller than in ERA-NBART, and much smaller in failure-free runs.

Algorithm 4: LRA-NBART (producer p_i)

```
upon init do
  myvalue := $\perp$;
  myhash := $\perp$;
  myhashsig := $\perp$;
  round := 0;

upon produce($p_i$,myvalue) $\wedge$ round = 0 do
  myhash := hash(myvalue);
  myhashsig := sign ($p_i$, myhash);

upon nextRound $\wedge$ round = 0 do // start of round 1
  round := 1;
  msgsig := sign ($p_i$, SUMMARY || myhash || myhashsig);
  forall $c_j \in C$ do
    send ($p_i$, $c_j$, [SUMMARY, myhash, myhashsig, msgsig])

upon nextRound $\wedge$ round $> 0 \wedge$ round $< f + 3$ do
  round := round +1;

upon deliver ($c_j$, $p_i$, [REQUEST, rhash, reqsig]) $\wedge$ round $> 1 \wedge$ round $\leq f + 2$ do
  if verifysig ($c_j$, REQUEST ||rhash, reqsig) then
    if consumerseq$_i$[round −2] = $c_j$ then
      if rhash = myhash then
        msgsig := sign ($p_i$, VALUE || myvalue);
        send ($p_i$, $c_j$, [VALUE, myvalue, msgsig])
```

Algorithm 5: LRA-NBART (consumer c_j)

```
upon init do
  myhash :=$\perp$;
  myvalue :=$\perp$;
  hashes := $[\perp]^P$;
  targets := producerseq$_j$;
  source :=$\perp$;
  round := 0;

upon nextRound $\wedge$ round $= 0$ do // start of round 1
  round := 1;

upon deliver ($p_i$, $c_j$, [SUMMARY, phash, phashsig, msgsig]) $\wedge$ round $= 1$ do
  if verifysig($p_i$, SUMMARY || phash || phashsig, msgsig)then
    if verifysig ($p_i$, phash, phashsig) then
      hashes[$p_i$] := $\langle$phash, phashsig$\rangle$;

upon nextRound $\wedge$ round $= 1$ do // start of round 2
  round := 2;
  myhash := $h : \#(\{p | hashes[p] = \langle h, *\rangle\}) > f$.
  source := removefirst(targets);
  if hashes[source] = $\langle myhash, *\rangle$ then
    msgsig := sign ($c_j$,REQUEST || myhash);
    send($c_j$, source, [REQUEST, myhash, msgsig])

upon deliver ($p_i$, $c_j$, [VALUE, pvalue, msgsig]) $\wedge$ round $\geq 2 \wedge$ round $\leq f + 2$ do
  if $p_i$ = source then
    if verifysig ($p_i$, VALUE || pvalue, msgsig) then
      if hash(pvalue) = myhash then
        myvalue := pvalue;

upon nextRound $\wedge$ round $\geq 2 \wedge$ round $< f + 2$ do
  round := round + 1;
  if myvalue = $\perp$ then
    hashes[source] := $\perp$;
    source := removefirst(targets);
    if hashes[source] = $\langle myhash, *\rangle$ then
      msgsig := sign ($c_j$,REQUEST || myhash);
      send($c_j$, source, [REQUEST, myhash, msgsig])

upon nextRound $\wedge$ round $= f + 2$ do // start of round $f + 3$
  round := round + 1;
  confsig := sign ($c_j$, hashes);
  msgsig := sign ($c_j$, CERTIFICATE ||hashes||confsig);
  send ($c_j$, TO, [CERTIFICATE, hashes, confsig, msgsig])
  consume ($c_j$, myvalue);
```

Algorithm 6: LRA-NBART (trusted observer *TO*)

```
upon init do
  evidence:= $[\perp]^C$;
  round := 0;

upon nextRound $\wedge$ round $< f + 3$ do
  round := round+1;

upon deliver ($c_j$, TO, [CERTIFICATE, confirm, confsig, msgsig]) $\wedge$ round $= f + 3$ do
  if verifysig ($c_j$, CERTIFICATE ||confirm||confsig, msgsig) then
    if verifysig ($c_j$, confirm, confsig) then
      evidence[$c_j$] := $\langle$confirm, confsig$\rangle$;

upon nextRound $\wedge$ round $= f + 3$ do // start of round $f + 4$
  certify (TO, evidence);
```

B. Analysis

In this thesis, we also prove the correctness of LRA-NBART, and we show that this algorithm provides a Nash equilibrium. The proofs are very similar to the proofs included in the analysis of ERA-NBART.

1) Correctness: As a basis for proving the correctness of LRA-NBART, we prove the following Lemmas.

Lemma 5.1: At the beginning of round 2, every non-Byzantine consumer possesses at least $f + 1$ hashes of the correct value v.

Lemma 5.2: Every non-Byzantine consumer obtains the value v until the beginning of round $f + 3$.

Then, as in ERA-NBART, we prove that the algorithm fulfils the NBART properties, one by one in different lemmas, and we conclude the following theorem.

Theorem 5.3: (Correctness) If all non-Byzantine participants obey the protocol, then the variant algorithm solves the NBART problem defined in terms of properties NBART 1-7.

2) Game Theoretical Analysis: To prove that it is in every Rational player best interest to obey the protocol, for LRA-NBART, we use the same notation as in ERA-NBART and the same definition of the utility function for a strategic game.

To show that LRA-NBART is a Nash equilibrium for risk-averse players, the provided proofs are almost identical to the proofs provided in ERA-NBART. The only significant difference is on showing that producers do not refuse requests from consumers instead of proving that they send v to their $consumerset$. More precisely, it is proved that each producer p_i does not obtain any benefits if it does not send the signature of the correct value to all consumers, or if it does not reply with the correct value to some request performed by a consumer c_j in round r, when *producerset*$_i[r] = c_j$. It is also proved that if p_i replies to a request performed by a consumer c_j when *producerset*$_i[r] \neq c_j$, its expected utility is lower than the expected utility of following the algorithm.

Regarding consumers, it is proved for each consumer c_j that the expected benefits are equal to 0 whenever c_j does not process and store all the valid signatures sent by producer, and does not send those signatures to TO.

VI. Complexity Analysis

This section compares the algorithms in terms of the following parameters:

- **Time complexity** (TC): Total number of rounds.
- **Message complexity** (MC): Total number of messages sent by non-Byzantine producers. We do not consider messages sent by Byzantine processes because their number cannot be upper bounded.
- **Bit complexity** (BC): Total number of bits sent by non-Byzantine producers. We also do not take into consideration bits sent by Byzantine processes for the same reasons stated previously.
- **Processing complexity** (PC): Maximum processing effort per producer, per consumer and per TO, performed by a non-Byzantine process, that includes the costs of verifying and computing signatures and hashes. This complexity represents the computational effort, which can be measured, for instance, in CPU cycles. We ignore the costs of simple operations, such as attributions, simple arithmetic operations, and counting operations.
- **Storage complexity** (SC): Maximum cost of storing the value, signatures, and hashes, per producer, consumer, and TO, during the execution of the algorithm. This takes into consideration not only the total number of stored bytes, but also the time during which processes must store that information. Therefore, we assume that there are fixed costs for storing a value, and a pair hash signature, per round. The SC is given by: i) the total number of stored values times the number of rounds each process stores those values times the fixed cost of storing the value per round; and ii) the total number of pairs with an hash and signature times the number of rounds times the cost of storing a single pair during a round.
- **Packet Transmission Complexity** (PTC): Cost of sending and receiving information. This is the total cost of transmitting or receiving all the data packets of the messages. We assume that there is a fixed cost for sending or receiving a single data packet, that includes all the underlying costs to this operation, such as storage in local buffers, CPU cycles for transferring the data, energy consumption, among others.

We compare ERA-NBART and LRA-NBART in the following three scenarios:

- Optimistic Scenario: $b_P = b_C = 0$ - provides evaluation for an environment where there are no Byzantine processes.
- Crash-Faults Scenario: $b_P = t_P$, $b_C = t_C$, and $b_P, b_C \geq 0$ - provides evaluation for an environment where completely arbitrary faults are rare.
- Arbitrary-Faults Scenario: $b_P = a_P$, $b_C = a_C$ and $b_P, b_C \geq 0$ - provides evaluation for an hostile environment.

We show that the time complexities are 4 and $f + 5$ rounds in ERA-NBART and LRA-NBART, respectively. The total number of messages is asymptotically identical in all scenarios and is $O(N^2)$. Also the PC, PTC, and SC of TO are identical in both algorithms and are also $O(N^2)$. A comparison of the algorithms regarding BC, PC, and PTC are summarized in Tables I, II, and III. We do not include the comparison of the SC, since the results are as expected. That is, both algorithms store the same amount of information, but since LRA-NBART has greater time complexity, then it also has greater SC.

Scenario	**Conditions**	**ERA-NBART**	**vs**	**LRA-NBART**
Optimistic Scenario	$l_v \gg l_s$	$O(Nf)$	$>$	$O(N)$
	$l_v \simeq l_s$	$O(N^2)$	$=$	$O(N^2)$
Crash-Faults Scenario	$l_v \gg l_s$	$O(Nf)$	$>$	$O(N)$
	$l_v \simeq l_s$	$O(N^2)$	$=$	$O(N^2)$
Arbitrary-Faults Scenario	$l_v \gg l_s$ and $N \geq f^2$	$O(Nf)$	$>$	$O(N)$
	$l_v \gg l_s$ and $N < f^2$	$O(Nf)$	$>$	$O(f^2)$
	$l_v \simeq l_s$	$O(N^2)$	$=$	$O(N^2)$

Table I
Comparison of the bit complexity.

	Scenario	**Conditions**	**ERA-NBART**	**vs**	**LRA-NBART**
Producers	Optimistic Scenario		$O(1)$	$=$	$O(1)$
	Crash-Faults Scenario	$k_v \gg k_s$	$O(1)$	$=$	$O(1)$
		$k_v \simeq k_s$	$O(1)$	$<$	$O(f)$
	Arbitrary-Faults Scenario	$k_v \gg k_s$	$O(1)$	$=$	$O(1)$
		$k_v \simeq k_s$	$O(1)$	$<$	$O(f)$
Consumers	Optimistic Scenario		$O(N)$	$=$	$O(N)$
	Crash-Faults Scenario		$O(N)$	$=$	$O(N)$
	Arbitrary-Faults Scenario		$O(N)$	$=$	$O(N)$

Table II
Comparison of the processing complexity.

This analysis leads to the conclusion that ERA-NBART is more suited for systems where the execution time is more relevant than communication costs, specially when the value is not very large. However, when participants strive to minimize bandwidth costs and the value is large, LRA-NBART presents interesting properties since it always incurs lower communication costs in most scenarios. Moreover, although LRA-NBART has greater costs for storing the value, in some cases, the storage complexity is less critical than the bit complexity, since the memory may be cheaper than fast Internet connections. In addition, the worst case scenario is a pessimistic analysis; in practice the number of Byzantine participants might be significantly lower than f.

	Scenario	Conditions	ERA-NBART	vs	LRA-NBART
Producers	Optimistic Scenario	$l_v \gg l_s$	$O(f)$	$>$	$O(1)$
		$l_v \simeq l_s$	$O(N)$	$=$	$O(N)$
	Crash-Faults Scenario	$l_v \gg l_s$	$O(f)$	$=$	$O(f)$
		$l_v \simeq l_s$	$O(N)$	$=$	$O(N)$
	Arbitrary-Faults Scenario	$l_v \gg l_s$	$O(f)$	$=$	$O(f)$
		$l_v \simeq l_s$	$O(N)$	$=$	$O(N)$
Consumers	Optimistic Scenario	$l_v \gg l_s$	$O(f)$	$>$	$O(1)$
		$l_v \simeq l_s$	$O(N)$	$=$	$O(N)$
	Crash-Faults Scenario	$l_v \gg l_s$	$O(f)$	$>$	$O(1)$
		$l_v \simeq l_s$	$O(N)$	$=$	$O(N)$
	Arbitrary-Faults Scenario	$l_v \gg l_s$	$O(f)$	$>$	$O(1)$
		$l_v \simeq l_s$	$O(N)$	$=$	$O(N)$

Table III
COMPARISON OF THE PACKET TRANSMISSION COMPLEXITY.

Also, it is unlikely for most of the Byzantine participants to follow a purely malicious behaviour. Therefore, in many practical situations the average case is closer to the best case, for which there is a single value transfer per consumer.

VII. SOLUTIONS FOR A MODEL WHERE $N_{\mathcal{P}} \neq N_{\mathcal{C}}$

In this section, we extend the previous solutions for an environment where the number of producers ($N_{\mathcal{P}}$) may be different than the number of consumers ($N_{\mathcal{C}}$). We state that it must be true that $N_{\mathcal{P}} \geq 2f_{\mathcal{P}} + 1$, where $N_{\mathcal{P}}$ is the upper bound on the number of Byzantine producers. Otherwise, consumers cannot determine which of the possible multiple received values is correct. Concerning consumers, the only restriction on $N_{\mathcal{C}}$ is that there must exist at least one non-Byzantine consumer to acknowledge the receipt of the correct value and therefore to certify the behaviour of producers. Hence, $N_{\mathcal{C}} \geq f_{\mathcal{C}} + 1$.

In order to ensure that the ERA-NBART algorithm is correct in this new scenario, it is only necessary to change the definition of *consumerset*$_i$ and *producerset*$_j$ for each producer p_i and consumer c_j, respectively. More precisely, each consumer must be associated by *producerset*$_j$ with exactly $f_{\mathcal{P}}+1$ producers, ensuring that at least a non-Byzantine producer transfers the value to each non-Byzantine consumer. In addition, it is desirable to optimize load balance among all producers, that is:

$$\forall_{p_i,p_k \in \mathcal{P}} |\#\mathit{consumerset}_i - \#\mathit{consumerset}_k| \leq 1 \quad (1)$$

Regarding LRA-NBART, besides ensuring that the requests performed by consumers are distributed to all producers, it is also desirable to distribute the load in each round. For instance, we do not want a producer to be idle in a given round while another producer receives requests from two or more consumers. These new rules can be more precisely defined as follows:

$$\forall_{p_i,p_k \in \mathcal{P}} |\#\mathit{consumerseq}_i - \#\mathit{consumerseq}_k| \leq 1 \quad (2)$$

$$\forall_{p_i,p_k \in \mathcal{P}} \forall_{r \in [0\ldots f]} |\#\mathit{consumerseq}_i[r] - \#\mathit{consumerseq}_k[r]| \leq 1 \quad (3)$$

In the thesis, we provide new definitions for the *producerset* and *consumerset* data structures that fulfil Inequality 1. We also provide new definitions for the *producerseq* and *consumerseq* data structures that, simultaneously, fulfil Inequalities 2 and 3.

New definitions of the predicates *hasProduced* and *hasAcknowledged* must be provided as follows:

- *hasProduced(evidence,* p_i*)* is true if the following condition holds: there are at least $N_{\mathcal{C}} - f_{\mathcal{C}}$ consumers $c_k \in \mathcal{C}$: $\mathit{evidence}[c_k][p_i] = \langle h(v), s_{p_i}(h(v))\rangle$. It is false otherwise.
- *hasAcknowledged(evidence,* c_j*)* is true if the following conditions hold: it exists a set of producers, named *correctset*$_j$, such that $\#\mathit{correctset}_j \geq N_{\mathcal{P}} - f_{\mathcal{P}}$ and for $\forall p_k \in \mathit{correctset}_j$ $\mathit{hasProduced}(\mathit{evidence}, p_k)$ is true and $\mathit{evidence}[c_j][p_k] = \langle h(v), s_{p_k}(h(v))\rangle$. It is false otherwise.

In this new scenario, the proofs of correctness of ERA-NBART and LRA-NBART are identical to the proofs provided for the scenario where $N_{\mathcal{P}} = N_{\mathcal{C}}$. Furthermore, the principles used to prove that ERA-NBART and LRA-NBART provide a Nash-equilibrium also hold in this new scenario.

VIII. CONCLUSIONS

In this work, we have introduced the NBART problem that abstracts the problem of transferring data from a set of producers to a set of consumers under the BAR system model.

We have presented an algorithm named ERA-NBART that solves the NBART problem for $N \geq 2f+1$, where N is the number of producers and consumers. We have shown that our algorithm is a Nash equilibrium, so Rational participants are unable to extract any benefit from deviating from the algorithm. We also presented an alternative algorithm named LRA-NBART that also solves the NBART problem and provides a Nash equilibrium. A comparison between the two algorithms allowed to conclude that ERA-NBART is more appropriate for scenarios where the execution time is more critical than the communication costs. Otherwise, LRA-NBART presents a lower complexity, because in most scenarios it only requires one transfer of the value per consumer.

NBART is a powerful construct to build peer-to-peer systems that support distributed storage and parallel processing based on volunteer processes. One of the main motivations of this work was to build such a system based on a P2P architecture, named BARRAGE, which aimed at supporting distributed computations using the MapReduce model. Another application of NBART could be a data backup system, where volunteers would provide their resources to backup the data of other users, in trade of a payment that would allow them to backup up their own data. In this scenario, NBART could be used to prevent volunteers from storing the data for long periods of time. Considering these two possible applications, the ERA-NBART algorithm would be more appropriate for BARRAGE, since this system is more demanding in terms of execution time. LRA-NBART would be more appropriate for the data backup system, since the main requirement of these systems is data availability instead of performance, and LRA-NBART incurs lower communication costs.

ACKNOWLEDGMENTS

This work was partially supported by the FCT (INESC-ID multi annual funding through the PIDDAC Program fund grant and by the project PTDC/EIA-EIA/102212/2008). Parts of this work have been performed in collaboration with other members of the Distributed Systems Group at INESC-ID, namely, João Leitão, Miguel Correia, and Oksana Denysyuk.

REFERENCES

[1] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[2] D. Anderson, "Boinc: A system for public-resource computing and storage," in *GRID'04*, Pittsburgh, USA, Nov. 2004, pp. 4–10.

[3] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *OSDI'04*, San Francisco, USA, 2004.

[4] S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "BAR fault tolerance for cooperative services," in *SOSP'05*, Brighton, United Kingdom, Oct. 2005, pp. 45–58.

[5] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–47, 1968.

[6] O. Martin and R. Ariel, *A Course in Game Theory*. MIT Press, 1994.

[7] A. Clement, J. Napper, H. Li, J.-P. Martin, L. Alvisi, and M. Dahlin, "Theory of bar games," in *PODC'07*, Portland, USA, Aug. 2007, pp. 358–359.

[8] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, pp. 382–401, Jul. 1982.

[9] D. Dolev and H. Strong, "Authenticated algorithms for Byzantine agreement." *SIAM J. Comput.*, vol. 12, no. 4, pp. 656–666, 1983.

[10] R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience," in *STOC'93*, New York, USA, 1993, pp. 42–51.

[11] L. Lamport, "The part-time parliament," *ACM Trans. on Computer Systems*, vol. 16, no. 2, pp. 133–169, May 1998.

[12] S. Lee and K. Shin, "Interleaved all-to-all reliable broadcast on meshes and hypercubes," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 5, no. 5, pp. 449–458, May 1994.

[13] P. Fraigniaud, "Asymptotically optimal broadcasting and gossiping in faulty hypercube multicomputers," *Computers, IEEE Transactions on*, vol. 41, no. 11, pp. 1410–1419, Nov. 1992.

[14] D. Malkhi and M. Reiter, "Byzantine quorum systems," in *STOC'97*, El Paso, USA, 1997, pp. 569–578.

[15] J.-P. Martin, L. Alvisi, and M. Dahlin, "Minimal Byzantine storage," in *DISC'02*, Toulouse, France, Oct. 2002, pp. 311–325.

[16] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.

[17] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative Byzantine fault tolerance," in *SOSP'07*, Stevenson, USA, Oct. 2007, pp. 45–58.

[18] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "EBAWA: efficient byzantine agreement for wide-area networks," in *HASE'10*, San Jose, USA, Nov. 2010.

[19] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. of ACM*, vol. 35, pp. 288–323, April 1988.

[20] R. Baldoni, J.-M. Helary, M. Raynal, and L. Tanguy, "Consensus in Byzantine asynchronous systems," *J. Discrete Algorithms*, vol. 1, no. 2, pp. 185–210, 2003.

[21] M. Correia, N. F. Neves, L. C. Lung, and P. Verissimo, "Low complexity Byzantine-resilient consensus," *Distributed Computing*, vol. 17, no. 3, pp. 237–249, 2005.

[22] K. Eliaz, "Fault tolerant implementation," *Review of Economic Studies*, vol. 69, no. 3, pp. 589–610, 2002.

[23] T. Moscibroda, S. Schmid, and R. Wattenhofer, "On the topologies formed by selfish peers," in *PODC'06*, Denver, USA, Jul. 2006, pp. 133–142.

[24] I. Abraham, D. Dolev, R. Gonen, and J. Halpern, "Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation," in *PODC'06*, Denver, USA, Jul. 2006, pp. 53–62.

[25] R. Axelrod, *The Evolution of Cooperation*. New York: Basic Books, 1984.

[26] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "BAR gossip," in *OSDI'06*, Seattle, USA, Nov. 2006, pp. 191–204.

[27] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin, "Flightpath: Obedience vs choice in cooperative services," in *OSDI'08*, San Diego, USA, Dec. 2008.

[28] E. L. Wong, J. B. Leners, and L. Alvisi, "It's on me! the benefit of altruism in BAR environment," in *DISC'10*, Cambridge, USA, Sep. 2010, pp. 406–420.