INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

N-Party BAR Transfer

Xavier Vilaça

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Prof. Pedro Manuel M. Antunes de Sousa
Orientador:	Prof. Luís Rodrigues
Vogal	Prof. Alysson Neves Bessani

October 2011

Acknowledgements

First of all, I would like to thank my advisor, Prof. Luís Rodrigues, for the unique opportunity to work with him, and for continuously providing the right motivation and advices. Without them, I would not be capable of successfully overcoming the main obstacles of this work.

I would also like to thank João Leitão for his fruitful suggestions and for the very enlightening discussions we had.

I would like to express my gratitude to Oksana Denysyuk and Miguel Correia for their help. Their theoretical insight were essential to the success of this work.

These acknowledgements should not be concluded without recognizing the importance of my colleagues Nuno Machado, Pedro Louro, João Fernandes and Pedro Ruivo, and other members of the GSD group at INESC-ID, for their useful comments and most precious support.

Finally, I would like to thank my parents and sister for their emotional support.

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).

Lisboa, October 2011

Xavier Vilaça

For my parents and sister,

Francisco, Maria Margarida and

Marta

Resumo

As redes *peer-to-peer* têm ganho ênfase no contexto da execução de projectos científicos, recorrendo a recursos disponibilizados por voluntários. Este tipo de arquitecturas levanta alguns desafios, já que nem todos os nós da rede seguem o comportamento esperado. Na realidade, há nós que seguem um comportamento arbitrário (Bizantino), ou um comportamento egoísta (Racional), tal como especificado no modelo *Byzantine-Altruistic-Rational* (BAR). Para além disso, se as computações se realizarem num modelo como o *MapReduce*, tem de existir algum mecanismo que permita às tarefas comunicarem directamente entre si.

Esta tese introduz o problema *N-party BAR Transfer* (NBART), que consiste na transferência fiável de dados de um conjunto de produtores para um conjunto de consumidores no modelo BAR. Trata-se de uma primitiva relevante para a computação voluntária pois permite a comunicação fiável entre tarefas e evita que os voluntários sejam forçados a armazenar os resultados das computações por longos períodos de tempo, podendo transferir os dados para outros voluntários, após terem armazenado essa informação por um período mínimo de tempo.

São propostos dois algoritmos que resolvem o problema NBART. O primeiro é executado num número constante de rondas, mas incorre em custos superiores de comunicação. Já o segundo reduz os custos de comunicação em detrimento do aumento do tempo de execução. Para ambos os algoritmos, prova-se a sua correcção desde que haja uma maioria de voluntários não Bizantinos e nenhum voluntário seja Racional. Através da Teoria de Jogos, demonstra-se que ambos os algoritmos providenciam equilíbrios de Nash.

Abstract

Peer-to-peer networks have emerged as a relevant architecture for executing scientific computations using resources provided by volunteers. However, this architecture poses several challenges, since volunteers may fail arbitrarily (Byzantine behaviour) or even follow a selfish behaviour (Rational behaviour), as indicated by the Byzantine-Altruistic-Rational (BAR) model. Also, if such distributed computations are to be performed in a computational model such as MapReduce, there must be some mechanism for reliably transferring data among tasks.

This thesis introduces the N-party BAR Transfer (NBART) problem, which is the problem of reliably transferring data from a set of producers to a set of consumers, in the BAR model. This is an important building block for volunteer computing, since it allows tasks to reliably communicate among each other, and volunteers are not obliged to store the data for long periods of time. Instead, each volunteer may transfer the data, after storing it for a minimum amount of time, to another volunteer.

This thesis also proposes two alternative algorithms for solving the NBART problem. One executes in a constant number of rounds and has greater communication complexity, while the other has a lower communication complexity but it has a greater execution time. It is proved that, if no volunteer follows a Rational behaviour, then both algorithms ensure a reliable transfer of data when a majority of producers and a majority of consumers are non-Byzantine. In addition, a theoretical analysis based on Game Theory is performed, proving that both algorithms provide a Nash equilibrium.

Palavras Chave
Keywords

Palavras Chave

Modelo BAR

Armazenamento de Dados

Transferência de Dados

Tolerância a Faltas Bizantinas

Comportamento Racional

Teoria de Jogos

Equilíbrio de Nash

Keywords

BAR Model

Data Storage

Data Transfer

Byzantine Fault Tolerance

Rational Behaviour

Game Theory

Nash-equilibrium

Índice

List of Figures

List of Tables

Acronyms

BAR Byzantine Altruistic Rational

NBART N-party BAR Transfer

P2P Peer-to-Peer

ERA-NBART Eager Risk-Averse NBART

LRA-NBART Lazy Risk-Averse NBART

1 Introduction

This thesis addresses the problem of transferring data in a peer-to-peer network. We study mechanisms for ensuring a reliable transfer of information even if some nodes of the network adopt a rational or arbitrary behaviour.

1.1 Motivation

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 (Anderson, Cobb, Korpela, Lebofsky, & Werthimer 2002) and, more recently, by the Boinc infrastructure that supports several computationally intensive research projects (Anderson 2004). If such computations are performed using MapReduce (Dean & Ghemawat 2004), 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 (Rational). We use the words *processes* or *participants* interchangeably to designate these entities.

The fact that Byzantine participants may adopt an arbitrary behaviour may compromise the safety of the system. Thus, it is imperative to deal with this kind of behaviour. Furthermore, systems not designed to cope with Rational behaviour may fall into the Tragedy of Commons (Hardin 1968): the job is not done because all participants are Rational and aim for profit by not performing (part of) their role.

Previous work that studied the problem of transferring data among sets of processes either focused only on Byzantine or on Rational behaviour (Lamport, Shostak, & Pease 1982; Cohen 2003; Cox & Noble 2003; Castro & Liskov 2002; Malkhi & Reiter 1997). Only recently, solutions have been proposed that take into account both Rational and Byzantine behaviour (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005; Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006; Li, Clement, Marchetti, Kapritsos, Robison, Alvisi, & Dahlin 2008; Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008). The system model used for modelling the behaviour of participants in those solutions has been coined the Byzantine-Altruistic-Rational (BAR) model (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005).

Game Theory (Martin & Ariel 1994) is an interesting approach for modelling Rational behaviour, since it provides a formal view of the interactions among Rational participants, which allow us to reach conclusions regarding their expected behaviour. In this approach, the protocols executed by the processes are modelled as a game, in which each player (i.e., process) follows a strategy to maximize its utility. The concept of Nash equilibrium (Nash 1951) may be applied to prove that no player has any incentives to deviate from the protocol, given that all the remaining players follow the protocol. Therefore, if a protocol provides a Nash equilibrium and it is a dominant strategy (no other Nash equilibrium has a greater utility), then all Rational players should follow it (Mailath 1998).

Therefore, this work focus on creating mechanisms for reliably transferring data among two sets of processes of a peer-to-peer network, where the behaviour of participants is characterized according to the BAR model. Hence, these mechanisms should tolerate Byzantine behaviour and should provide incentives for Rational participants to follow a behaviour that does not compromise the safety of the system. We intend to use the concepts of Game Theory to analyse Rational behaviour.

1.2 Contributions

This work studies the problem of transferring data from a set of N_P producers to a set of N_C consumers, where the behaviour of participants is modelled by the BAR model. We decided to name this problem N-party BAR Transfer (NBART). Regarding Rational behaviour, we distinguish between risk-averse and risk-seeking processes. Risk-averse processes are not willing to deviate from the protocol if that may risk their utility. On the other hand, risk-seeking processes follow the behaviour that maximizes their expected utility, even if that behaviour may risk their utility.

More precisely, the thesis makes the following contributions:

- It introduces the NBART problem, defining its main properties and challenges.
- It proposes two alternative algorithms that solve the NBART problem in a synchronous environment where processes are risk-averse:
  - The first algorithm, named Eager Risk-Averse NBART (ERA-NBART), has low execution time and high bit complexity.
  - The second algorithm, named Lazy Risk-Averse NBART (LRA-NBART), has low bit complexity and high time complexity.

1.3 Results

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 of the proposed solutions.

1.4 Research History

The initial aim of this work was to design, implement, and evaluate a subsystem for data storage in the BAR model, named BARRAGE. The main goal of this system was to support distributed computations over a peer-to-peer network of volunteers, by allowing producers to store intermediate results in processes named storers that are neither producers nor consumers. This way, it would be possible to preserve the data for long periods of time without requiring producers to remain connected until the final consumption of the produced data. The main challenges were how to ensure a reliable transfer of the data from the producers to the storers, and how to preserve the data until its final consumption since storers may not remain connected to the network indefinitely. Eventually, it was realized that a mechanism for reliably transferring data from producers to storers could also be used to transfer the data between different sets of storers and from the final set of storers to the consumers. Therefore, focus was given to solve this particular problem.

This work was performed in the context of the HPCI project: "High-Performance Computing over the Large-Scale Internet" (PTDC/EIA-EIA/102212/2008). During this work, we benefited from the fruitful collaboration with the remaining members of the GSD team working on HPCI, namely João Leitão, Miguel Correia, and Oksana Denysyuk.

Parts of this work were published by Vilaça, Leitão, & Rodrigues (2011a), by Vilaça, Leitão, & Rodrigues (2011b), and by Vilaça, Leitão, Correia, & Rodrigues (2011).

1.5 Structure of the Document

The remaining of this document is organized as follows. Chapter 2 describes previous work related to the problems that were introduced. Chapter 3 provides a definition of the NBART problem. The proposed solutions are presented in Chapter 4. Finally, Chapter 5 concludes the report and provides directions for future work.

2 Related Work

2.1 Introduction

This chapter provides an overview of the previous work related to the transfer of information among processes in different fault models. A special emphasis is given to the BAR model and solutions that use it to portray the behaviour of processes.

First, we introduce in Section 2.2 the different system models used in the literature, which are relevant to understand and compare the described solutions and to motivate the use of the BAR model. In Section 2.3, we survey the traditional work on data transfer among processes. In Section 2.4, we describe the main existing mechanisms for providing incentives to Rational processes to obey the protocols. Then, in Section 2.5, we survey the most important work that deals with Rational and Byzantine behaviour, simultaneously. Finally, in Section 2.6, we describe the most important solutions which are later used as basis for comparison with the work presented in this thesis.

2.2 System Models

A distributed system can be defined as a set of processes that exchange messages over communication channels (Lamport 1978). In this context, there are several different models that capture the behaviour of a distributed system. The most important characteristics to consider in a system model are its *synchrony* and *types of faults* of its components (processes and communication channels).

In this section, we characterize the different timing assumptions that define the level of synchrony of a system and we identify the main types of faults of its components. Then, we describe the possible causes of faults, strategies that may be used to deal with faults, and fault models that are more relevant to this work.

2.2.1 Synchrony

The synchrony of a system model can be modelled using two parameters, that capture the timing assumptions about the behaviour of components. The first parameter is the *process speed interval*, which represents the difference in the processing speed of the fastest and the slowest process of the system. The other parameter is the *communication delay*, which is the latency of communication steps, i.e., the time elapsed between the emission and reception of messages. The level of synchronism of a model ranges from *purely synchronous* if there is a known upper bound on both parameters which always holds, to *purely asynchronous* if none of the parameters are upper bounded. Many intermediate levels of synchrony may be considered. For instance, there may be an unknown upper bound on the parameters which holds forever; or there may be known or unknown upper bounds which eventually hold forever or during a given period of time.

2.2.2 Types of Faults

In a distributed system where processes exchange messages, components may be faulty if they fail to perform some work or non-faulty if they always perform as expected. Eventually, a fault may lead to a failure if its effects become externally visible (Avizienis, Laprie, Randell, & Landwehr 2004).

Several types of faults may occur. We distinguish between *crash faults*, *omission faults*, and *arbitrary faults*:

Crash Faults: A crash occurs when a component stops executing operations it should perform. However, a crash never results in a process performing erroneous operations or in a communication channel producing duplicated or spurious messages. For instance, crash faults may occur when a cable is unplugged, when there is a power loss, or when a software bug or a hardware malfunction cause a computer to disconnect from the network.

Omission Faults: Omission faults occur whenever a component fails to perform a simple operation. More precisely, processes may omit messages or certain execution steps, and communication channels may omit the transmission of a message. For instance, a message may be silently discarded due to a buffer overflow.

Arbitrary Faults: Arbitrary faults are a broader class of faults that may include faults by crash, malicious faults, faults due to Rational behaviour, among others. For instance, malicious processes may deliberately produce wrong information, omit messages, or discard stored data; Rational processes may omit certain messages or execution steps; malicious communication channels may drop or tamper with messages; or honest components may crash due to arbitrary reasons.

2.2.3 Causes of Faults

Faults may occur due to many possible reasons. To better identify the appropriate strategy to deal with faults, it is useful to classify the causes of faults into three main classes: *natural causes*, *malicious behaviour*, and *rational behaviour*.

2.2.3.1 Natural Causes

A common source of faults is related to events not intentionally caused by humans. For instance, power loss or a user that inadvertently disconnects its computer from a power source may cause a process to stop performing operations; a disconnected or damaged cable may prevent communication channels from delivering messages; a software or hardware bug may cause a process to produce erroneous data; among others.

2.2.3.2 Malicious Behaviour

A malicious fault always occur due to the intentional actions of an adversary: a malicious user such as an hacker, a component infected by malicious code (worms, viruses, trojan horses, bots,...); a network of controlled components (botnets); etc. Malicious components may act with many distinct goals: destroy or jeopardize the services provided by a particular company; revealing secrets hold by an organization; increase their account balance or access other accounts; among others. Their actions may be arbitrary, therefore they may cause any type of faults. For instance, malicious components may selectively switch between correct and malicious behaviour.

2.2.3.3 Rational Behaviour

A process follows a Rational behaviour if it aims at maximizing a given utility function that is known. That is, if it aims at maximizing the benefits obtained from the system, while minimizing the costs incurred for performing operations, such as sending messages, performing computations, or storing data for long periods of time. A fault due to Rational behaviour occurs whenever the behaviour that maximizes the utility of a component is not compatible with the expected behaviour.

We can identify the following main problems derived from Rational behaviour:

Free-Riding: The problem of free-riding in the context of P2P has been discussed by Hughes, Coulson, & Walkerdine (2005). Free-riders will exploit the altruism of other participants to take advantage of the system without reciprocating in a reasonable manner, i.e., providing at least as much resources as they consume. Experience with several practical systems (Liang, Kumar, & Ross 2005) has shown that this problem can ultimately prevent the system from working. This leads to a situation called *Tragedy of the Commons* (Hardin 1968), where every process is expecting others to provide services, but that never happens and the system stops working.

Alternative Behaviour: Most protocols provide different behavioural options for each step of an interaction. It is possible for a Rational process to take advantage of this and choose a particular option that minimizes its cost or allows it to gain unfair advantages, instead of making the adequate choice. For instance, in a partner selection mechanism that is based on a random number generator, a Rational component always generate the same number in order to always pick the best peer (Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006).

Collusion: This problem occurs both with malicious and Rational processes. Collusion happens when several processes coordinate their actions to defeat the mechanisms in place that aim at preventing or detecting incorrect behaviour (Hayrapetyan, Tardos, & Wexler 2006).

2.2.4 Strategies for Dealing with Faults

Several distinct techniques may be used to address the problem of faults in systems. In this thesis, we focus on *fault prevention* and *fault tolerance* techniques (Avizienis, Laprie, Randell, & Landwehr 2004).

Fault Prevention: Techniques for preventing faults increase the dependability of the system by reducing the likelihood of fault occurrence. Good programming practices can prevent software errors and avoid vulnerabilities that may lead to security attacks. If security is taken into account, it may be possible to prevent malicious processes from disclosing or disrupting information, and from forging identities. Incentive mechanisms can provide enough reasons for rational participants to follow the specified protocols, if this is the alternative that maximizes their profit.

Fault Tolerance: Fault tolerance consists in adding mechanisms that allow the system to continue to provide the intended service despite the occurrence of faults. Fault tolerance may be achieved by detecting the faults and applying corrective measures, or simply by masking the fault using redundancy. In the context of data transfer, it aims at avoiding data loss or corruption.

In this thesis, we are focused on systems characterized by the BAR model. Hence, faults may occur as a result of any possible cause. Faults due to Rational behaviour may be prevented by providing incentives for Rational processes to not misbehave. However, faults caused by natural events or malicious behaviour cannot be avoided, thus fault tolerance mechanisms must be applied.

2.2.5 Fault Models

As we have seen, components may fail due to many reasons and exhibit different behaviours when faults occur. Fault models are abstractions that describe the types of faults that may occur in real systems. For the purpose of this work, we assume that communication channels do not fail. Hence, we will focus on fault models that capture the possible faults of processes. More precisely, we will describe the main fault models of the literature that identify the possible faults that may occur by classifying processes according to their expected behaviour. These are the *crash-stop*, *Byzantine*, and BAR fault models.

2.2.5.1 Crash-stop Fault model

In the crash-stop fault model, a faulty process follows the specified behaviour until it crashes. Therefore, a faulty process never executes erroneous steps or produces incorrect results. A process is said to be correct if it never crashes.

2.2.5.2 Byzantine Fault Model

The Byzantine fault model classifies processes as either correct or Byzantine[1]. Byzantine processes follow an arbitrary behaviour. For instance, they may crash, disrupt data integrity, among others. A particular case of this fault model is the *authenticated Byzantine fault model*, where messages are authenticated such that Byzantine processes cannot forge messages.

It is important to notice that the Byzantine Fault Model classifies any process that deviates from the expected behaviour as Byzantine. If Rational processes deviate from the protocol, then they are considered to be Byzantine in this model.

2.2.5.3 BAR Fault Model

BAR stands for Byzantine, Altruist, and Rational (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005). The model combines the previous models, in what concerns the behaviour of processes. Byzantine are incorrect processes just like we have described in the previous section.

Rational processes are characterized by a well defined utility function that represents their expected profit (and that Rational processes strive to maximize). This utility function captures what these processes seek to achieve so we can model our protocols to conciliate that function with the common good. Notice that Byzantine processes may also be Rational, in the sense that they may also adopt a defined strategy. The only difference is that their utility function is unknown.

[1]The word Byzantine was firstly used by Lamport, Shostak, & Pease (1982) in the definition of the Byzantine generals problem.

To model a Rational behaviour, it is necessary to consider what are the *benefits* from using the system and what are the *costs*. Then, it should be ensured that the utility of deviating from the protocols is at least the same as following the expected behaviour.

Altruist processes contribute to the system in an obedient manner. Thus, they follow the protocol with or without incentives. The existence of this last type of processes should not be mandatory. Some problems are only solved if assumptions about the fraction of Altruist processes (Wong, Leners, & Alvisi 2010). Though, a BAR tolerant system should provide the desired service, even when all processes are either Byzantine or Rational.

2.3 Data Transfer without Rational Behaviour

In this section, we introduce the main work related to the transfer of information among processes. The most relevant work in this area does not deal with Rational behaviour, thus we focus on work for fault models that do not consider Rational behaviour. We briefly survey the traditional problems of *Reliable Broadcast* and *Consensus*, which are relevant for defining the NBART problem, because the aim of these problems is to ensure that a set of processes delivers the same value broadcast by one or more processes. Solutions to these problems can be used by producers of NBART to broadcast the value to all consumers. Therefore, it is relevant to describe these problems in order to establish a comparison with NBART. Then, we describe the protocols for implementing replicated registers with different concurrency semantics, based on *Byzantine Quorums* and the *Replicated State Machine* approach. This is a recently explored aspect of Byzantine fault tolerant protocols that addresses the problem of transferring and replicating data from clients to servers. Therefore, it is relevant to compare this class of protocols with the possible solutions to the NBART problem.

2.3.1 Reliable Broadcast

Reliable broadcast protocols (Wensley, Lamport, Goldberg, Green, Levitt, Melliar-Smith, Shostak, & Weinstock 1978; Hadzilacos & Toueg 1994; Guerraoui & Rodrigues 2006) ensure that, even in the presence of faults, a set of processes reaches an agreement regarding a message m they deliver, that was broadcast by some process named sender (sender(m)). The sender may or may not be part of the set of processes that receive the broadcast value (dest(m)).

In the following, we address some variants of Reliable Broadcast that are relevant for this work, namely *Regular Reliable Broadcast*, which is the main variant of Reliable Broadcast, and *Probabilistic Reliable Broadcast*, for which solutions have been devised for the BAR model. Other Reliable Broadcast problems, such as *Uniform reliable Broadcast*, *Causal Order Broadcast*, and *FIFO Order Broadcast*, are not discussed since we only aim at ensuring a reliable transfer of data among non-Byzantine processes, and we do not implement any order when delivering the data.

2.3.1.1 (Regular) Reliable Broadcast

Regular Reliable Broadcast ensures that processes agree on the broadcast value even if the sender fails (Wensley, Lamport, Goldberg, Green, Levitt, Melliar-Smith, Shostak, & Weinstock 1978; Kaashoek, Tanenbaum, & Hummel 1989; Chandra & Toueg 1990).

The problem can be defined by four properties **RB1-RB3** for faults by crash or omission (Hadzilacos & Toueg 1994):

RB1 *Validity*: If a correct process broadcasts a message m, then it eventually delivers m.

RB2 *Agreement*: If a correct process delivers a message m, then all correct processes eventually deliver m.

RB3 *Integrity*: For any message m, every correct process delivers m at most once, and only if m was previously broadcast by *sender(m).*

In an environment with arbitrary faults, the integrity property must be redefined, for instance, as follows (Hadzilacos & Toueg 1994):

RB3 *Integrity*:: For any message m, every correct process delivers m at most once, and if sender(m) is correct then m was previously broadcast by sender(m).

2.3.1.2 Probabilistic Reliable Broadcast

The previous definition is deterministic in the sense that it is guaranteed that all processes deliver the same value if at least the sender is correct. However, in large scale networks, it may be necessary to provide greater scalability at the expense of only providing probabilistic guarantees. Instead of broadcasting messages to all processes of the network, processes send messages only to a subset of the remaining processes, which is determined by the topology of the network. For instance, one approach consist on epidemic dissemination, also known as *gossip* (Golding & Long 1992; Birman, Hayden, Ozkasap, Xiao, Budiu, & Minsky 1999; Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006), that uses randomized dissemination, i.e., each process sends a message to a subset of processes randomly chosen. More organized topologies, such as the hypercube topology (Lee & Shin 1994; Johnsson & Ho 1989; Fraigniaud 1992) or other types of hierarchical topologies, may also be used to disseminate information.

This problem can be defined by the following properties for faults by crash and omission (Guerraoui & Rodrigues 2006):

PB1 *Probabilistic Validity*: There is a given probability such that for any two correct processes p_i and p_j, every message broadcast by p_i is eventually delivered by p_j with this probability.

PB2 *No duplication*: No message is delivered more than once.

PB3 *No creation*: If a message m is delivered by some correct process p_j, then m was previously broadcast by some process p_i.

As in Reliable Broadcast, if arbitrary faults may occur, then the *No creation* property must be redefined, for instance, as follows, considering that p_i is the sender:

PB3 *No creation*: If a message m is delivered by some correct process p_j and p_i is correct, then m was previously broadcast by p_i.

2.3.2 Consensus and Related Problems

In Consensus and related problems, a set of processes must reach an agreement about a given value after one or more processes have proposed some value, even in the presence of faults (Pease, Shostak, & Lamport 1980; Lamport, Shostak, & Pease 1982). In this section, we describe *Consensus* and some of the directly related problems, namely *Interactive Consistency*, and *Byzantine Agreement*, which can be directly applied to solve NBART. For the sake of completeness, we also describe the problem of *Total Order Broadcast*, which has been proven to be equivalent to Consensus, and *Terminating Reliable Broadcast*, for which solutions have been devised for the BAR model. Then, we discuss some of the major theoretical results in Consensus, and we present an alternative definition of Consensus included in *Paxos* algorithms, which is more similar to NBART with its three roles of processes.

2.3.2.1 Consensus

The problem of Consensus (Pease, Shostak, & Lamport 1980) can be specified in terms of two primitives: *propose* and *decide*. Each process has an initial value that it proposes to agreement, through the primitive *propose*. All processes must eventually reach an agreement and decide the same value, by invoking *decide*.

The following conditions must hold in order to reach a Consensus, for faults by omission and crash only (Hadzilacos & Toueg 1994):

- *Termination*: Every correct process eventually decides some value.
- *Agreement*: If a correct process decides v, then all correct processes eventually decide v.
- *Integrity*: If a correct process decides v, then v was previously proposed by some process.

If processes fail arbitrarily, then the meaning of a process (correct or Byzantine) proposing a value in the integrity property is ambiguous. To overcome this problem, this property may be redefined as follows (Hadzilacos & Toueg 1994):

- *Integrity*: If all processes are correct and a process decides v, then v was previously proposed by some process.

2.3.2.2 Interactive Consistency

The problem of interactive consistency (Pease, Shostak, & Lamport 1980; Thambidurai & keun Park 1988) is a variant of the Consensus problem, in which each process also proposes a value. However, the aim is to compute a vector v, such that $v[p_i]$ contains the value proposed by the process p_i, or a null value, otherwise.

The following properties must hold for each correct process p_i, as initially proposed by Pease, Shostak, & Lamport (1980), for arbitrary faults:

- The non-faulty processes compute exactly the same vector.
- The element of this vector corresponding to a given non-faulty process is the value proposed by that process.

2.3.2.3 Byzantine Generals

The Byzantine Generals problem (Lamport, Shostak, & Pease 1982; Bracha & Toueg 1985; Schneider 1984; Dolev & Strong 1983), is another variant of Consensus in which only one process proposes a value. The problem was originally defined (Lamport, Shostak, & Pease 1982) as a set of Byzantine generals that seek to reach an agreement on a plan of action. For this, each general acts as the commanding general of a Byzantine agreement instance by issuing an order to the other generals. Then, the remaining generals (lieutenants) must reach an agreement on the order sent by the commanding general. The following properties must hold (Lamport, Shostak, & Pease 1982), for arbitrary faults:

- All loyal lieutenants obey the same order.
- If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

2.3.2.4 Total Order Broadcast

The *total order broadcast* (TO-RB), also known as Atomic Broadcast (Kaashoek & Tanenbaum 1996; Chang & Maxemchuk 1984; Lamport 1978; Schneider 1990), is a variant of Consensus that extends the problem of Reliable Broadcast (described in Section 2.3.1) to ensure that all correct processes deliver a sequence of broadcast messages by the same order.

A possible definition of TO-RB is given by the following property (Hadzilacos & Toueg 1994), in addition to the RB1-RB4 properties of Reliable Broadcast:

- *Total Order*: If correct processes p_i and p_j both deliver messages m and m', then p_i delivers m before m' if and only if p_j delivers m before m'.

Notice that this property is already appropriate for an environment where arbitrary faults occur.

2.3.2.5 Terminating Reliable Broadcast

The *Terminating Reliable Broadcast* (TRB) (Hadzilacos & Toueg 1994) is an extension of the Reliable Broadcast problem, where there is a-priori knowledge that a given process is supposed to broadcast a single message. Processes must either agree on the value broadcast by the sender or must agree on a special message F_s which indicates that the sender is faulty and therefore that no process will ever deliver a message broadcast by the sender. This problem is identical to the Byzantine Generals problem, aside from the fact that TRB was originally defined for an environment where processes only fail by crash or omission.

The TRB properties are defined as follows (Hadzilacos & Toueg 1994), where s is the sender:

- *Termination*: Every correct process eventually delivers exactly one message.
- *Validity*: If s is correct and broadcasts a message m, then it eventually delivers m.
- *Agreement*: If a correct process delivers a message m, then all correct processes eventually deliver m.
- *Integrity*: If a correct process delivers a message m then $\mathit{sender}(m) = s$. Furthermore, if $m \neq F_s$ then m was previously broadcast by s.

In an environment with Byzantine processes, the integrity property becomes ambiguous. That ambiguity may be removed by simplifying this property (Hadzilacos & Toueg 1994):

- *Integrity* If a correct process delivers a message m, then $\mathit{sender}(m) = s$.

2.3.2.6 Theoretical Results

The most prominent theoretical result regarding Consensus has been known as the FLP impossibility result (Fischer, Lynch, & Paterson 1985), which states that Consensus cannot be solved deterministically in a fully asynchronous environment, even if only a single process may fail by crash. This impossibility result can be circumvented using several distinct techniques. For instance, assuming a partially synchronous environment (Dwork, Lynch, & Stockmeyer 1988); using a privileged distributed component, generically designated by wormholes (Correia, Neves, Lung, & Veríssimo 2005); using fault detectors (Chandra, Hadzilacos, & Toueg 1996;

Baldoni, Helary, Raynal, & Tanguy 2003); or using randomized techniques that guarantee that an agreement is reached with high probability (Bracha & Toueg 1985).

Important theoretical results were also obtained regarding the relation among the Consensus variants. More precisely, it has been shown that solving Consensus in an environment where at least a process may fail is equivalent to solving Total Order Broadcast (Dolev, Dwork, & Stockmeyer 1987; Chandra, Hadzilacos, & Toueg 1996), Interactive Consistency (Pease, Shostak, & Lamport 1980), and Byzantine Agreement (Fischer 1983). The problem of Terminating Reliable Broadcast is only equivalent to Consensus in a synchronous environment. In an asynchronous environment, TRB can be used to solve Consensus, but the converse is not true (Hadzilacos & Toueg 1994).

Another important results are related to the minimal execution time, complexity, and number of processes. For instance, Dolev & Strong (1983) proved that Consensus can only be solved in $f+1$ phases (time is divided in phases, which consist on single steps of information exchange) with a message complexity of $O(nf)$, where f is the maximum number of processes that may fail. Lamport, Shostak, & Pease (1982) have shown that it is only possible to solve the Byzantine generals problem with at least $3f + 1$ generals, for f Byzantine generals, if messages are not signed (the sender can forge messages). It is also shown that, with signed messages, it is possible to lower this bound to $f + 2$ generals. These bounds are only applicable if the system is synchronous. For asynchronous systems, the work of Bracha & Toueg (1985) showed that the problem can be solved with $3f + 1$ generals, whether messages are signed or not, assuming a probabilistic behaviour regarding the message system. Hence, the solution is not deterministic, but it reaches an agreement with an expected finite number of steps.

Other theoretical results have been obtained for different variants of the Consensus and for different system models. However, since they are not relevant for the purpose of this thesis, we will not discuss them.

2.3.2.7 Consensus According to Paxos

In the Paxos algorithms (Lamport 1998; Gafni & Lamport 2003; Lamport & Massa 2004; Lamport 2006a; Martin & Alvisi 2006; Chandra, Griesemer, & Redstone 2007; Lamport 2011), the Consensus problem is redefined in a fault model where processes only fail by crash or omission. The new definition encompasses the existence of three different roles for processes (Lamport 2006b):

Proposers: A proposer can propose values.

Acceptors: The acceptors cooperate in some way to choose a single proposed value.

Learners: A learner can learn what value has been chosen.

The following main properties must be fulfilled (Lamport 2006b), for a Consensus that tolerates f faults by crash:

- *Nontriviality*: Only a proposed value may be learned.
- *Safety*: At most one value can be learned.
- *Progress*: If a proposer p, a learner l, and a set of $N - f$ acceptors are non-faulty and can communicate with one another, and if p proposes a value, then l will eventually learn a value.

2.3.3 Byzantine Registers

The previous problems of Reliable Broadcast and Consensus ensure that processes reach an agreement regarding the value proposed by one or more processes. In this Section, we describe some of the solutions that approach the problem of implementing replicated registers with different concurrency semantics. The problem is usually stated as a set of clients that manipulates a state variable (register) that is replicated over a set of servers. Clients can perform read and write operations on the register. The main goal is to ensure the availability and consistency of the value of the register according to different concurrency semantics (Lamport 1978), even if processes may fail arbitrarily. Solutions to this problem may ensure a reliable transfer of data from clients to servers, therefore it is relevant to compare them with the NBART problem.

Some of the main characteristics that distinguish existing solutions are its tolerance to faulty clients as all of the existing solutions support the existence of a fraction of Byzantine servers, and the existence of single versus multiple writers. Normally, solutions tolerate an arbitrary number of readers. Existing solutions can be classified into three main approaches: *Byzantine Quorums*, *State Machine Replication*, and *Hybrid* approaches.

2.3.3.1 Byzantine Quorums

A possible way of storing data and tolerating a certain number of Byzantine failures is to use the notion of quorums (Malkhi & Reiter 1997; Martin, Alvisi, & Dahlin 2002; Alvisi, Malkhi, Pierce, Reiter, & Wright 2000; Malkhi, Reiter, & Wool 1997). A quorum is a subset of servers with which the clients communicate directly. All read and write operations are applied to a given quorum, guaranteeing that any two operations intersect in a sufficient number of non-Byzantine servers, such that the outcome of each operation respects the consistency semantics that the protocol aims to preserve.

2.3.3.2 State Machine Replication

In the State Machine Replicated approach (Castro & Liskov 2002; Clement, Kapritsos, Lee, Wang, Alvisi, Dahlin, & Riche 2009; Schneider 1990; Veronese, Correia, Bessani, & Lung 2010; Castro, Rodrigues, & Liskov 2003; Distler & Kapitza 2011; Lamport 1998), servers communicate among each other to agree on the order of the operations issued by clients and their outcome. For write operations, servers must agree on the order of the operation in order to preserve the consistency of the stored data. For read operations, servers decide which value should be returned, such that clients maintain a consistent view of the register across multiple read operations.

The work related to the Paxos algorithms (Lamport 1998; Lamport & Massa 2004; Lamport 2006a) generalized this approach as a sequence of commands that must be chosen by all servers. For that purpose, proposers propose the $i-th$ command of the sequence using one of the Paxos algorithms. This allows all the acceptors to reach an agreement about which should be the $i-th$ command to be executed. Clients are learners in the sense that they may learn which commands were executed and in what order.

2.3.3.3 Hybrid Approaches

As the name implies, these approaches combine the two previous schemes (Cowling, Myers, Liskov, Rodrigues, & Shrira 2006; Bessani, Correia, & Sousa 2010). The situations when either a quorum-based or a replicated state machine approach are used vary according the application semantics. For instance, in the work of Cowling, Myers, Liskov, Rodrigues, & Shikra (2006), a quorum-based approach is used to obtain a majority of confirmations for each operation. In the presence of contention between writers, a replica-based approach is used to determine the final order of the conflicting operations. Other solutions, such as the one proposed by Bessani, Correia, & Sousa (2010), use quorums for efficiently executing read and write operations, while the replica-based approach is used only when strictly necessary, to implement the more expensive *read-modify-write* operation.

2.4 Previous Work that Considers Rational Behaviour

This section presents two classes of previous work related to fault models that consider Rational behaviour. More precisely, we describe the main existing work that specifically deals with the problem of free-riding and fair exchange of information, namely *incentive-based protocols*, *fair exchange* and *non-repudiation protocols.*

2.4.1 Incentive-Based Protocols

Incentive-based protocols implement mechanisms that reward (or punish) processes for offering (or failing to provide) the intended service. The current state of art includes systems based on the following mechanisms: *direct reciprocity*, and *reputation.*

2.4.1.1 Direct Reciprocity

In direct reciprocity-based protocols, instead of being paid for a service, each process always provides a service in exchange for the service it receives. When storing data, participants have to store locally the same amount as they want to store on other processes. This is frequently used in backup systems (Cox, Murray, & Noble 2002; Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005; Elnikety, Lillibridge, Burrows, & Zwaenepoel 2002). Other forms of direct reciprocity are based on tit-for-tat (Axelrod 1984) relationships, where processes can retaliate if the peer with whom they interact fail to provide some service. For instance, in Bittorrent (Cohen 2003), processes can deny service to free-riders. Another example is the ∞-tit-for-tat (Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008), where Rational processes can force others to send penance messages for each message omitted in past interactions, in order to prevent processes from avoiding costs by misbehaving and, therefore, increasing their utility.

2.4.1.2 Reputation

In these protocols, the system keeps track of the quality of services that a process has provided to the system using a reputation value. When a process provides service, its reputation increases. If the process fails to provide the expected service, its reputation decreases. Therefore, the strategy of a rational process is to maximize its reputation. Several solutions fall into this class (Cohen 2003; Gupta & Somani 2004; Walsh & Sirer 2006; Keidar, Melamed, & Orda 2006). Reputation protocols can be divided in three classes:

Local Reputation Systems: Each process keeps the reputation information of other processes and updates it at each interaction (Cohen 2003).

Voting Reputation Systems: This scheme makes use of the local reputation information from different processes in order to get a more accurate estimate of the real contribution of a given target process to the system. Whenever there is the need to determine the reputation of a process, a voting mechanism is executed (Walsh & Sirer 2006).

Transaction-based Systems: Processes use proofs that a certain service was provided, also known as certificates, to assert their reputation (Gupta & Somani 2004).

2.4.2 Non-Repudiation and Fair-Exchange

Non-repudiation (Kremer, Markowitch, & Zhou 2002; Onieva, Zhou, & Lopez 2008) is the problem of ensuring that when two or more processes engage in a transfer, either no process obtains any useful information, or no process is able to repudiate the transfer. For instance, if a process i has some data that is useful to j, then after a non-repudiation transfer, either j obtains the data and a proof that it was sent by i while i also obtains a proof that it has sent the data, or no process obtains any useful information. *Fair-exchange* is similar to non-repudiation, aside from the fact that in this type of exchanges, all processes possess some data useful to other processes. Therefore, the aim is to ensure that either all processes obtain the desired data, or no process is able to retrieve any useful information.

Fair-exchange and non-repudiation solutions can be classified in two major branches regarding the number of processes that engage in an exchange: *two-party* and *multi-party*.

2.4.2.1 Two-Party Non-Repudiation and Fair-Exchange

In this problem, a process P1 possesses some information of interest to P2. Processes are considered to be honest or faulty. P2 is honest if it is interested in obtaining the information P1 owns, while P1 is considered to be honest if it is interested in obtaining a proof that it sent the information to P1. The following properties must hold to ensure non-repudiation (Kremer, Markowitch, & Zhou 2002):

- *Non-repudiation of receipt*: A non-repudiation protocol provides non-repudiation of receipt, if and only if it generates a non-repudiation of receipt evidence, destined to P1, that can be presented to an adjudicator, who can unambiguously decide whether P2 received a given message or not.
- *Non-repudiation of origin*: A non-repudiation protocol provides non-repudiation of origin, if and only if it generates a non-repudiation of origin evidence, destined to P2, that can be presented to an adjudicator, who can unambiguously decide whether P1 is the author of a given message or not.
- *Strong Fairness*: A non-repudiation protocol provides strong fairness if and only if at the end of a protocol execution either P1 got the non-repudiation of receipt evidence for the message m, and P2 got the corresponding message m as well as the non-repudiation of origin evidence for this message, or none of them got any valuable information.

The property of fairness can be weakened to only provide probabilistic guarantees, resulting in the following property:

- *Probabilistic Fairness*: A non-repudiation protocol is ϵ-fair if and only if the probability that at the end of a protocol execution either P1 got the non-repudiation of receipt evidence for the message m, and P2 got the corresponding message m as well as the non-repudiation of origin evidence for this message, or none of them got any valuable information, is $\geq 1-\epsilon$.

In the fair-exchange problem, both processes P1 and P2 possess information of interest to the other. Unlike non-repudiation, it is only necessary to ensure fairness, such that either processes P1 and P2 obtain the information owned by the other or none of them is capable of obtaining any valuable information.

The notion of timeliness can be introduced for both problems, which results in the following property (Kremer, Markowitch, & Zhou 2002):

- *Timeliness*: A protocol provides timeliness if and only if all honest parties always have the ability to reach, in a finite amount of time, a point in the protocol where they can stop the protocol while preserving fairness.

In the presence of faulty processes, timeliness can only be ensured in a synchronous environment, as the non-faulty process is unable to distinguish a failure from a message delay in an asynchronous model.

The most important aspect regarding two-party fair exchange and non-repudiation solutions is the involvement of a *trusted third party* (TTP). Some applications may require that the non-repudiation evidences gathered by processes be independent of the intervention of a TTP. To ensure this, the property of *true fairness* must be introduced:

- *True Fairness*: A non-repudiation protocol provides true fairness if and only if it provides strong fairness and, if the exchange is successful, the non-repudiation evidences produced during the protocol are independent of how the protocol is executed.

Solutions that not require the existence of a TTP were the first to be proposed (Han 1996; Markowitch & Roggeman 1999). On the remaining solutions, the TTP can be classified as *inline*, *online*, and *transparent*:

Inline TTP: The TTP is involved in the transmission transmission of each message (Bahreman & Tygar 1994; Coffey & Saidha 1996).

Online TTP: The TTP is involved in each exchange, but not in each transmission of messages (Zhang & Shi 1996; Rabin 1983).

Offline TTP: The TTP is only involved if some of the processes misbehave or a failure occurs (Kremer & Markowitch 2000b; Zhou, Deng, & Bao 1999).

Transparent TTP: The non-repudiation evidences gathered by processes P1 and P2 in situations when the TTP has to intervene and it has not are indistinguishable (Markowitch & Saeednia 2001).

2.4.2.2 Multi-Party Non-Repudiation and Fair-exchange

The multi-party non-repudiation and fair-exchange problems (Onieva, Zhou, & Lopez 2008) can be defined as a set of $n > 2$ processes trying to transfer information in a fair manner. Several particular problems can be derived from this generic definition. For non-repudiation protocols, this can be generalized as the *m-to-k* non-repudiation problem, i.e., m out of n processes possess some information of interest to k processes. Each of those m processes may send the same information to $k' \leq k$ processes, or a different value for each of the receiving process. Each of the k processes must provide an evidence of receipt to the m processes and must obtain an evidence of origin. In the end, either all m processes hold an evidence of receipt and all k processes possess the information and an evidence of origin, or no process possesses any valuable information. Particular cases of interest are the *many-to-one* (Asokan, Schunter, & Waidner 1997), *many-to-many* (Asokan, Baum-waidner, Schunter, & Waidner 1998), and one-to-many (Markowitch & Kremer 2000).

As in two-party non-repudiation and fair-exchange, these problems can be characterized by *fairness* and *liveness* properties. Furthermore, given that different solutions may require individual transfers of information, then the property of *confidentiality* must be preserved in order to guarantee that only the receiver to whom a given message m is intended to is able to disclose m.

Several solutions have been proposed that make use of different topologies or that require the participation of a TTP. Regarding topologies, the most prominent are the ring topology (Khill, Kim, Han, & Ryou 2001) and the matrix topology (Asokan, Baum-waidner, Schunter, & Waidner 1998). As in two-party transfers, the TTP can be classified as *inline* (Bao, Deng, Nguyen, &

Varadharajan 1999), *online* (Kremer & Markowitch 2000a), *offline* (Markowitch & Kremer 2000), and *transparent* (Garay & MacKenzie 1999).

2.5 Previous Work that Considers Byzantine and Rational Behaviour

In this section, we describe the main work that tolerates Rational and Byzantine behaviour, namely solutions devised for the BAR model and work that also dealt with Byzantine and Rational behaviour but without using the BAR model.

2.5.1 Previous Work in the BAR Model

Protocols devised for the BAR model assume that rational processes strive to minimize costs, such as bandwidth, storage space, communication, and amount of computation. However, it is also assumed that these participants benefit from the services provided by the protocols. Hence, the expected behaviour of each rational process (strategy) consists on maximizing the benefits and minimizing the costs. With regard to protocols, this may imply sending the shortest messages on each step or avoid sending any message at all, sparing as much storage space as possible, and minimizing the amount of computations performed.

Existing work that uses the BAR model focus on the problems of implementing a replicated state machine and data backup (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005), agreement among processes (Clement, Napper, Li, Martin, Alvisi, & Dahlin 2007; Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008), data dissemination (Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006; Li, Clement, Marchetti, Kapritsos, Robison, Alvisi, & Dahlin 2008), and studying the influence of Altruism in fair exchange (Wong, Leners, & Alvisi 2010). Tools for model checking solutions devised for the BAR model were also proposed for an environment where processes interact repeatedly (Mari, Melatti, Salvo, Alvisi, Clement, & Li 2008; Mari, Melatti, Salvo, Tronci, Alvisi, Clement, & Li 2009).

Existing solutions strive to guarantee that, for each step of the protocol, the best strategy for a rational process is to perform as expected. For this purpose, the concept of (exact) *Nash equilibrium* from Game Theory (Nash 1951) is applied. In a Nash equilibrium, a rational participant does not benefit more from changing its strategy. Most of BAR implementations (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005; Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006; Mokhtar, Pace, & Quéma 2010; Clement, Napper, Li, Martin, Alvisi, & Dahlin 2007; Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008; Mari, Melatti, Salvo, Alvisi, Clement, & Li 2008; Mari, Melatti, Salvo, Tronci, Alvisi, Clement, & Li 2009) aim at guaranteeing that each step of the protocol is an exact Nash equilibrium, and provide informal proofs that this property holds.

A more flexible approach is taken by Li, Clement, Marchetti, Kapritsos, Robinson, Alvisi, & Dahlin (2008), using the concept of (approximate) ϵ-Nash equilibrium. Here, it is possible that alternative strategies provide a greater benefit. The ϵ consists on an upper bound to the factor by which the expected benefit increases by changing strategy. In an ϵ-Nash equilibrium, rational processes only change their strategy if the expected raise of the benefits is greater than ϵ.

All the existing solutions in the literature adopt several measures for ensuring an exact or approximate equilibrium, as described below:

- Messages have a strict format and are digitally signed. A signed message that is wrongly formatted, or sent in a wrong protocol step, is a *proof of misbehaviour* (POM).
- The number of available choices is minimized, in order to reduce the opportunities for the rational process to select a behaviour that is not beneficial to the system. For instance, non-deterministic choices should be avoided, as the rational participants may select the most favourable choice instead of using a random variable. This can be achieved using verifiable pseudo non-determinism, that achieves good probabilistic distribution while preventing processes from manipulating it.
- Balanced costs. A common strategy is to design the protocol such that each message has the same length, so that there are no incentives to send a different message than the expected one.

Existing BAR protocols are based on certain assumptions. Firstly, rational processes are considered to be risk-averse. This means that they always follow the safest choice and that they are conservative about the impact of Byzantine participants in the network. Hence, a rational process never colludes with a Byzantine participant, just to obtain a greater benefit, if this puts in risk its participation in the system. Secondly, it is assumed that, if the protocol provides a Nash equilibrium, all the rational processes follow it. In other words, when presented with several alternative strategies with equivalent utilities, a Rational participant always adopt the strategy that complies with the protocol. Another common assumption for asynchronous systems is that Rational processes never delay sending a message if that does not provide any increase in its utility, in what is called the *promptness principle* (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005).

2.5.2 Other Works

Prior to the definition of the BAR model, there was already some work on that dealt with Byzantine and Rational behaviour (Eliaz 2002). The concepts presented in this work were later extended to other contexts (Moscibroda, Schmid, & Wattenhofer 2006; Abraham, Dolev, Gonen, & Halpern 2006).

Eliaz (2002) used Game Theory (Martin & Ariel 1994) to model games with Byzantine and Rational players. The authors formalized the concept of k-Nash equilibrium, which is a situation where no Rational process has incentives to deviate from the expected behaviour, given the existence of up to k Byzantine processes. This concept was applied to the problems related to auctions. The work by Moscibroda, Schmid, & Wattenhofer (2006) applied the concept of Byzantine Nash equilibrium to the virus inoculation game.

Abraham, Dolev, Gonen, & Halpern (2006) extended the previous work by introducing the concept of (k, t)-robustness. A protocol is (k, t)-robust if no Rational process has incentives to deviate from the expected behaviour when up to t processes are Byzantine, even if it colludes with $k - 1$ processes. This concept was applied to the secret sharing problem, where it was assumed that processes did not incur communication costs.

2.6 Existing Solutions

This section gives a brief description of some of the most relevant solutions found in the literature, that consider agreement algorithms, schemes that deal with Rational behaviour, and BAR protocols.

2.6.1 Agreement

2.6.1.1 Terminating Reliable Broadcast

We present the Terminating Reliable Broadcast algorithm (Dolev & Strong 1983) used by Clement, Li, Napper, Martin, Alvisi, & Dahlin (2008) to implement a replicated state machine in the BAR model. This algorithm is correct in a synchronous environment, in a sufficiently connected network with reliable communication channels and authenticated messages, and when all processes are either Altruistic or Byzantine, assuming that there are up to f Byzantine processes and the number of processes is greater than or equal to $f+2$. Notice, however, that this algorithm is not tolerant to Rational behaviour, but it was proven that it can provide a Nash equilibrium if incentives are provided to Rational processes to follow the algorithm (Clement, Napper, Li, Martin, Alvisi, & Dahlin 2007; Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008).

The algorithm works as follows. In round 1, the source broadcasts the signed value. In each round r of the following f rounds, for each message m received by process i and signed by all processes of a given set S of length r such that $i \notin S$ and the first signature is from the source, i stores the received value, signs m, and sends it to all processes not in S. In the end, each process either delivers a value v if and only if it has received a single value from the source, or delivers a special null value otherwise. This protocol ensures that if a correct process delivers a value v, then all correct processes deliver that value, as it must have been some round were that value was broadcast by some correct process to all the other correct processes. That is because if a correct process i delivers a message m, then it either received m in round $f+1$ or in any round lower than $f+1$. In the first scenario, i must have received m signed by $f+1$ different processes implying that there was a correct process j that has broadcast m to all processes. In the second case, i broadcasts m to all correct processes. Nevertheless, there is always a correct process that broadcasts m to all processes, ensuring that all correct processes deliver m. This also implies that if some correct process receives a set of messages $\{m_1, m_2, \ldots, m_n\}$ signed by the source, then all the remaining correct processes also have received those messages, and, therefore, all correct processes deliver the special null value.

2.6.2 Free-Riding

2.6.2.1 Bittorrent

Bittorrent (Cohen 2003) is a partially centralized P2P file distribution protocol. To exchange content, processes establish tit-for-tat relationships. The entire network is managed by trackers, which are central servers that keep lists of downloaders for each file. The process of obtaining a file starts by contacting a tracker that returns a randomly selected set of participants. The requester establishes a relationship with each member of that set. In each interaction, processes proceed with an exchange of missing data pieces.

Interactions are managed by a local reputation mechanism based on download ratios. To maximize its reputation, processes are obliged to balance their download rate. With this, the protocol strives to achieve a *pareto efficiency* (Aumann & Dombb 2010). In economical analysis, this is a situation in which no participant can obtain a *pareto improvement*, i.e., can increase its utility without worsening the utility of others. This means that participants might commit themselves to obtain better download rates by only establishing relationships with processes with higher upload rates. Whenever the download ratio of a given process i in j reaches a level lower than a given threshold, then j "chokes" i, that is, it refuses to share files with i.

2.6.2.2 Samsara

Cox & Noble (2003) approach the problem of free-riding by relying on direct reciprocity relationships. Two particular problems are addressed. First, how to ensure that, in a P2P data storage system, processes do not have incentives to delete the stored data. For that, a challenge-response mechanism is used. This leads to the second problem of determining the appropriate action when processes fail to reply to a challenge. A grace period is used to prevent unfair data deletion.

Regarding fair contribution, processes are forced to locally garner storage claims of other participants. These claims consist on verifiable meta-data that have the same size of the information that is stored on each partner. This way, each process is forced to locally store the same amount of information that it consumes. There is an inherent overhead to this strategy that is minimized by claiming-paths. These are transitive relationships between processes, where claims along the path can be replaced by normal data. For instance, if a participant A stores a claim from B, that also holds a claim from C, then it is possible for B to replace C's claim by A's data.

Enforcing data storage requires bidirectional audits. Data owners use a simple challenge-response mechanism to monitor data storers, while, at the same time, storers audit data owners to ensure claims storage. When claiming-paths are created, these audits are forwarded along the path.

Samsara also deals with Rational behaviour when granting grace periods. Instead of waiting for the expiration of this period, participants start deleting data blocks with an increasing probability. Special care is taken to minimize situations where crashed processes unfairly lose their data, if they reply before the grace period expires.

2.6.3 BAR Model

2.6.3.1 BAR for Cooperative Services

Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth (2005), firstly introduced the BAR model and proposed a general three-layer architecture for every solution that comprises the BAR model, for cooperative services provided across multiple administrative Domains (MAD). Every step of a protocol is an exact Nash equilibrium such that it is of the process's best interest to follow the expected behaviour. A prototype for data backup is described as a proof of concept for the BAR model. The utility function considers the benefit from backing-up data, and includes communication and storage costs. It is assumed that participants do not collude.

In the replicated state machine, each operation is seen as a command that is proposed by a process, replicated among all processes. Each message is broadcast using a TRB (Clement, Napper, Li, Martin, Alvisi, & Dahlin 2007; Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008) primitive. To tolerate Byzantine behaviour, each message is signed and follows a strict format. Therefore, any incorrectly formatted message presents a proof of misbehaviour (POM) and is sent to every process using the TRB primitive. When it is not possible to prove that a participant misbehaved, processes can insert suspected processes into a bad-list. This list is exchanged periodically and when a participant is included on a quorum (majority) of bad-lists, it is classified as Byzantine and removed from the system.

Enforcing fair participation requires that all participants have the opportunity to submit a command. This is done by dividing the time in operations and rounds. Each participant submits orders when elected as the leader of the first round of an operation. This election is achieved by using a verifiable pseudo-random number generator, such that no participant can manipulate the election, and the process is verifiable. The first leader of an operation has the ability to submit a totally new order. If it fails, another leader is elected for the next round. To prevent participants from ignoring steps of the protocol, a message queue for each user is locally stored on each participant. Whenever a message is expected from a user, a hole is introduced in its queue, so that it cannot send another message until it dispatches the expected one. All the messages are balanced such that the expected behaviour is never more costly than any alternative. To deal with message delay, penance messages must be sent periodically, removing any benefit obtained from that delaying.

Another mechanism is implemented to force processes to perform the expected work, such as answering to requests or periodically sending messages. More precisely, a trusted witness mediates every interaction, and ensures that each process operates as expected. Since every message is broadcast, the full membership provides an implementation for that witness. This mechanism combined with message queues denies service to a process that fails to send any message.

An optimization allows a participant i to transfer a large value to another process j without sending it to the witness. Instead, i sends a summary of the request to the witness, who forwards it to j. The complete request is sent directly from i to j. Then, j sends the complete reply to i and a summary to the witness, who forwards it to i. If however i does not transfer the information to j or vice versa, then the protocol proceeds as normal, and the original request must be sent directly to the witness.

In the BAR backup prototype (BAR-B), all the participants possess a certain quota of storage on other processes. This is ensured by periodically exchanging storage information about the neighbours, and hence disseminating information about possible unbalanced relationships. To prove that data is not discarded, processes that store data must periodically reply to a challenge. Each block of data possess a validity controlled by a lease. Later, when the storer is challenged, it must provide a valid reply, a proof that data was deleted, or a proof that the lease has expired. Any other reply constitutes a POM. Non-responses are avoided by the global witness mechanism.

2.6.3.2 BAR Primer

Clement, Li, Napper, Martin, Alvisi, & Dahlin (2008) complemented the work of Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth (2005), by formalizing the problem of Terminating Reliable Broadcast (TRB) as a repeated game with infinite number of executions. Here, they identify the protocol of Dolev and Strong (Dolev & Strong 1983), presented in Section 2.6.1.1, as a possible strategy for a Rational participant to adopt. Using a game-theory analysis (Martin & Ariel 1994), they prove that the algorithm is not a Nash equilibrium. As an alternative, they propose the protocol *Just TRB* that uses mechanisms, such as cost balancing, penance messages, and predictable communication patterns (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005), to provide incentives for Rational participants to behave as expected. They also prove that this protocol provides a Nash equilibrium.

For the game theory analysis, the authors formally defined the game as an infinite sequence of instances of TRB, in which a leader broadcasts a value to all the other processes. It is assumed that the leader is non-Byzantine for an infinite number of executions. Four properties are defined as necessary for ensuring liveness and safety on TRB, namely, *Validity*, *Integrity*, *Agreement*, and *Termination*, which are the normal properties of TRB in an environment with Byzantine behaviour. Therefore, this problem can also be seen as a Byzantine agreement.

To prove that the TRB of Dolev and Strong (D-S TRB) is not a Nash equilibrium, the authors identify a possible alternative strategy and demonstrate that the utility is greater than the obtained by following D-S TRB. Concerning Just TRB, it is proven that the lower-bound to all the possible utilities obtained from obeying the protocol is greater than the upper-bound to all the utilities obtained from deviating from the protocol. As a corollary, it follows that Just TRB provides a Nash equilibrium.

2.6.3.3 BAR Gossip

BAR Gossip (Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006) uses the BAR model to tolerate both Byzantine and Rational processes in a peer-to-peer gossip protocol. In this type of protocols, processes exchange information with a random set of neighbours in each interaction.

This ensures, with high probability, that, after a sufficient number of interactions, data is spread across the network. BAR Gossip creates an exact Nash equilibrium for participants to follow every steps of the protocols and to ensure fair exchange of data. It is assumed that processes do not collude.

A trusted central server called tracker is responsible for starting the dissemination process. Data is partitioned into several blocks. Initially, the source selects a random group of processes and sends different subsets of blocks of data to each selected participant. Then, for a certain number of rounds, each process choose other neighbours to exchange missing information in a fair manner.

Byzantine processes have several opportunities to attack the system by conditioning the dissemination process. To deal with this problem, in each round a pseudo-random number generator is used to select the partners of each participant. This also has the aim of mitigating collusion and denial of service. In addition, each message is signed so that, as in (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005), any incorrectly formatted reply constitutes a POM. These proofs are spread by sending it to the source, who will disclose an eviction order. To provide incentive for sending POMs to the origin, the sender is included in the next set of users that receive all the updates in the initial round.

The system strives to avoid free-riding by demanding a balanced exchange between processes. Partners calculate a set of updates that each one is missing, send them ciphered with session keys, and only then they exchange session keys. This delay is to give incentive for rational processes to obey all the processes of the protocol, in an approximate solution to the Fair Exchange problem. The fact that participants still avoid sending the encryption keys is mitigated by a penance mechanism. A process that fails to send the key is bombarded with pestering messages. These extra costs may dissuade processes to misbehave.

The system also addresses the possibility of participants lying about their history, to avoid spreading updates. With the balanced approach, processes obtain the same amount they send, so they are encouraged to exchange as much as possible. In the unbalanced protocol, a process may under-report the updates it has available. BAR-Gossip does not solve this problem, but relies on the fact that under-reporting increases the risk of receiving a repeated update. Whenever a participant promises to send an update and does not meet that promise or when it sends invalid data, this will be used as a POM against it, since all messages are signed. Any invalid session key that is traded is also used as an accusation alongside with the ciphered information.

2.6.3.4 Flightpath

The authors of BAR Gossip (Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006) proposed several improvements on the previous solution to allow a more flexible solution. Unlike the two other BAR implementations (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005; Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006), in Flightpath (Li, Clement, Marchetti, Kapritsos, Robison, Alvisi, & Dahlin 2008) an approximate Nash-equilibrium is used to obtain better performance.

Besides being the source of information, the trusted tracker is also responsible for managing membership by constantly pinging all the participants. Special care is taken to deal with churn (Liben-Nowell, Balakrishnan, & Karger 2002). More precisely, the authors try to avoid the situation where a significant amount of processes join the network simultaneously and, since they do not have anything to exchange, they overwhelm previously joined participants with requests for new updates without contributing to the system. Two solutions are proposed. One is to divide time in epochs. Each epoch is associated with a list of members. When a process joins the network, the tracker places it in the epoch $e + 2$ where e is the current epoch. When moving from e to $e + 1$, the tracker shuffles the list of members from the new epoch, in order to prevent processes from taking advantage from a fixed list, and disseminates it. This solution is concerned with how to spread membership information in an efficient manner. However, it does not allow a recently joined process to immediately start receiving updates. That problem is solved with an algorithm known as the Tub algorithm. Here, the list of members is reordered: i) participants that joined the network before $e - 2$ are ordered in a random fashion, where e is the current epoch; ii) the remaining participants are ordered by entrance in the network. This list of members is partitioned into groups called tubs. Recently joined processes are placed in the most recent tub. Then, they create a view of its potential partners with whom they can exchange updates. That view is composed by all the members of the process's tub and by an exponentially decreasing number of participants from the older tubs.

This protocol uses the same strategies of BAR Gossip for preventing free-riding. However, it allows processes to establish concurrent trades with multiple partners. This introduces the problem of a process being overloaded with concurrent requests. To mitigate this, a mechanism of reservations is used to limit the total number of neighbours with whom each process communicates, by evenly distributing the load among processes. Additionally, the number of updates traded is limited so that different concurrent neighbours do not send a repeated update. Instead of trading all the missing updates, each neighbour sends a subset of it. Still, there is the possibility of some missing updates not being exchanged. Erasure-codes (Plank 1997) are proposed to mitigate this problem. Moreover, slightly unbalanced exchanges are allowed. Still, each process keeps an imbalance ratio for each of its neighbours, that is upper bounded, to avoid free-riding.

2.6.3.5 FireSpam

FireSpam (Mokhtar, Pace, & Quéma 2010) is a gossip system for disseminating messages, that deals with the problem of filtering SPAM messages, that is, unwanted messages. The main challenge that the authors face is to implement such a system, modelling the behaviour of participants with the BAR model. For that, it is proved that the protocol is an exact Nash equilibrium.

The long-term benefit of Rational participants is considered to be the receipt of correct messages (not SPAM). Furthermore, Rational processes avoid SPAM messages and costs of communication. With that in mind, the authors proposed a ladder overlay. Messages are propagated from the bottom of the ladder to the top. Hence, processes on the lowest levels are more subjected to SPAM than processes on the upper levels. As an incentive for filtering SPAM, participants are able to *climb* the ladder as much as the capability of the filtering they perform. Each gossip operation consists on a process publishing a message on the lower level participants. Then, whenever a process receives a message, it forwards to a preselected group of participants.

The main incentive for Rational processes to behave as desired is based on black-lists. Each participant keeps a black-list of processes that previously misbehaved. This list is propagated on each published message. One important remark is that every publisher has to submit a fixed size list, even if it is empty. This is for balancing costs so that Rational processes do not avoid sending their lists. During a process of dissemination, Rational processes do not forward messages to black-listed participants, otherwise they are included in other processes black-list. It is assumed that Rational processes avoid the costs of being black-listed.

This solution also addresses the problem of determining the filtering capability of each process, to determine the level where they should the placed, in the overlay. For this purpose, the overlay Fireflies (Johansen, Allavena, & van Renesse 2006) is used to select a group of monitors for each process. Their responsibility is to monitor the filtering capability of the monitored process, by receiving reports for each message that the participant forwards from the processes that receive the forwarded message. If a monitor receives a majority of reports, it broadcasts another report to the other monitors, otherwise it places the process in its blacklist. If a majority of monitors places the process in their black-lists, an eviction order is broadcast to the network. Whenever a monitor or forwarded process fails to send a report, it is included in the black-list of the participant that detects this misbehaviour.

2.7 Summary

This chapter provided an overview on the related work of this thesis. We first described the most relevant types of faults, causes of faults, and fault models for the behaviour of processes. Then, we surveyed previous work on data transfer for an environment in which processes are either faulty or obedient. We also described previous work on dealing with Rational behaviour and on simultaneously tolerating Byzantine and Rational behaviour. We concluded the chapter by describing some of the most relevant solutions related to the NBART problem.

3 NBART Problem

In this chapter, we provide a more precise definition of the NBART problem. In Section 3.1, we start by providing an informal definition. In Section 3.2, the main challenges of NBART are identified, which are used as motivation for the definition of the NBART properties in Section 3.3. In Section 3.4, we show that none of the solutions to related problems solve NBART. Finally, in Section 3.5, we describe two use cases where NBART could be applied.

3.1 Informal Definition

The NBART problem can be defined as the transfer of a value v from a producer to a consumer, in the BAR model. The value is produced by a deterministic function (*produce(p,v)*) used by each producer to input the value v to the transfer. We denote by $\bar{v}$ the value produced in a non-Byzantine process (that we simply name the *correct value*), as the value one would expect to be produced in a deterministic non-faulty computation. Each consumer c obtains a value v and consume it by invoking *consume(c,v)*.

Given that a particular producer may be Byzantine and the consumer has no previous knowledge of $\bar{v}$, it is not possible to trust on the value v sent by a single producer. In fact, it is required the participation of $N_{\mathcal{P}} \geq f_{\mathcal{P}}+1$ producers such that consumers may be able to receive the correct value from at least one non-Byzantine producer, where $f_{\mathcal{P}}$ is the upper bound on the number of Byzantine producers. If simple majority mechanisms are used, then $N_{\mathcal{P}} \geq 2f_{\mathcal{P}}+1$ in order to have at least a majority $(N_{\mathcal{P}} - f_{\mathcal{P}})$ of non-Byzantine producers to produce the same value. A reliable transfer also requires at least one non-Byzantine consumer to consume the value, hence $N_{\mathcal{C}} \geq f_{\mathcal{C}} + 1$, where $f_{\mathcal{C}}$ is the upper bound on the number of Byzantine consumers (Figure 3.1).

NBART may be executed multiple times, by having the consumers of a given instance i to serve as producers for the next instance $i+1$. This way, a given value may be preserved across

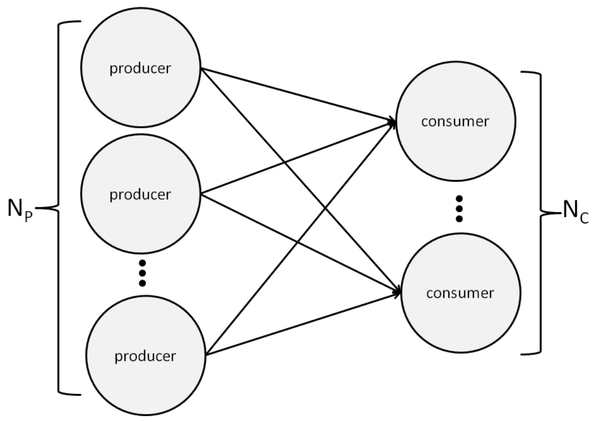

Figure 3.1: NBART: Reliable transfer of data.

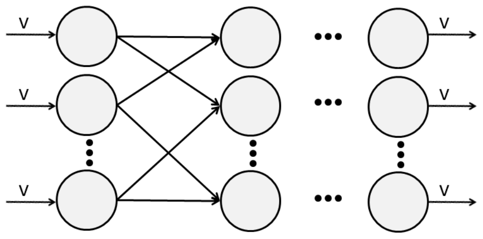

Figure 3.2: Data preservation through successive transfers.

multiple instances of NBART, until the transferred data is no longer required (Figure 3.2). For this to be possible in an environment where processes may be Byzantine or Rational, the transfer must be reliable:

- All non-Byzantine producers should produce the same value $v = \bar{v}$.

- All non-Byzantine consumers should consume the same value $v = \bar{v}$.

3.2 Challenges

3.2.1 Byzantine-Altruistic Solution

To motivate the challenges involved in solving the NBART problem, we will use an algorithm that solves this problem under a simpler model, namely in a synchronous system, with reliable channels, in the presence of Byzantine processes but no Rational processes, i..e, in a setting where all non-Byzantine processes follow the algorithm altruistically.

In such a setting the following simple algorithm solves the N-party Byzantine-Altruistic (BA) Transfer problem, assuming for simplicity that $N_{\mathcal{P}} = N_{\mathcal{C}} = N$, $f_{\mathcal{P}} = f_{\mathcal{C}} = f$, and $N \geq 2f + 1$:

- In the first round, every producer signs its value and send the value and its signature to every consumer. In this round, Byzantine processes may omit to send the value to one or more recipients, send arbitrary values or malformed messages (for instance with a wrong signature).
- In the second round, each consumer selects a value v such that v appears in at least $f + 1$ messages from different producers and consumes v.

It is easy to show that such protocol achieves the required reliability in the Byzantine failure model, since, by assumption, all non-Byzantine producers are Altruistic and, therefore, produce the same value $v = \bar{v}$ and send it to all the consumers. Since there are at most f Byzantine producers, at the end of the first round each consumer will have $f+1$ values from non-Byzantine producers. Given that all non-Byzantine consumers are Altruistic, they all consume the same value, which must be $\bar{v}$.

However, this is no longer the case in the BAR model. Rational producers may not have any incentives to send the value to all the consumers if sending the data to a subset of consumers provides a greater expected benefit. This is particularly a problem when considering the possibly high communication costs of transferring an arbitrarily large value. Unfortunately, if a rational process opts not to send the correct value to one or more consumers, those consumers may be unable to collect the required majority to consume the value. To overcome this problem, it is necessary to deal with Rational behaviour.

3.2.2 Dealing with Rational Behaviour

Rational volunteers only offer their resources if some incentives are provided for their contribution. Incentives may be obtained directly from the transfer if for instance producers benefit directly from the fact that consumers obtain their value. However in the general case processes do not benefit directly from performing the required tasks, hence some sort of indirect incentive must be provided, such as an indirect reward, a punishment for not accomplishing certain goals, or even the effect that current actions might have on future instances of NBART.

A concrete example of an indirect reward mechanism is the credit system employed by Boinc (Anderson 2004), where volunteers are rewarded with credits that are used to rank processes in a public "merit" chart, and that may be subsequently used as a currency for consuming system resources.

In Sections 2.4 and 2.5, we have seen several mechanisms used to provide incentives for Rational processes to not misbehave, such as direct reciprocity, reputation, tit-for-tat relationships, among others. We can classify those incentives in two major classes, namely incentives provided basing on a single interaction among processes and incentives provided basing on multiple interactions. Since we do not assume that processes interact through a protocol other than an NBART protocol, then we can only distinguish between *incentives provided in multiple instances* of NBART and *incentives provided in a single instance.*

3.2.2.1 Incentives Provided in Multiple Instances

Incentives provided in multiple instances rely upon multiple interactions among processes, through the NBART primitive. In these types of incentives, each action performed by each process i may have an effect on the future utility of i, generally perpetuated by some other process j or sets of processes. That effect can be beneficial if i obtains a greater benefit by performing certain operations, where the most common mechanism used is *direct reciprocity*; or it can be harmful if some sort of punishment is applied to i for not performing certain operations, in which case the most common mechanism used is *tit-for-tat.*

As we have seen in Section 2.4.1, in direct reciprocity mechanisms, each process always provides a service in exchange for the service it receives. For instance, if we could assume that consumers obtained some direct benefit from receiving the correct value, then the main incentive for a producer p to send a value to a consumer c would be that in a later interaction c would also send a value to p. However, in NBART, we do not want to assume that processes obtain any direct benefit from performing the specified operations. Tit-for-tat mechanisms were presented as a particular case of direct reciprocity in which the retribution is harmful. For instance, if a producer p omits a message to c, then c may also omit a message to p in future interactions, therefore denying some benefit to p. Other mechanisms such as reputation schemes could be used to provide incentives on multiple instances of NBART. For instance, if a process i would not send a message to j, then j could reduce the local reputation of i, and when the local reputation of i in j would drop below a certain threshold, then j would avoid providing any service to i.

The main obstacle to applying these mechanisms is the fact that they are only effective if the future effects are expected with sufficiently high probability, in order to persuade processes to perform all the required operations. That is of course dependent on the frequency by which processes interact and consequently depends on the mechanism used to select the sets of consumers and producers, in each instance of NBART. To ensure a reliable transfer among sets of processes, this mechanism must ensure that with high probability the number of Byzantine processes in each of the sets of producers and consumers has a known upper bound, and faults occur independently. Although in this thesis, we do not propose any mechanism for fulfilling those requirements, such mechanism must depend on the membership of the system, which, in a dynamic peer-to-peer network, may vary significantly from instance to instance. Therefore, if a producer p_i interacts with a consumer c_j during a given instance r, there is no guarantee that both processes will ever interact in future instances, and incentives provided in multiple instances of NBART might not be sufficient to persuade processes to perform all the required operations.

Due to this fact, we are interested in NBART solutions devised for non-repeated interactions, in which the benefits and costs are determined according to the information gathered from a single instance of NBART.

3.2.2.2 Incentives Provided in a Single Instance

To provide incentives in a single instance of NBART, processes must prove that they correctly participated in a transfer. That is, producers must prove that they sent the value to enough consumers and consumers must prove that they consumed the correct value. These proofs contain the *observable behaviour* of each process i, i.e., the information which determines the final utility of process i, after engaging in a single transfer.

We model the mechanism of providing incentives to processes using the abstraction of a *certification* mechanism, which consists on collecting the data which contains the *observable behaviour* of each process i in a data structure named *evidence*, to assess the behaviour of each process. Appropriate rewards or punishments may be provided accordingly. How the *certification* mechanism is implemented is orthogonal to the NBART problem definition: it may be implemented by a logically centralized trusted component that gathers cryptographic evidence from each process, or by some decentralized system (in the lines of decentralized reputation systems).

Regardless of the implementation of the *certification* mechanism, it is impossible to assess the behaviour of a producer without the help of the consumer. Therefore, to cope with rational behaviour, NBART requires consumers to publicly *acknowledge* the receipt of the correct value from the producer. Furthermore, to ensure that rational consumers have the incentive to produce such acknowledgements, we assume that they are also rewarded for that task.

Two predicates, which we denote by *hasProduced(p_i,evidence)* and *hasAknowledged(c_j,evidence)*, are defined on the *evidence* such that one can extract information about the performance of each individual producer p_i or consumer c_j, respectively. More precisely, if these predicates are true, then the utility of the respective producer p_i or consumer c_j must be positive, so that they may have incentives to engage in the transfer.

Notice that different implementations of NBART may require each producer to send its value to a different set of consumers (or even to other producers). Therefore, both the evidence and the associated predicates are implementation dependent.

3.3 Properties

The NBART problem can be stated more precisely by the following main 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):* Eventually, every non-Byzantine consumer consumes a value.
- **NBART 5** *(Evidence):* The *certification* mechanism eventually produces the *evidence.*
- **NBART 6** *(Producer Certification):* If producer p_i is non-Byzantine, then *hasProduced (evidence, p_i)* is *true*
- **NBART 7** *(Consumer Certification):* If consumer c_j is non-Byzantine, then *hasAknowledged (evidence, c_j)* is *true*

With these definitions in mind, we can provide a more precise characterization of the *benefits* that Rational processes 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.* Notice that in the certification properties, we state that if a process i is non-Byzantine, then i obtains a positive utility, ensuring that non-Byzantine processes have some incentives to follow that protocol.

3.4 Related Problems

We now describe some of the most relevant past works that addressed similar challenges to the ones posed by NBART, and we explain why such solutions cannot be directly applied to solve the NBART problem. More precisely, we will compare NBART with the classical problems of agreement and Byzantine replication in the Byzantine fault tolerance literature, presented in Section 2.3. We will also establish a comparison with the problems related to Rational behaviour, namely free-riding and non-repudiation, described in Section 2.4. We will conclude with a comparison with problems solved in the BAR model, with special emphasis given to agreement and data dissemination.

3.4.1 Byzantine Fault Tolerance

3.4.1.1 Agreement among Processes

Solutions to the problems of Reliable Broadcast and Consensus, described in Sections 2.3.1 and 2.3.2 may be used by each producer to reliably broadcast its value to all the consumers, in the presence of Byzantine and Altruistic processes. 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. More precisely, each agreement protocol would have a message complexity of at least $O(Nf)$ (Dolev & Strong 1983), resulting in a total complexity of $O(N^2 f)$. Probabilistic Reliable Broadcast protocols may be used to reduce the complexity of agreement protocols. However, only probabilistic guarantees are provided.

Another major difference is that these protocols are executed among a single set of processes, while NBART requires the existence of two sets of 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 (Lamport 1998), where each proposer may propose a different value. An alternative would be to use a leader-election protocol to select one producer as a leader, responsible for gathering the values from all producers, determining the correct value, and sending it to all consumers through a Paxos algorithm. This has the clear advantage of reducing the message complexity. However, this solution would have several disadvantages. Firstly, it would be necessary to deal with the failure of the leader, which would result in a high message complexity in the presence of Byzantine faults. Secondly, the leader would become a bottleneck. Thirdly, to tolerate Byzantine behaviour, it would be required the existence of at least $3f + 1$ consumers for f Byzantine faults (Lamport 2011). Finally, as in other protocols of classical agreement, this solution only considers Byzantine and Altruistic processes, and therefore is vulnerable to Rational behaviour (Clement, Napper, Li, Martin, Alvisi, & Dahlin 2007; Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008).

3.4.1.2 Byzantine Replication

In Section 2.3.3, we also described a set of solutions to the problem of implementing registers with different concurrency semantics using Byzantine quorum systems or the replica-based approach. In both cases the objective is to ensure that Byzantine processes 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 participants that produce the data independently, although following a deterministic function, to another set of participants which have to determine which is the correct data.

3.4.2 Problems Derived from Rational Behaviour

3.4.2.1 Free-Riding

Inn Section 2.4.1, we presented some of the mechanisms used to provide incentives for Rational processes to follow the protocols. These incentives were proposed in the context of file sharing and data backup systems, whose main aim is to solve the problem of free-riding.

The main problems with these solutions is that they provide incentives using mechanisms such as direct reciprocity and tit-for-tat relationships. As we have stated in Section 3.2, we do not want to use any of these mechanisms in NBART. Another major problem is that these solutions do not prove that it is on every Rational process interest to follow each step of the protocol.

For instance, Bittorrent (Cohen 2003) is not fully resilient to Rational behaviour (Shneidman, Parkes, & Massoulié 2004). This happens due to the beginning of an interaction, when processes that upload files allow participants that recently joined the network to keep a low upload/download ratio. Rational processes can take advantage of this fact to continuously download content from different sources without reciprocating. A similar strategy can be used by Rational processes in Samsara (Cox & Noble 2003) to overcome the grace period: after two processes i and j agree on storing the data of each other and transferring it, i may immediately discard the data sent by j and wait for the grace period to expire before choosing another partner. Meanwhile, j may continue to store its data.

3.4.2.2 Non-Repudiation

Non-repudiation and fair-exchange protocols described in Section 2.4.2 may be used by producers to exchange cryptographic information, which may serve as proof of their correct participation in the transfer. However, these kind of protocols only deal with the problem of exchanging information in a fair manner. In an environment with Byzantine behaviour, these protocols do not ensure that a transfer will ever occur, as some of the processes might be Byzantine and may never send any information to any other process. That is because the timeliness property only ensures that the transfer eventually ends, it does not guarantee that after a finite amount of time all the processes obtain the desired information.

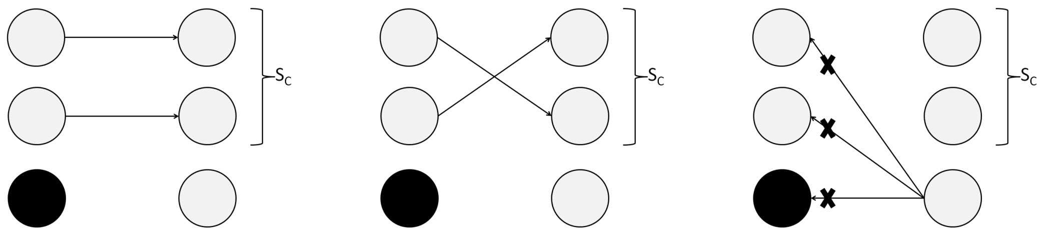

Figure 3.3: Problem of applying non-repudiation, for $f = 1$ and $N = 3$.

Nevertheless, simple non-repudiation could be used by each producer p to send and Evidence Of Origin (EOO) to a consumer c along with the value. Then, c would have to send an Evidence Of Receipt (EOR) to p. The EOO would allow c to prove that it has received and acknowledged the value from p, while p could use EOR to prove that it has sent the value to c. Then, each producer could be rewarded by gathering a minimum number of different EOR cryptographic structures, whereas consumers could gather a minimum number of EOO cryptographic structures. In either case, in the presence of N producers and N consumers, and up to f Byzantine processes in each of the sets of producers and consumers, it would not be possible to require processes to gather more than N-f cryptographic structures.

In a scenario where there are exactly f Byzantine producers and less than f Byzantine consumers (Figure 3.3) , it is possible for each of the $N-f$ non-Byzantine producers to gather $N-f$ acknowledgements from all the consumers of a given set S_C of $N-f$ non-Byzantine consumers. In this scenario, no non-Byzantine producer has any incentive to establish an exchange with any non-Byzantine consumer that does not belong to S_C. Hence, the non-Byzantine consumers that do not belong to S_C are unable to retrieve the value from any non-Byzantine producer, violating the NBART base properties.

3.4.3 Related Problems in the BAR Model

3.4.3.1 BAR Agreement and Byzantine Replication

In Section 2.6, we described a solution for Byzantine agreement in the BAR model (Clement, Li, Napper, Martin, Alvisi, & Dahlin 2008) and a system that uses this solution to implement a replicated state machine (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005). However, the incentives provided to Rational processes are either based on ∞-tit-for-tat mechanisms, that we do not want to use in NBART, or on cost balancing techniques, that cannot be used to balance the cost of transferring an arbitrarily large value because the overhead would be too high. Also, the guaranteed response mechanism, that could be used to guarantee that a producer sends a value a to a consumer, requires the active participation of a witness, which must be implemented through message broadcast to all the remaining processes.

3.4.3.2 Data Dissemination

The BAR Gossip (Li, Clement, Wong, Napper, Roy, Alvisi, & Dahlin 2006) and Flight-Path (Li, Clement, Marchetti, Kapritsos, Robison, Alvisi, & Dahlin 2008) data dissemination algorithms described in Section 2.6 are not directly applicable to solve NBART as they assume that the source of the information is trusted and provide no deterministic guarantee that the disseminated information reaches its destination. Furthermore, data transfer between each pair of processes is performed using direct reciprocity in a fair exchange interaction. The FireSpam (Mokhtar, Pace, & Quéma 2010) system filters the spam sent by the source, but it is not concerned with the validity of the information being disseminated.

3.4.3.3 k-Fault-Tolerant Nash Equilibrium

In Section 2.5.2, we described work that did not use the BAR model for dealing with Rational and Byzantine behaviour. This work introduced the notion of k-Fault-Tolerant Nash equilibrium (k-FNTE) in the presence of up to k Byzantine processes. This concept was applied to auction games and virus inoculation games, but it was not extended to more complex distributed games with communication between participants. Furthermore, the notion of (k, t)-robustness has only been applied to solve the problem of secret sharing. On the contrary to our work, it was 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 processes take into consideration communication costs can be (k, t)-robust (Clement, Napper, Li, Martin, Alvisi, & Dahlin 2007).

3.4.4 Discussion

In conclusion, to the best of our knowledge, no previous work in the literature addresses the problem of transferring an arbitrarily large value from a set of producers to a set of consumers, where participants can be Byzantine, Rational, or Altruistic, and incentives are provided in a single interaction among processes.

Table 3.1 summarizes the comparison among solutions to NBART and solutions to related problems, in terms of following desired properties:

1. Deterministic reliable data transfer
2. Two sets of processes: producers and consumers
3. Take advantage of multiple producers that produce the same value
4. The source is not trusted
5. Deal with Byzantine behaviour
6. Deal with Rational behaviour
7. Resilient to the problem of using simple non-repudiation
8. Resilient to free-riding
9. Incentives provided in a single instance
10. Consider communication costs among processes
11. Do not require the existence of altruism

3.5 Use Cases

In order to clarify how NBART could be used in practice, we describe two possible applications where this primitive would be useful: a volunteer computing system named BARRAGE, and a data backup system named BARCKUP.

Properties / **Solutions**	1	2	3	4	5	6	7	8	9	10	11
Solutions to classical agreement	✓	×	×	✓	✓	×	-	-	-	-	-
Paxos	✓	✓ [1]	×	✓	✓ [2]	×	-	-	-	-	-
Solutions to Byzantine Replication	✓	×	×	✓	✓	×	-	-	-	-	-
Classical data dissemination solutions	×	×	✓	✓ [3]	✓	×	-	-	-	-	-
Bittorrent and Samsara [4]	-	-	-	-	×	✓	-	×	×	✓	✓
Solutions to non-repudiation [4]	-	-	-	-	×	✓	×	✓	✓	-	✓
Preliminary work to the BAR model [4]	-	-	-	-	✓	✓	-	✓	×	×	✓
BAR-B and BAR Primer	✓	×	×	✓	✓	✓	-	✓	×	✓	✓
BARGossip, FlightPath	×	×	✓	×	✓	✓	×	✓	×	✓	×
FireSpam	×	×	✓	×	✓	✓	×	✓	×	✓	✓
Solutions to NBART	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

[1] Producers are the proposers and consumers are simultaneously acceptors and learners;
[2] Paxos solutions tolerant to Byzantine behaviour were already proposed (Martin & Alvisi 2006; Lamport 2011);
[3] Some previous works dealt with the possibility of the source being untrusted (see for instance the work of Chow & Renesse (2010));
[4] These works do not specifically deal with the problem of reliably transferring data.

Table 3.1: Comparison between previous solutions and possible solutions to NBART.

3.5.1 Use Case: BARRAGE

In the genesis of our work is the desire to build a hybrid peer-to-peer system, inspired by systems such as Boinc (Anderson 2004), that allows to use computational resources (cpu and storage) provided by volunteers to execute parallel computations. We name this system BARRAGE. Contrary to Boinc, which is limited to computations based on the bag-of-tasks paradigm, BARRAGE should also support computations using computational models that require communication among tasks, such as MapReduce (Dean & Ghemawat 2004). In addition, BARRAGE should tolerate Byzantine and Rational behaviour. Hence, an implementation of this system would be similar to the BFT MapReduce implementation of Hadoop (Costa, Pasin, Bessani, & Correia 2011), aside from the mechanisms implemented in BARRAGE for ensuring a reliable transfer of data and for dealing with Rational behaviour.

Similarly to projects such as SETI@home (Anderson, Cobb, Korpela, Lebofsky, & Werthimer 2002), we assume that computations are coordinated by a logically centralized entity. Volunteers register with the coordinator, indicating the periods of time when they are (expected) to be available and which type of resources they are willing to share. Also, similar to SETI@home, volunteers aim at having their effort rewarded in some form, simply by having the names of the users in a list of contributors, or by getting credits that allow them to use the infrastructure as clients at a later time (i.e., submit their own computational tasks).

The coordinator is responsible for assigning tasks to the volunteers. For instance, the coordinator may instruct a volunteer to participate in a computation as a reducer, indicating the credentials of the mappers and of the other reducers involved in the computation and the time at which the data transfer is expected to take place. In some cases, intermediate results might need to be stored in the system for periods longer than the volunteers are willing to offer. In this case, intermediate results might need to be transferred from a set of intermediate storage processes to another set, benefiting from the fact that different volunteers will be available in different time periods.

It is clear from the description above that NBART is a key component of our architecture. Mappers will be involved in a NBART protocol to transfer their results to the reducers or to intermediate storage processes by invoking *produce*. Temporary storage processes will be involved in NBART executions to receive the data by invoking *consume*, and to transfer it to the next generation of storage processes by invoking *produce*. Reducers engage in NBART to obtain the data from mappers or storage participants and make results available through the primitive *consume*. The central entity could be responsible for gathering the *evidence* from information sent by consumers in each instance of NBART, and by executing the *certification* mechanism, rewarding each participant that behaved correctly with credits.

3.5.2 Use Case: BARCKUP

Another application of NBART could be a data backup system similar to BAR-B (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005) that would leverage on the resources provided by volunteers to backup personal data. This system could be named BAR data baCKUP (BARCKUP). This system would be appropriate for peer-to-peer networks, unlike BAR-B that relies on the cooperative services provided by a restricted set of processes that continuously interact among each other.

As in BAR-B, information regarding the reputation of each process, i.e., the ratio between its contribution and its resource consumption, would be registered. A single trusted entity could be responsible for storing this information and for registering the periods of time during which processes would be available for storing data and the amount of space available in each process, similarly to BARRAGE. This central coordinator would also be responsible for selecting the sets of processes that would backup a given block of data during a certain period of time.

In this scenario, a backup operation would proceed as follows. The user would request an initial set of consumers to the central coordinator. Then, the user would send the data to the consumers by invoking *produce*. In this case, the user would also have to send a special authorization signed by the coordinator, allowing the consumers to trust the value sent by the user as if it was sent by a majority of producers. Notice, however, that to prevent free-riding, the coordinator would only grant an authorization to the user if its reputation was higher than a certain threshold. After a certain period of time, those consumers would act as producers of another instance of NBART and transfer the data to another set of consumers, provided by the coordinator.

To provide incentives to Rational processes, the coordinator would be responsible to produce the *evidence* and reward processes by increasing their reputation whenever they behaved correctly in a NBART instance. This reward would be used as a payment to allow those processes to backup their own data. Since the periods of time during which processes store the data may be long enough for some processes to crash, it would be fair to reward processes for the time they stored the data. For that, a set of monitors could be designated to each process i, in order to periodically check if i continues to store the data it was supposed to store. For this purpose, a mechanism as the one proposed in Fireflies (Johansen, Allavena, & van Renesse 2006) could be used. However, this mechanism would have to be modified in order to deal with Rational behaviour.

3.6 Summary

In this chapter, we defined the NBART problem. We started by providing an informal definition, which allowed us to identify the main challenges that must be faced when dealing with Rational behaviour in the NBART problem, and for that we proposed a simple solution to this problem for an environment where processes are either Altruistic or Rational. Then, we defined the NBART properties and showed that no solution to related problems solves all the challenges posed by the NBART problem. We concluded the chapter by describing two use cases where NBART would be useful.

4 Synchronous Risk-Averse Solutions

This Chapter provides two synchronous solutions for the NBART problem considering that processes are risk-averse, namely the *Eager Risk-Averse NBART* (ERA-NBART) and *Lazy Risk-Averse NBART* (LRA-NBART) algorithms.

In Section 4.1, the generic system model for both algorithms is described. Then, in Sections 4.2 and 4.3, we provide the full description of ERA-NBART and LRA-NBART, along with the proofs of correctness, a game theoretic analysis, and a complexity evaluation. In Section 4.4 we compare the two algorithms in terms of their complexity. Finally, in Section 4.5, we discuss a possible extension to the presented solutions.

4.1 System Model

The NBART problem involves a set of *producers* $\mathcal{P}$ ($\#\mathcal{P} = N_{\mathcal{P}}$) and a set of *consumers* $\mathcal{C}$ ($\#\mathcal{C} = N_{\mathcal{C}}$). For simplicity, we start by assuming that $N_{\mathcal{P}} = N_{\mathcal{C}} = N$, when initially presenting the algorithms. We also assume that $f_{\mathcal{P}} = f_{\mathcal{C}} = f$. Then, in Section 4.5 we extend the algorithms for the case where it may be true that $N_{\mathcal{P}} \neq N_{\mathcal{C}}$ and $f_{\mathcal{P}} \neq f_{\mathcal{C}}$. We assume there is a special abstract entity called *trusted observer* (TO) responsible for executing the *certification* mechanism, i.e., for gathering the *evidence* and properly rewarding producers and consumers.

We assume that the system is purely *synchronous* 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 algorithm, 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 collisionresistant 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. 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 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. Furthermore, we assume that Rational processes are risk-averse, i.e., when considering the expected benefits and costs, they assume that the worst possible scenario for their utility may occur and therefore follow a strategy that does not incur in any risk of not obtaining benefits.

4.2 ERA-NBART

We now present the ERA-NBART algorithm. 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, named $producerset_j$ as the set of producers that may send the value to consumer c_j, i.e., $producerset_j = \{p_i | c_j \in consumerset_i\}$.

4.2.1 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 four rounds:

- **Round 0** Each producer produces the value.
- **Round 1** Each producer p_i sends the value along with the signature to all consumers of $consumerset_i$, and sends only the signature to all consumers $c_j \notin consumerset_i$.
- **Round 2** Each consumer consumes the value and sends the cryptographic information to TO.
- **Round 3** TO produces the evidence and rewards processes according to the value of the predicates.

Algorithm 1: NBART Algorithm (producer p_i)

```
01 upon init do
02    myvalue := $\bot$;
03    myhash :=$\bot$;
04    myhashsig := $\bot$;
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]);
```

4.2.2 Algorithm in Detail

In round 0, a producer computes the hash of the value and signs it (Alg. 1, lines 6-8). When round 1 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 $\bot$, 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 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 $\bot$ 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

Algorithm 2: NBART Algorithm (consumer c_j)

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

07 upon nextRound ∧ round = 0 do // start of round 1
08    round := 1;

09 upon deliver ($p_i$, $c_j$, [VALUE, pvalue, phash, phashsig, msgsig]) ∧ 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[$p_i$] := ⟨pvalue, phash, phashsig⟩;

15 upon deliver ($p_i$, $c_j$, [SUMMARY, phash, phashsig, msgsig]) ∧ 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[$p_i$] := ⟨⊥, phash, phashsig⟩;

20 upon nextRound ∧ 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[$p_i$] = ⟨*, myhash, *⟩ do
25       confirm[$p_i$] := ⟨values[$p_i$].hash, values[$p_i$].signature⟩;
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);
```

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 by invoking the *certification* mechanism (line 11).

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 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

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);
```

$evidence[c_j][p_k] = \langle h(v), s_{p_k}(h(v))\rangle$. 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.

4.2.3 Analysis

The analysis of ERA-NBART has three parts. First, we prove its correctness. Then, we demonstrate that it provides a Nash equilibrium. Finally, we perform a complexity analysis.

4.2.3.1 Correctness

This section provides a proof of the correctness of ERA-NBART, i.e., that it satisfies the properties NBART 1-7. The proof assumes that at most f producers and f consumers are Byzantine and that the rest of the processes follow the algorithm, i.e., are Altruistic. The case of Rational processes is left for Section 4.2.3.2, in which we show that Rational processes also follow the algorithm.

We now show with the following Lemmas that ERA-NBART satisfies each of the NBART properties.

Lemma 4.2.1. *(Validity)* If a non-Byzantine consumer consumes v, then v was produced by some non-Byzantine producer.

Proof. A non-Byzantine consumer c consumes v only if it receives a $hash(v)$ from at least $f+1$ producers and v from at least one producer. There are at most f Byzantine producers, which implies that c receives $hash(v)$ from at least a non-Byzantine producer p_i. Henceforth, c consumes v only if v was input by p_i.

Lemma 4.2.2. *(Integrity)* No non-Byzantine consumer consumes more than once.

Proof. A consumer consumes a value when the *consume* primitive is called. A trivial inspection of Alg. 2 shows that this primitive can be called only once in a non-Byzantine consumer, thus it consumes the value no more than once.

Lemma 4.2.3. *(Agreement)* No two non-Byzantine consumers consume different values.

Proof. By Lemma 4.2.1, if a non-Byzantine consumer consumes v, then v was produced by some non-Byzantine producer. By assumption, every non-Byzantine producer produces the same value. Therefore, non-Byzantine consumers never deliver a value different from v.

Lemma 4.2.4. *(Termination)* Eventually, every non-Byzantine consumer consumes a value.

Proof. All the non-Byzantine producers produce and send v or its hash to all the consumers in the beginning of round 1. Given that channels are reliable and synchronous, all non-Byzantine consumers receive these values in that round. Therefore, when round 2 begins, every non-Byzantine consumer must possess both the correct value and $f+1$ or more hashes of that value, so it executes the *consume* primitive which means consuming v.

These four properties ensure the reliable transfer of the correct value in the presence of Byzantine participants. In the following Lemmas, we prove that the properties NBART 5-7 related to Rational behaviour are fulfilled, therefore ensuring that each process that obeys the protocol is rewarded after the completion of the transfer.

Lemma 4.2.5. *(Evidence)* The certification mechanism eventually produces evidence about the transfer.

Proof. A trivial inspection of the algorithm shows that *certify*(TO, *evidence*) is executed at the end of round 2, which is the same as saying the certification mechanism eventually produces evidence about the transfer.

Lemma 4.2.6. *(Producer Certification)* If producer p is non-Byzantine, then *hasProduced(evidence, p)* is *true.*

Proof. By Lemmas 4.2.3 and 4.2.4, every non-Byzantine consumer delivers the same value v. Before delivering these consumers send their *confirm* vectors to the trusted observer. Therefore, there are at least $N - f$ non-Byzantine consumers $c_k \in \{c_1 \ldots c_{N-f}\}$ that send confirm vectors to the trusted observer at the start of round 2. If producer p_i followed the algorithm, each of these consumers c_k has received *hash*(v) from p_i, and included $\langle h(v), s_{p_i}(h(v))\rangle$ in the message sent to the trusted observer. Since all those messages are included in the evidence generated by the trusted observer, *hasProduced(evidence, p_i)* is true.

Lemma 4.2.7. *(Consumer Certification)* If consumer c is non-Byzantine, then *hasAcknowledged(evidence, c)* is *true.*

Proof. A non-Byzantine consumer sends its *confirm* vector to the trusted observer. Also, since there are at least $N-f$ non-Byzantine producers, consumer c_j includes $\langle h(v), s_{p_i}(h(v))\rangle$ for each of these non-Byzantine producers p_i in the *confirm* vector sent to the trusted observer. According to Lemma 4.2.6, there exists a set *correctset* of at least $N - f$ producers $p_k \in \{p_1 \ldots p_{N-f}\}$ for which *hasProduced(evidence,p_k)* is true (the set of $N - f$ non-Byzantine producers). Therefore, *hasAcknowledged(evidence, c_j)* becomes true for any non-Byzantine consumer c_j.

Theorem 4.2.8. *(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.

Proof. The proof follows directly from Lemmas 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5, 4.2.6, and 4.2.7.

4.2.3.2 Game Theoretical Analysis

To prove that the protocol provides a Nash equilibrium, we model the NBART problem as a strategic game $\Gamma = (M, S_M, \vec{u})$, where $M = \mathcal{P} \bigcup \mathcal{C}$ is the set of players, S_M the set of all possible strategies, and $\vec{u}$ is a vector with the utility functions of all players.

Each player 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 set of possible strategies for player i is denoted S_i. S_M consists on the set of all possible strategies, i.e., $\bigcup_{i \in M} S_i$. The set of all possible strategies of *producers* is S_P. Altruistic producers send $hash(v)$ to all consumers and the value v to the consumers of $consumerset_i$. Rational producers send $hash(v)$ to any subset of $\mathcal{C}$ and the value to any subset $\mathcal{C}' \subseteq \mathcal{C}$. Similarly, $S_{\mathcal{C}}$ denotes the set of all possible strategies that can be followed by *consumers.* 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 from $S_{\mathcal{F}}$, where $\mathcal{F} = \mathcal{F}_{\mathcal{P}} \cup \mathcal{F}_{\mathcal{C}}$ is the set of all Byzantine producers ($\mathcal{F}_{\mathcal{P}}$) and Byzantine consumers ($\mathcal{F}_{\mathcal{C}}$), such that $|\mathcal{F}_{\mathcal{P}}| \leq f$ and $|\mathcal{F}_{\mathcal{C}}| \leq f$. Notice that these are pure strategies, that is, the decision each Rational process makes on which strategy to follow is deterministic.

We now identify the reasons why modelling the NBART problem as a strategic game is not a limitation of our analysis. Strategic games are appropriate for interactions between players where a player cannot form his expectation from the behaviour of the other players on the basis of information about the way that the game was played in the past. The information gathered by each process regarding the past behaviour of other processes is determined by the number of instances of NBART in which those processes interacted, which depends on the mechanism used to form the sets of processes in each instance. This mechanism must ensure that with high probability the number of Byzantine processes of each set is upper bounded by f. Furthermore, in a large peer-to-peer network, it is true that N is much smaller than the total number of processes in the system, and the processes connected to the network during the periods when that mechanism is applied vary from instance to instance. Thus, it is reasonable to assume that processes interact with a very small frequency, which implies that the information of each process regarding the nature of other processes is limited and never certain. Since processes do not incur in risks, they cannot form their expectation from the behaviour of the other players on the basis of information about the way that the game was played in the past. Therefore, it is reasonable to model our solution as a strategic game.

In addition, it is only adequate to model a protocol as a strategic game if players do not change their strategy during the execution of the game, which is true in our protocol. Producers cannot increase their knowledge of the strategies of other players during the execution of the algorithm, as they do not obtain any information from any other participant. Thus, the initial chosen strategy remains adequate for all the rounds of the protocol. On the other hand, each consumer c_j learns about the behaviour of producers during round 1, so it can determine which producers adopted the strategy of sending it the value or its hash. However, according to the definition of *hasAcknowledged*, the *TO* rewards c_j based not only on the information included in the certificate sent by the consumer, but also based on the information the producers (certified by c_j) sent to the remaining consumers. Therefore, the information gathered by c_j is insufficient for the consumer to determine if an alternative strategy provides greater expected profits. For these reasons, it is reasonable to assume that Rational participants do not change their strategy during the execution of the protocol.

We define a *profile of strategies* as the correspondence between players and their respective strategy: $\vec{\sigma}_M : M \mapsto S_M$. By definition, σ_i denotes the strategy followed by player $i \in M$. We define $\vec{\sigma}_M$ as the composition of different profile strategies for disjoint subsets of players $M_1, M_2, ..., M_N$: $\vec{\sigma}_M = (\vec{\sigma}^1_{M_1}, \vec{\sigma}^2_{M_2}, ..., \vec{\sigma}^N_{M_N})$, where $\vec{\sigma}^i_{M_i}$ is the strategy followed by all players of M_i and $M = M_1 \cup M_2 \cup \ldots \cup M_N$.

We also define a *utility function* $u_i(\vec{\sigma}_M) = \beta_i(\vec{\sigma}_M) - \nu_i(\vec{\sigma}_M)$ as the profit that player i obtains when all the players follow the strategy specified by $\vec{\sigma}_M$. β_i denotes the benefit obtained by player i. It is assumed that a producer p gets a benefit of ϕ_P only if *hasProduced(evidence, p)* holds true. Otherwise, the benefit is 0. A consumer c only gets a benefit ϕ_C if it consumes the correct value v (therefore, all non-Byzantine consumers consume the correct value) and if *hasAcknowledged(evidence, c)* holds true. The function $\nu_i(\vec{\sigma}_M)$ maps the costs incurred by player i when every player follows the strategies specified by $\vec{\sigma}_M$. We assume that $\phi_P > \nu_p(\vec{\sigma}_M)$ and $\phi_C > \nu_c(\vec{\sigma}_M)$ for any non-Byzantine producer p and consumer c, respectively. To distinguish the arbitrary behaviour of Byzantine players from the strategies of Altruistic and Rational players, we denote by $\vec{\pi}_{\mathcal{P}} \in \Pi_{\mathcal{P}}$ the profile of strategies of Byzantine producers and by $\vec{\pi}_{\mathcal{C}} \in \Pi_{\mathcal{C}}$ the profile of strategies of Byzantine consumers.

The remaining of this section provides a proof that the protocol provides a Nash equilibrium. In the BAR model, Rational players also take into consideration Altruistic and Byzantine behaviour (Aiyer, Alvisi, Clement, Dahlin, Martin, & Porth 2005). A utility function for Rational player i that considers Byzantine, Altruistic and Rational behaviour, denoted by $\bar{u}_i$, is the expected utility for i if it obeys a given non-Byzantine strategy σ_i when all the remaining participants either obey a non-Byzantine strategy specified by the profile $\vec{\sigma}_M$ (that includes the Altruistic strategy of following the protocol) or follow a Byzantine strategy specified by the profile $\vec{\pi}_{\mathcal{F}}$. Given that Byzantine participants may behave arbitrarily, in the definition of the expected utility function it is necessary to consider not only the expected number of Byzantine players but also the probability of each Byzantine player following each of the possible Byzantine strategies. In this work, we assume that players are risk-averse, therefore the expected utility considers the worst possible scenario of Byzantine behaviour, i.e., it assumes that all Byzantine players adopt a strategy that minimizes the utility of non-Byzantine players (Clement, Napper, Li, Martin, Alvisi, & Dahlin 2007).

Hereupon, we provide a definition for the *expected utility* $\bar{u}_i(\vec{\sigma}_M)$ of player $i \in M$ when all non-Byzantine players follow the strategy specified by $\vec{\sigma}_M$. Let $\vec{\sigma}'_{M\setminus\mathcal{F},\vec{\pi}_\mathcal{P},\vec{\pi}_\mathcal{C}} = (\vec{\sigma}_{M\setminus\mathcal{F}}, \vec{\pi}_\mathcal{P}, \vec{\pi}_\mathcal{C})$ be a profile of strategies where all non-Byzantine players follow the strategy specified by $\vec{\sigma}_{M\setminus\mathcal{F}}$, Byzantine producers follow the strategy specified by $\vec{\pi}_\mathcal{P}$, and Byzantine consumers follow the strategy specified by $\vec{\pi}_\mathcal{C}$. The expected utility of player i is given by the following equation:

$$\bar{u}_i(\vec{\sigma}_M) = \min_{\mathcal{F}_\mathcal{P}:|\mathcal{F}_\mathcal{P}|\leq f, \mathcal{F}_\mathcal{C}:|\mathcal{F}_\mathcal{C}|\leq f} \circ \min_{\vec{\pi}_\mathcal{P}\in\Pi_\mathcal{P}, \vec{\pi}_\mathcal{C}\in\Pi_\mathcal{C}} u_i(\vec{\sigma}'_{M\setminus\mathcal{F},\vec{\pi}_\mathcal{P},\vec{\pi}_\mathcal{C}}) \tag{4.1}$$

Notice the distinction between the expected utility $\bar{u}_i(\vec{\sigma}_M)$, which denotes the minimum utility player i expects to obtain when all non-Byzantine participants follow the strategy specified by $\vec{\sigma}_M$, and the effective utility $u_i(\vec{\sigma}'_{M\setminus\mathcal{F},\vec{\pi}_\mathcal{P},\vec{\pi}_\mathcal{C}})$, which is the difference between the benefits obtained and the costs incurred by i when Byzantine players follow the specific strategies specified by $\vec{\pi}_\mathcal{P}$ and $\vec{\pi}_\mathcal{C}$.

We can now define the functions $\bar{\beta}_i(\vec{\sigma}_M)$ and $\bar{\nu}_i(\vec{\sigma}_M)$ as the expected benefits and costs for the worst possible scenario of Byzantine behaviour. Thus, the expected utility of player $i \in M$ can also be defined as $\bar{u}_i(\vec{\sigma}_M) = \bar{\beta}_i(\vec{\sigma}_M) - \bar{\nu}_i(\vec{\sigma}_M)$.

We now introduce the notion of *Nash equilibrium.* Let $\vec{\sigma}^*_{M\setminus\{i\},\sigma^*_i} = (\vec{\sigma}_{M\setminus\{i\}}, \sigma^*_i)$ denote the profile of strategies where all non-Byzantine players follow the strategy specified by $\vec{\sigma}_{M\setminus\{i\}}$ and player i follows a given strategy σ^*_i. A Nash equilibrium is a profile of strategies for which no player benefits from deviating from its strategy, which can be stated as follows:

Definition. $\vec{\sigma}_M$ is a Nash equilibrium if $\forall_{i \in M} \forall_{\sigma_i^* \in S_i} \bar{u}_i(\vec{\sigma}_M) \geq \bar{u}_i(\vec{\sigma}^*_{M \setminus \{i\}, \sigma_i^*})$.

The following Lemmas provide the complete proof that neither the producers nor the consumers benefit from deviating from the protocol. We use $\vec{\sigma}_{\mathcal{P}}$ and $\vec{\sigma}_{\mathcal{C}}$ to denote the profile of strategies of, respectively, producers and consumers that comply with the protocol. $\vec{\sigma}_M$ denotes the composition of the profiles of strategies $\vec{\sigma}_{\mathcal{P}}$ and $\vec{\sigma}_{\mathcal{C}}$, and $\vec{\sigma}^*_M$ denotes an alternative profile of strategies.

In the next Lemma and Corollary, we show that a producer does not benefit from sending the expected information to less than N consumers and from not sending the value to all consumers of *consumerset*. Then, in Theorem 4.2.11, we show that no producer can increase its utility by deviating from the algorithm, when all consumers follow the expected strategy.

Lemma 4.2.9. For each producer $p \in \mathcal{P}$, for each k such that $0 \leq k < N$, let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P} \setminus \{p\}}, \vec{\sigma}_{\mathcal{C}}, \sigma_p^*)$ be a deviating profile of strategies, where σ_p^* is the strategy of sending messages with the value or its signature to k consumers. Then, $\bar{\beta}_p(\vec{\sigma}^*) = 0$.

Proof. According to the Equation 4.1, Rational players determine their utility considering the worst case scenario of Byzantine and Rational behaviour. Suppose the set of k consumers to which p sends the information includes all the Byzantine players. According to the protocol, the trusted observer only receives vectors containing signed hashes during the second round. Hence, if p only sends the signature of the value or its hash to $k < N$ consumers at the beginning of the first round and if no Byzantine consumer sends their vectors to the trusted observer, the trusted observer only receives $max(k - f, 0) < N - f$ vectors that contain $\langle h(v), s_p(h(v)) \rangle$ during the second round. Therefore, *evidence* will not contain $N - f$ entries with $\langle h(v), s_p(h(v)) \rangle$, *hasProduced(evidence, p)* will hold false, and $\bar{\beta}_p(\vec{\sigma}^*) = 0$.

Corollary 4.2.10. For each producer $p_i \in \mathcal{P}$, for each k such that $0 \leq k < f + 1$, let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P} \setminus \{p_i\}}, \vec{\sigma}_{\mathcal{C}}, \sigma_{p_i}^*)$ be a deviating profile of strategies, where $\sigma_{p_i}^*$ is the strategy of sending the value to k consumers. Then, $\bar{\beta}_{p_i}(\vec{\sigma}^*) = 0$.

Proof. The proof comes trivially from the previous lemma.

Theorem 4.2.11. No producer has any incentives to deviate from the protocol.

Proof. It follows from Lemma 4.2.9 and Corollary 4.2.10 that if the producer p follows an alternative strategy specified by $\vec{\sigma}^*_M$, then $\bar{\beta}_p(\vec{\sigma}^*_M) = 0$, $\bar{u}_p(\vec{\sigma}^*_M) = -\bar{\nu}_p(\vec{\sigma}^*_M)$, and $\bar{u}_p(\vec{\sigma}^*_M) < 0$, for the worst possible scenario. According to the Theorem 4.2.8, $\bar{\beta}_p(\vec{\sigma}_M) = \phi_P$, $u_p(\vec{\sigma}_M) = \phi_P - \bar{\nu}_p(\vec{\sigma}_M)$, and $\bar{u}_p(\vec{\sigma}_M) > 0$, since $\phi_P > \bar{\nu}_p(\vec{\sigma}_M)$. Therefore, $\bar{u}_p(\vec{\sigma}_M) > \bar{u}_p(\vec{\sigma}^*_M)$. Since it is assumed that Rational participants are risk-averse, producers do not have incentives to deviate from the protocol.

We now show that no consumer benefits either by not sending the *confirm* vector to the *TO* or by not processing all the information it receives from the producers. Then, in Theorem 4.2.15, we prove that no consumer can increase its utility by deviating from the algorithm, given that producers follow the expected behaviour.

Lemma 4.2.12. For any consumer $c \in \mathcal{C}$, let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P}}, \vec{\sigma}_{C\setminus\{c\}}, \sigma^*_c)$ be a deviating profile of strategies, where σ^*_c is the strategy of not sending its vector containing hashes sent by producers to the trusted observer during round 2. Then, $\bar{\beta}(\vec{\sigma}^*_M) = 0$.

Proof. The proof derives directly from that fact that, if a consumer c does not send its vector, this information is not included in the evidence and *hasAcknowledged(evidence, c)* holds false. Hence, $\bar{\beta}_c(\vec{\sigma}^*_M) = 0$.

Lemma 4.2.13. For any consumer $c \in \mathcal{C}$, let P_c be the set of producers that sent the correct value or hash to the consumer c, and let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P}}, \vec{\sigma}_{C\setminus\{c\}}, \sigma^*_c)$ be a deviating profile of strategies, where σ^*_c is the strategy of sending an incomplete vector of hashes to the trusted observer with only $f + 1 \leq k < |P_c|$ entries different from the $\perp$ value. Then, $\bar{\beta}_c(\vec{\sigma}^*_M) = 0$.

Proof. The worst possible scenario for a non-Byzantine consumer c_j occurs when $|F_{\mathcal{P}}| = f$ and for all these Byzantine producers *hasProduced* is *false*, while they still send valid information to c_j. In this case, there is only one set *correctset*$_j$, where, for all $p \in$ *correctset*$_j$, *hasProduced(evidence,p)* is true: the set of non-Byzantine producers. If c_j does not set *hashes*$[p_i] = \langle hash(v), s_{p_i}(hash(v)) \rangle$ and p_i is non-Byzantine, then, at the trusted observer, *evidence*$[c_j]$ will not contain the information of at least $N - f$ producers from *correctset*$_j$. Therefore, *hasAcknowledged(evidence,c)* holds *false*, and $\bar{\beta}_c(\vec{\sigma}^*_M) = 0$.

Theorem 4.2.14. No consumer has any incentives to deviate from the protocol.

Proof. It follows from Lemmas 4.2.12 and 4.2.13 that if the consumer c follows an alternative strategy specified by $\vec{\sigma}^*_M$, then $\bar{\beta}_c(\vec{\sigma}^*_M) = 0$, $\bar{u}_c(\vec{\sigma}^*_M) = -\bar{\nu}_c(\vec{\sigma}^*_M)$, and $\bar{u}_c(\vec{\sigma}^*_M) < 0$, for the worst possible scenario. According to the Theorem 4.2.8, $\bar{\beta}_c(\vec{\sigma}_M) = \phi_C$, $\bar{u}_c(\vec{\sigma}_M) = \phi_C - \bar{\nu}_c(\vec{\sigma}_M)$, and $\bar{u}_c(\vec{\sigma}_M) > 0$, since $\phi_C > \bar{\nu}_c(\vec{\sigma}_M)$. Therefore, $\bar{u}_c(\vec{\sigma}_M) > \bar{u}_c(\vec{\sigma}^*_M)$. Since it is assumed that Rational participants are risk-averse, consumers do not have any incentive to deviate from the protocol.

The following Theorem concludes that ERA-NBART provides a Nash equilibrium.

Theorem 4.2.15. (Nash equilibrium) The profile of strategies $\vec{\sigma}_M$ where every player follows the protocol is a Nash equilibrium.

Proof. It follows from Theorems 4.2.11 and 4.2.15 that for every player $i \in M$ and, for all alternative profiles of strategies $\vec{\sigma}^*_M$ where i deviates from the protocol, $\bar{u}_i(\vec{\sigma}^*_M) < \bar{u}_i(\vec{\sigma}_M)$. Hence, $\vec{\sigma}_M$ is a Nash equilibrium.

From the previous proofs, it is possible to observe that our solution is a dominant strategy, that is, any other Nash equilibrium has a utility lower than the utility that each Rational process expects to obtain when following our solution. This follows from the conclusion that if a Rational process omits a certain message or an execution step, in the worst case and in a scenario where all the remaining Altruistic and Rational processes follow the algorithm, then its expected utility is negative. It is clear that if other Rational processes also deviate from the algorithm, then the expected utility never increases: if some Rational producer does not send a message to a consumer c, then c may not provide enough signed hashes to TO and may not be rewarded; if a non-Byzantine consumer fails to process and store some hash from a producer p, then p may never be rewarded. Hence, if there is another Nash equilibrium, it must have an expected utility lower than the provided by our algorithm. In fact, since for any profile of strategies where some Rational process does not follow our algorithm, the expected utility of each non-Byzantine process i that performs some operations is negative, and therefore i can increase its utility by not performing any operations. Thus, the only alternative profile of strategies that is a Nash equilibrium is the one where no process performs any operation. The expected utility of that profile of strategies is 0, allowing us to conclude that our algorithm also provides a dominant strategy and Rational processes should follow it.

4.2.3.3 Complexity Analysis

This section evaluates the algorithm in terms of the following parameters:

Time complexity (TC): Total number of rounds.

Message complexity (MC): Total number of messages sent by non-Byzantine processes. 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 processes. 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.

It is easy to see that the total number of rounds for termination in this case is constant and is equal to 4 rounds. For the remaining complexity measures, we distinguish between arbitrary Byzantine behaviour and crash-fault Byzantine failures, by assuming that there are a_P producers and a_C consumers whose behaviour is arbitrary, that is, maximizes the complexity of the algorithm; and that up to t_P producers and t_C consumers fail by crash before the beginning of the transfer and do not execute any operation. We consider that all a_P Byzantine producers follow the worst possible behaviour which is to send correctly signed messages to all consumers, forcing them to verify all the signatures and sending the signed information to TO. Also, all a_C Byzantine consumers force TO to verify their signatures in the CERTIFICATE messages. The actual numbers of Byzantine producers and consumers are given by $b_P = a_P + t_P$, $b_C = a_C + t_C$, $0 \leq b_P \leq f$ and $0 \leq b_C \leq f$. We assume that messages are transmitted in data packets of maximum size mtu excluding headers.

Hereupon, it is possible to define the parameters for the complexity analysis. We use the letter l to denote bit lengths and the letter k to denote processing, packet transmission, and storage costs. The BC parameters are listed in Table 4.1. In Table 4.2, we list the parameters for the analysis of the PC, SC, and PTC.

Parameter	
Value	l_v
Hash	l_h
Signature	l_s
Data block header	l_m

Table 4.1: Bit complexity parameters.

Parameter	
Producing the value	k_v
Computing an hash	k_h
Computing or verifying a signature	k_s
Sending or receiving a data block	k_m
Storing the value per round	k_{sv}
Storing an hash and a signature per round	k_{sh}

Table 4.2: Processing, storage, and packet transmission complexity parameters.

The MC is $(N - b_P)N + N - b_C$. The justification is the following. Each non-Byzantine producer sends $f + 1$ VALUE messages and $N - f - 1$ SUMMARY messages, which gives a total of $(N - b_P)N$ messages. Each non-Byzantine consumer sends a single message to TO, resulting in a total of $N - b_C$ messages.

The BC is presented in Table 4.3, excluding headers. Each row contains the bit length of each of the messages VALUE, SUMMARY, and CERTIFICATE, the bit complexity per non-Byzantine process regarding the total number of messages of a given type, and the bit complexity of all messages sent by non-Byzantine processes.

Message	**Bit length per message**	**Bit length per process**	**Bit complexity**
VALUE	$l_v + l_h + 2l_s$	$(f + 1)(l_v + l_h + 2l_s)$	$(N - b_P)(f + 1)(l_v + l_h + 2l_s)$
SUMMARY	$l_h + 2l_s$	$(N - f - 1)(l_h + 2l_s)$	$(N - b_P)(N - f - 1)(l_h + 2l_s)$
CERTIFICATE	$(N - t_P)(l_h + l_s) + 2l_s$	$(N - t_P)(l_h + l_s) + 2l_s$	$(N - b_C)(N - t_P)(l_h + l_s) + 2(N - b_C)l_s$

Table 4.3: ERA-NBART: bit complexity excluding headers.

Notice that if n_b is the effective bit complexity (bit complexity excluding the headers), then the total bit complexity is given by $n_b + \frac{n_b}{mtu} \times l_m = n_b(1 + \frac{l_m}{mtu})$. Hence, the total bit complexity is equal to the effective bit complexity times an overhead factor $\alpha_m = 1 + \frac{l_m}{mtu}$, resulting in the following bit complexity:

$$\alpha_m((N - b_P)(f + 1)l_v + (N - b_P)N(l_h + 2l_s) + (N - b_C)(N - t_P)(l_h + l_s) + 2(N - b_C)l_s)$$

The PC, PTC, and SC of producers, consumers and TO are presented in Tables 4.4, 4.5, and 4.6, respectively. We use the parameters $b_{ps} = a_{ps} + t_{ps}$ to represent the number of Byzantine producers that belong to any $producerset_j$ for which the complexity is maximum. Also, there is a constant overhead factor $1 + \frac{k_m}{mtu}$ derived from receiving or sending data block headers, which we denote by α'_m.

Measure	**Operation**	**Cost**
PC	Produce a value	k_v
	Generate hash and three signatures of the hash and messages	$3k_s + k_h$
	Total	$k_v + 3k_s + k_h$
PTC	Send data blocks with value, hash, and signature	$\alpha'_m((f+1)l_v + N(l_h + 2l_s))$
	Total	$\alpha'_m((f+1)l_v + N(l_h + 2l_s))$
SC	Store the value (1 round)	k_{sv}
	Store the hash and signature (1 round)	k_{sh}
	Total	$k_{sv} + k_{sh}$

Table 4.4: ERA-NBART: producer processing, packet transmission, and storage complexity.

4.3 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.

4.3.1 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

Measure	Operation	Cost
PC	Process VALUE message	$(f+1-t_{ps})(2k_s+k_h)$
	Process SUMMARY messages	$(N-f-1-t_P+t_{ps})2k_s$
	Send CERTIFICATE message	$2k_s$
	Total	$(N-t_P+1)2k_s+(f+1-t_{ps})k_h$
PTC	Receive VALUE messages	$\alpha'_m(f+1-b_{ps})(l_v+2l_s+l_h)$
	Receive SUMMARY messages	$\alpha'_m(N-f-1-b_P+b_{ps})(2l_s+l_h)$
	Send CERTIFICATE message	$\alpha'_m(N-t_P)(l_h+l_s)$
	Total	$\alpha'_m((f+1-b_{ps})l_v+(N-b_P)(2l_s+l_h)+$ $(N-t_P)(l_h+l_s))$
SC	Store up to $a_{ps}+1$ different values (1 round)	$(a_{ps}+1)k_{sv}$
	Store $N-t_P$ signed hashes (1 round)	$(N-t_P)k_{sh}$
	Total	$(a_{ps}+1)k_{sv}+(N-t_P)k_{sh}$

Table 4.5: ERA-NBART: consumer processing, packet transmission, and storage complexity.

Measure	Operation	Cost
PC	Process CERTIFICATE messages	$(N-t_C)(N-t_P+2)k_s$
PTC	Receive CERTIFICATE messages	$\alpha'_m(N-t_C)((N-t_P)(l_h+l_s)+2l_s)$
SC	Store hashes and signatures (1 round)	$(N-t_P)(N-t_C)k_{sh}$

Table 4.6: ERA-NBART: TO processing, packet transmission, and storage complexity.

$\perp$ (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. The definitions of these data structures are the following: $\mathit{producerseq}_j = [r \to p_i | r \in [0\dots f], i = j + r \ mod \ N]$; $\mathit{consumerseq}_i = [r \to c_j | r \in [0\dots f], \mathit{producerseq}_j[r] = p_i]$.

Algorithm 4: LRA-NBART (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 ($p_i$, SUMMARY || myhash || myhashsig);
12   forall $c_j \in C$ do
13     send ($p_i$, $c_j$, [SUMMARY, myhash, myhashsig, msgsig])

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

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

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 and after certifying each participant (Alg. 6, lines 6-11). 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 complexity of transferring the value in the worst scenario is $O(N+f^2)$ while in ERA-NBART is always $O(Nf)$, as we will see in Section 4.4.

Algorithm 5: LRA-NBART (consumer c_j)

```
01 upon init do
02   myhash :=⊥;
03   myvalue :=⊥;
04   hashes := $[\perp]^P$;
05   targets := $producerseq_j$;
06   source :=⊥;
07   round := 0;

08 upon nextRound ∧ round = 0 do // start of round 1
09   round := 1;

10 upon deliver ($p_i$, $c_j$, [SUMMARY, phash, phashsig, msgsig]) ∧ round = 1 do
11   if verifysig($p_i$, SUMMARY || phash || phashsig, msgsig)then
12     if verifysig ($p_i$, phash, phashsig) then
13       hashes[$p_i$] := ⟨phash, phashsig⟩;

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

21 upon deliver ($p_i$, $c_j$, [VALUE, pvalue, msgsig]) ∧ round ≥ 2 ∧ round ≤ $f + 2$ do
22   if $p_i$ = source then
23     if verifysig ($p_i$, VALUE || pvalue, msgsig) then
24       if hash(pvalue) = myhash then
25         myvalue := pvalue;

26 upon nextRound ∧ round ≥ 2 ∧ round < $f + 2$ do
27   round := round + 1;
28   if myvalue = ⊥ then
29     hashes[source] := ⊥;
30     source := removefirst(targets);
31     if hashes[source] = $\langle myhash, *\rangle$ then
32       msgsig := sign ($c_j$,REQUEST || myhash);
33       send($c_j$, source, [REQUEST, myhash, msgsig])

34 upon nextRound ∧ round = $f + 2$ do // start of round $f + 3$
35   round := round + 1;
36   confsig := sign ($c_j$, hashes);
37   msgsig := sign ($c_j$, CERTIFICATE ||hashes||confsig);
38   send ($c_j$, TO, [CERTIFICATE, hashes, confsig, msgsig])
39   consume ($c_j$, myvalue);
```

Algorithm 6: LRA-NBART (trusted observer TO)

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

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

06 upon deliver ($c_j$, $TO$, [CERTIFICATE, confirm, confsig, msgsig]) $\wedge$ round $= f+3$ 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 $= f+3$ do // start of round $f+4$
11   certify ($TO$, evidence);
```

4.3.2 Analysis

We now provide the proofs of correctness for LRA-NBART. We use the same notation used in the analysis of ERA-NBART. We also make the same assumptions considering the synchronization of rounds, the correct behaviour of non-Byzantine processes and the property of non-collision of hash functions.

4.3.2.1 Correctness

In the following lemmas, we prove that each non-Byzantine consumer is able to obtain at least $f+1$ identical hashes of the correct value and the value itself.

Lemma 4.3.1. At the beginning of round 2, every non-Byzantine consumer possesses at least $f+1$ hashes of the correct value v.

Proof. According to the protocol, every non-Byzantine producer produces v until the end of the round 0. By assumption, after the first *nextround* event, they immediately send $hash(v)$ to all consumers. Since there are at most f Byzantine producers, it is guaranteed that during round 1 every non-Byzantine consumer receives at least $f+1$ hashes of v as $N \geq 2f+1$, before the beginning of round 2.

Lemma 4.3.2. Every non-Byzantine consumer obtains the value v until the beginning of round $f + 3$.

Proof. According to Lemma 4.3.1, every non-Byzantine consumer c obtains $hash(v)$ until the beginning of round 2. Then, for each subsequent round up to $f + 2$, it requests the value to a different producer until it obtains it. Since each consumer can ask for the value to $f + 1$ producers, then there exists some round $r, 2 \leq r \leq f + 2$ for which $source \notin \mathcal{F}_P$. By assumption, *source* sends v to c no later than the end of round r. Since $r \leq f + 2$, every non-Byzantine consumer obtains the value v until the beginning of round $f + 3$.

We now prove that LRA-NBART fulfils each of the NBART properties.

Lemma 4.3.3. *(Validity)* If a non-Byzantine consumer consumes v, then v was produced by some non-Byzantine producer.

Proof. A non-Byzantine consumer c consumes a value v only if it has received a hash vouching for v from at least $f + 1$ producers and v from one producer. By Lemmas 4.3.1 and 4.3.2, any consumer c obtains at least $f + 1$ hashes of v and the value itself. Since no non-Byzantine producer sends any value other than v and since there are at most f Byzantine producers, then it is impossible for a non-Byzantine consumer to receive $f + 1$ hashes vouching for a value different from the value produced by all non-Byzantine producers. Therefore, v must have been produced by some non-Byzantine producer.

Lemma 4.3.4. *(Integrity)* No non-Byzantine consumer consumes more than once.

Proof. Identical to the proof of Lemma 4.2.2.

Lemma 4.3.5. *(Agreement)* No two non-Byzantine consumers consume different values.

Proof. Identical to the proof of Lemma 4.2.3.

Lemma 4.3.6. *(Termination)* Eventually, every non-Byzantine consumer consumes a value.

Proof. By Lemma 4.3.1, all the non-Byzantine consumers obtain at least $f + 1$ signed hashes of a value v until the beginning of round 2. By Lemma 4.3.2, all non-Byzantine consumers receive v before the beginning of round $f+3$. Therefore, when the *nextround* event is triggered initiating round $f+3$, every non-Byzantine consumer must possess both v and at least $f+1$ hashes of that value, eventually consuming v.

Lemma 4.3.7. *(Evidence)* The *certification* mechanism eventually produces evidence about the transfer.

Proof. Identical to the proof of Lemma 4.2.5.

Lemma 4.3.8. *(Producer Certification)* If producer p_i is non-Byzantine, then *hasProduced(evidence, p_i)* is *true.*

Proof. By the Lemma 4.3.6, every non-Byzantine consumer delivers the same value v. Before delivering, these consumers send their *hashes* vectors to the trusted observer. Therefore, there are at least $N-f$ non-Byzantine consumers $c_k \in \mathcal{C}$ that send hash vectors to the trusted observer at the start of round $f+3$. If producer p_i has complied with the algorithm, each of these consumers c_k has received $h(v)$ from p_i, and included $\langle h(v), s_{p_i}(h(v))\rangle$ in the information sent to the trusted observer. Since all those messages are included in the evidence generated by the trusted observer, *hasProduced(evidence,p_i)* is true.

Lemma 4.3.9. *(Consumer Certification)* If consumer c is non-Byzantine, then *hasAcknowledged(evidence, c)* is *true.*

Proof. Identical to the proof of Lemma 4.2.7.

Theorem 4.3.10. *(Correctness)* If all non-Byzantine participants obey the algorithm, then LRA-NBART solves the NBART problem defined in terms of properties NBART 1-7.

Proof. Follows directly from Lemmas 4.3.3, 4.3.4, 4.3.5, 4.3.6, 4.3.7, 4.3.8, and 4.3.9.

4.3.2.2 Game Theoretical Analysis

To prove that it is in every Rational player best interest to follow the behaviour specified in LRA-NBART, we use the same notation as in Section 4.2.3 and the same definition of the utility function for a strategic game.

In the following Lemmas, we show that the variant is a Nash equilibrium for risk-averse players. In fact, these proofs are almost identical to the ones of 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 *consumerseq*.

Lemma 4.3.11. For each producer $p \in \mathcal{P}$, let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P}\setminus\{p\}}, \vec{\sigma}_{\mathcal{C}}, \sigma^*_p)$ be a deviating profile of strategies, where σ^*_p is the strategy of only sending the signature of the value to every consumer $c \in \mathcal{C}_p$, for any set $\mathcal{C}_p \subset \mathcal{C}$. Then, $\bar{\beta}_p(\vec{\sigma}^*_M) = 0$.

Proof. According to the definition of $\bar{u}$, Rational players determine their utility considering the worst case scenario of Byzantine behaviour. Suppose $\mathcal{F}_{\mathcal{C}} \subseteq \mathcal{C}_p$, for $\#(\mathcal{F}_{\mathcal{C}}) = min(f, \#\mathcal{C}_p)$. According to the protocol, the trusted observer only receives CERTIFICATE messages during the round $f+3$. Hence, if p only sends the signature of the value or its hash to $k < 2f+1$ consumers at the beginning of the round 1 and if no Byzantine consumer sends a confirmation to the trusted observer, the trusted observer only receives $k-f < N-f$ confirmations with $\langle h(v), s_p(h(v))\rangle$ during the round $f+3$. Therefore, the evidence will not contain $N-f$ entries with $\langle h(v), s_p(h(v))\rangle$, *hasProduced(evidence, p)* is false, and therefore $\bar{\beta}_p(\vec{\sigma}^*_M) = 0$.

Lemma 4.3.12. For each producer $p_i \in P$, for some $r, 2 \leq r \leq f+2$, let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P}\setminus\{p_i\}}, \vec{\sigma}_{\mathcal{C}}, \sigma^*_{p_i})$ be a deviating profile of strategies, where $\sigma^*_{p_i}$ is the strategy of not replying to a request from the consumer $c_j = consumerseq_i$. Then, $\bar{\beta}_{p_i}(\vec{\sigma}^*_M) = 0$.

Proof. According to the definition of $\bar{u}$, Rational players determine their utility considering the worst case scenario of Byzantine behaviour. Suppose $\mathcal{F}_{\mathcal{C}} \subseteq consumerseq_i$ and $\#\mathcal{F}_{\mathcal{C}} = f$. If a non-Byzantine consumer c_j sends a request during round r to $p_i = producerseq_j[r-2]$ and p_i refuses it, then c_j will set $confirm[p_i] = \perp$ since, at the beginning of the round $r+1$, c_j will observe that $myvalue = \perp$. Hence, the evidence will not contain $N-f$ entries with $\langle h(v), s_{p_i}(h(v))\rangle$, *hasProduced(evidence, p_i)* is false, and therefore $\bar{\beta}_{p_i}(\vec{\sigma}^*_M) = 0$.

Lemma 4.3.13. For each producer $p_i \in P$, for some $r, 2 \leq r \leq f+2$, let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P}\setminus\{p_i\}}, \vec{\sigma}_{\mathcal{C}}, \sigma^*_{p_i})$ be a deviating profile of strategies, where $\sigma^*_{p_i}$ is the strategy of replying to a request from a consumer c_j such that $c_j \neq consumerseq_i[r-2]$. Then, $\bar{\nu}_{p_i}(\vec{\sigma}^*_M) > \bar{\nu}_{p_i}(\vec{\sigma}_M)$.

Proof. If a producer p_i follows $\sigma^*_{p_i}$ by not rejecting a request from a consumer c_j such that $p_i \neq consumerseq_j[r-2]$, it will incur the cost of transferring the value to c_j. According to Lemma 4.3.12, it cannot reject any further requests, hence the expected number of accepted requests is the same in both strategies aside from the additional accepted request in $\vec{\sigma}^*_M$, implying that $\bar{\nu}_{p_i}(\vec{\sigma}^*_M) > \bar{\nu}_{p_i}(\vec{\sigma}_M)$.

Theorem 4.3.14. No producer has any incentives to unilaterally deviate from the protocol.

Proof. It follows from Lemmas 4.3.11 and 4.3.12 that if the producer p follows an alternative strategy $\vec{\sigma}^*_M$ where it does not send the hash to all the consumers or the value when a consumer requests it during the expected round, then $\vec{\beta}_p(\vec{\sigma}^*_M) = 0$, $\bar{u}_p(\vec{\sigma}^*_M) = -\bar{\nu}_p(\vec{\sigma}^*_M)$, and $\bar{u}_p(\vec{\sigma}^*_M) < 0$, for the worst possible scenario. According to the Theorem 4.3.10, $\vec{\beta}_p(\vec{\sigma}_M) = \beta_P$, $\bar{u}_p(\vec{\sigma}_M) = \beta_P - \vec{\nu}_p(\vec{\sigma}_M)$, and $\bar{u}_p(\vec{\sigma}_M) > 0$, since β_P is greater than the incurred costs. Therefore, $\bar{u}_p(\vec{\sigma}_M) > \bar{u}_p(\vec{\sigma}^*_M)$. Furthermore, if p only deviate by accepting requests during non-expected rounds, then by Lemma 4.3.13 $\vec{\nu}_p(\vec{\sigma}^*_M) > \vec{\nu}_p(\vec{\sigma}_M)$. Since by construction $\vec{\beta}_p(\vec{\sigma}^*_M) = \vec{\beta}_p(\vec{\sigma}_M) = \beta_P$, then $\bar{u}_p(\vec{\sigma}_M) > \bar{u}_p(\vec{\sigma}^*_M)$. Therefore, risk-averse producers do not have incentives to deviate from the protocol.

Lemma 4.3.15. For any consumer $c \in \mathcal{C}$, let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P}}, \vec{\sigma}_{\mathcal{C}\setminus\{c\}}, \sigma^*_c)$ be a deviating profile of strategies, where σ^*_c is the strategy of not sending the *confirm* vector to the trusted observer. Then, $\vec{\beta}(\vec{\sigma}^*_M) = 0$.

Proof. Identical to the proof of Lemma 4.2.12.

Lemma 4.3.16. For each consumer $c_j \in \mathcal{C}$, let p_i be the producer such that $p_i =$ *consumerseq*$_j[r-2]$ for round $r, 2 \leq r \leq f+2$, and let $\vec{\sigma}^*_M = (\vec{\sigma}_{\mathcal{P}}, \vec{\sigma}_{\mathcal{C}\setminus\{c_j\}}, \sigma^*_{c_j})$ be an alternative profile of strategies, where $\sigma^*_{c_j}$ is the strategy of sending the REQUEST message to some producer $p_k \neq p_i$ during round r. Then, $\bar{\nu}_{c_j}(\vec{\sigma}^*_M) > \bar{\nu}_{c_j}(\vec{\sigma}_M)$.

Proof. The algorithm stipulates that during round r, each producer p_i only accepts requests from consumer c_j if $c_j =$ *consumerseq*$_i[r-2]$. Therefore, if c_j sends a REQUEST to a producer p_i such that $p_i \neq$ *producerseq*$_j[r-2]$, it will have its request rejected unless p_i is Byzantine and c_j colludes with it, which does not happen by the assumption that Rational players do not collude with other players. Therefore, to obtain the value, c_j will have to send the same number of additional requests until it obtains the value than it would had to send following the algorithm, in order to guarantee that it is rewarded. Since c must send at least one more request message following σ^*_c than following σ_c, it is true that $\bar{\nu}_{c_j}(\vec{\sigma}^*_M) > \bar{\nu}_{c_j}(\vec{\sigma}_M)$.

Lemma 4.3.17. For any consumer $c \in \mathcal{C}$, let $\mathcal{P}_c$ be the set of producers that sent the correct value or hash to the consumer c, and let $\vec{\sigma}^* = (\vec{\sigma}_{\mathcal{P}}, \vec{\sigma}_{\mathcal{C}\setminus\{c\}}, \sigma_c^*)$ be a deviating profile of strategies, where σ_c^* is the strategy of sending an incomplete *confirm* vector to the trusted observer with only $f + 1 \leq k < \#\mathcal{P}_p$ entries different from the $\perp$ value. Then, $\bar{\beta}_c(\bar{\sigma}_M^*) = 0$.

Proof. Identical to the proof of Lemma 4.2.13.

Theorem 4.3.18. No consumer has any incentives to unilaterally deviate from the protocol.

Proof. It follows from Lemmas 4.3.15, 4.3.16, and 4.3.17, that if a consumer follows any alternative strategy specified by the profile $\vec{\sigma}^*$, then $\bar{\beta}_c(\vec{\sigma}_M^*) = 0$ or $\bar{\nu}_{c_j}(\vec{\sigma}_M^*) > \bar{\nu}_{c_j}(\vec{\sigma}_M)$. In the former case, $\bar{u}_c(\vec{\sigma}_M^*) = -\bar{\nu}_c(\vec{\sigma}_M^*)$, and $\bar{u}_c(\vec{\sigma}_M^*) < 0$, for the worst possible scenario. According to the Theorem 4.3.10, $\bar{\beta}_c(\vec{\sigma}_M) = \phi_C$, $\bar{u}_c(\vec{\sigma}_M) = \phi_C - \bar{\nu}_c(\vec{\sigma}_M)$, and $\bar{u}_c(\vec{\sigma}_M) > 0$, since $\phi_C > \bar{\nu}_c(\vec{\sigma}_M)$. In the later case, since by construction $\bar{\beta}_{c_j}(\vec{\sigma}_M^*) = \bar{\beta}_{c_j}(\vec{\sigma}_M) = \phi_C$ at best, then $\bar{u}_{c_j}(\vec{\sigma}_M) > \bar{u}_{c_j}(\vec{\sigma}_M^*)$. Therefore, risk-averse consumers do not have any incentive to deviate from the protocol.

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

Theorem 4.3.19. (Nash equilibrium) The profile of strategies $\vec{\sigma}_M$ where every player follows the protocol is a Nash equilibrium.

Proof. It follows from Theorems 4.3.14 and 4.3.18 that for every player $i \in M$ and, for all alternative profiles of strategies $\vec{\sigma}_M^*$ where only i deviates from the protocol, $\bar{u}_i(\vec{\sigma}_M^*) < \bar{u}_i(\vec{\sigma}_M)$. Hence, $\vec{\sigma}_M$ is a Nash equilibrium.

Notice that LRA-NBART is also a dominant strategy, since the only alternative Nash equilibrium is for processes to not execute any operation, which results in a utility of 0. Therefore, Rational players should follow this algorithm.

4.3.2.3 Complexity

We now perform an evaluation of the LRA-NBART algorithm, in terms of the same complexity measures used to evaluate ERA-NBART. We use the same notation as in the complexity analysis of ERA-NBART, with the addition of a constant l_r that denotes the bit length of a

REQUEST message. It is clear from the definition of the algorithm that the time complexity is linear to f, more precisely, $f+5$ rounds.

In the following paragraphs, we discuss in detail the number of messages exchanged between the processes during the execution of the algorithm. Each non-Byzantine producer sends N SUMMARY messages to all consumers, which gives a total of $(N-b_P)N$ messages. In addition, each non-Byzantine producer sends a VALUE message in reply to each REQUEST message. The worst possible scenario is when all malicious consumers send valid REQUEST messages in each of the $f+1$ rounds. An upper bound on the total number of VALUE messages is given by $N-b_C+a_C(f+1)-max\{b_P+a_C+f+1-N,0\}\times\frac{1}{2}(b_P+a_C+f+2-N)$:

- Each non-Byzantine consumer must send at least a request to a non-Byzantine producer resulting in $N-b_C$ VALUE messages;
- Malicious Byzantine consumers may send a total of $a_C(f+1)$ requests, resulting in the same number of VALUE messages;
- Since we do not count messages sent by Byzantine processes we must subtract the number of requests made by malicious Byzantine consumers to Byzantine producers, i.e., $max\{b_P+a_C+f+1-N,0\}\times\frac{1}{2}(b_P+a_C+f+2-N)$:
  - If $N\geq b_P+a_C+f+1$, then, in the worst scenario, malicious consumers are not related to Byzantine producers by *producerseq*;
  - If $N<b_P+a_C+f+1$, then there is a malicious consumer that is associated to one Byzantine producer, another is associated to two Byzantine producers, and so on;
  - In this last scenario, the total number of associations among Byzantine processes is given by the sum: $\sum_{i=1}^{b_P+a_C+f+1-N} i=\frac{1}{2}(b_P+a_C+f+2-N)(b_P+a_C+f+1-N)$;
  - Therefore, the total number of associations is determined by $max\{b_P+a_C+f+1-N,0\}\times\frac{1}{2}(b_P+a_C+f+2-N)$.

In addition to the $N-b_C$ requests sent by non-Byzantine consumers to non-Byzantine producers, the worst scenario for the number of REQUEST messages occurs when a non-Byzantine consumer has to request the value to b_P Byzantine producers, other consumer must request it to $b_P - 1$ producers, and so on, which results in $\sum_{i=1}^{b_P} i = (b_P + 1)\frac{b_P}{2}$. Finally, all non-Byzantine consumers send a CERTIFICATE message to TO ($N - b_C$ messages). Therefore, we have $2(N - b_C) + (b_P + 1)\frac{b_P}{2}$ CERTIFICATE and REQUEST messages.

This leads us to conclude that the total number of messages is $(N - b_P)N + 3(N - b_C) + a_C(f + 1) - max\{b_P + a_C + f + 1 - N, 0\} \times \frac{1}{2}(b_P + a_C + f + 2 - N) + (b_P + 1)\frac{b_P}{2}$

The remaining complexity measures are depicted in Tables 4.7, 4.8, 4.9, and 4.10. Regarding the bit complexity, we once again omit the cost of transferring headers, but in this case we do not include the bit length per non-Byzantine process, since the number of bits transmitted varies from process to process. The total bit complexity is given by $\alpha_m((N - b_C + a_C(f+1) - max\{b_P + a_C + f + 1 - N, 0\} \times \frac{1}{2}(b_P + a_C + f + 2 - N))(l_v + l_s) + (N - b_P)N(l_h + 2l_s) + (N - b_C)((N - t_P)(l_h + l_s) + 2l_s) + (N - b_C + (b_P + 1)\frac{b_P}{2})(l_h + l_s))$.

Message	**Bit length per message**	**Bit complexity**
VALUE	$l_v + l_s$	$(N - b_C + a_C(f+1) - max\{b_P + a_C + f + 1 - N, 0\} \times \frac{1}{2}(b_P + a_C + f + 2 - N))(l_v + l_s)$
SUMMARY	$l_h + 2l_s$	$(N - b_P)N(l_h + 2l_s)$
CERTIFICATE	$(N - t_P)(l_h + l_s) + 2l_s$	$(N - b_C)((N - t_P)(l_h + l_s) + 2l_s)$
REQUEST	$l_h + l_s$	$(N - b_C + (b_P + 1)\frac{b_P}{2})(l_h + l_s)$

Table 4.7: LRA-NBART: bit complexity excluding headers.

4.4 ERA-NBART vs LRA-NBART

We will compare the algorithms 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.

Measure	**Operation**	**Cost**
PC	Produce a value	k_v
	Generate three signatures [1]	$3k_s + k_h$
	Process REQUEST messages	$(b_{ps} + 1)k_s$
	Total	$k_v + (b_{ps} + 4)k_s + k_h$
PTC	Send SUMMARY messages	$\alpha'_m N(l_h + l_s)$
	Send VALUE messages	$\alpha'_m(b_{ps} + 1)(l_v + l_s)$
	Receive REQUEST messages	$\alpha'_m(b_{ps} + 1)(l_h + l_s)$
	Total	$\alpha'_m(N(l_h + l_s) + (b_{ps} + 1)(l_v + 2l_s + l_h))$
SC	Store the value ($f + 3$ rounds)	$(f + 3)k_{sv}$
	Store the hash and signature (1 round)	k_{sh}
	Total	$(f + 3)k_{sv} + k_{sh}$

[1] Each producer only has to compute one signature of a VALUE message, that may be used in all replies to REQUEST messages.

Table 4.8: LRA-NBART: producer processing, packet transmission, and storage complexity.

We assume that $l_s \gg l_h$ and $k_s \gg k_h$. Since we assume that $N \geq 2f + 1$, then we use $O(N - f) = O(N)$, and we distinguish between $O(N)$ and $O(f)$ in order to have a generic evaluation for scenarios where $N \gg f$. For the bit and bandwidth complexity, we will analyse the two possibilities: a) $l_v \gg l_s$; and b) $l_v \simeq l_s$. When comparing the storage complexity of both algorithms, our analysis will be based on two scenarios: a) $k_{sv} \gg k_{sh}$; and b) $k_{sv} \simeq k_{sh}$.

Regarding *time complexity*, ERA-NBART is always executed in 4 rounds and LRA-NBART is always executed in $f + 5$ rounds. Also, the message complexity is $O(N^2)$, since each producer has to send its signature to all consumers. In addition, in both algorithms, the PC, PTC, and SC of TO are identical in all three scenarios, that is, $O(k_s N^2)$, $O(l_s N^2)$, and $O(k_{sh} N^2)$, respectively.

Therefore, TO has to process N^2 signatures, due to N signatures of producers sent by each consumer; the cost of receiving the certificates sent by consumers is quadratic to the size of a signature, which is explained by the fact that TO receives from each consumer $O(N)$ signatures of producers; and TO has to store those N^2 hashes.

4.4.1 Bit Complexity

The comparison of the BC is summarized in Table 4.11. We can see that, in the first two scenarios, if $l_v \gg l_s$, then the BC of LRA-NBART ($O(N)$) is significantly lower than the BC of

Measure	Operation	Cost
PC	Process SUMMARY messages	$(N - t_P)2k_s$
	Process VALUE messages	$(a_{ps} + 1)(k_s + k_h)$
	Send REQUEST messages	$(b_{ps} + 1)k_s$
	Send CERTIFICATE message	$2k_s$
	Total	$(2N - 2t_P + a_{ps} + b_{ps} + 4)k_s + (a_{ps} + 1)k_h$
PTC	Receive SUMMARY messages	$\alpha'_m(N - b_P)(2l_s + l_h)$
	Receive VALUE messages	$\alpha'_m(l_v + l_s)$
	Send REQUEST messages	$\alpha'_m(b_{ps} + 1)(l_s + l_h)$
	Send CERTIFICATE message	$\alpha'_m(N - t_P)(l_h + l_s)$
	Total	$\alpha'_m(l_v + (3N - 2b_P - t_P + b_{ps} + 2)l_s + (2N - b_P - t_P + b_{ps} + 1)l_h)$
SC	Store the value	$(f + 1 - b_{ps})k_{sv}$
	Store the hashes and signatures ($f + 2$ rounds)	$(f + 2)(N - t_P)k_{sh}$
	Total	$(f + 1 - b_{ps})k_{sv} + (f + 2)(N - t_P)k_{sh}$

Table 4.9: LRA-NBART: consumer processing, packet transmission, and storage complexity.

Measure	Operation	Cost
PC	Process CERTIFICATE messages	$(N - t_C)(N - t_P + 2)k_s$
PTC	Receive CERTIFICATE messages	$\alpha'_m(N - b_C)((N - t_P)(l_h + l_s) + 2l_s)$
SC	Store hashes and signatures (1 round)	$(N - t_P)(N - t_C)k_{sh}$

Table 4.10: LRA-NBART: TO processing, packet transmission, and storage complexity.

ERA-NBART ($O(Nf)$). However, if $l_v \simeq l_s$, then the BC of both algorithms is $O(N^2)$ in those scenarios. That is because, in LRA-NBART, even if there are Byzantine producers that fail by crash before sending the value to the corresponding consumers, they never sending invalid data, which implies that there is only one transfer of the value per non-Byzantine consumer. Still, in both algorithms, each producer must send N messages with the signature of the value. In the worst possible scenario of arbitrary faults, LRA-NBART has complexity $O(N + f^2)$ if $l_v \gg l_s$, since Byzantine consumers may force one producer to send f VALUE messages, another to send $f - 1$ messages of the same type, and so on, resulting in a total of f^2 messages that contain the value, in addition to the N messages sent to non-Byzantine consumers. Therefore, if $N > f^2$, the BC of LRA-NBART ($O(N)$) is still significantly lower than the BC of ERA-NBART ($O(Nf)$).

On the other hand, if $N \leq f^2$, the BC of LRA-NBART ($O(f^2)$) is of the same order than the BC of ERA-NBART $O(Nf)$, although LRA-NBART is still slightly better.

Scenario	**ERA-NBART**	**LRA-NBART**
Optimistic Scenario	$O(Nfl_v + N^2l_s)$	$O(Nl_v + N^2l_s)$
Crash-Faults Scenario	$O(Nfl_v + N^2l_s)$	$O(Nl_v + N^2l_s)$
Arbitrary-Faults Scenario	$O(Nfl_v + N^2l_s)$	$O((N + f^2)l_v + N^2l_s)$

Table 4.11: Summary of the bit complexity of ERA-NBART and LRA-NBART.

4.4.2 Processing Complexity

The PC of ERA-NBART and LRA-NBART for the three scenarios is depicted in Table 4.12 for producers and consumers.

	Scenario	**ERA-NBART**	**LRA-NBART**
Producers	Optimistic Scenario	$O(k_v + k_s)$	$O(k_v + k_s)$
	Crash-Faults Scenario	$O(k_v + k_s)$	$O(k_v + fk_s)$
	Arbitrary-Faults Scenario	$O(k_v + k_s)$	$O(k_v + fk_s)$
Consumers	Optimistic Scenario	$O(Nk_s)$	$O(Nk_s)$
	Crash-Faults Scenario	$O(Nk_s)$	$O(Nk_s)$
	Arbitrary-Faults Scenario	$O(Nk_s)$	$O(Nk_s)$

Table 4.12: Summary of the processing complexity of ERA-NBART and LRA-NBART.

In the optimistic scenario, the PC of producers in both algorithms is $O(1)$, regardless of the relation between k_v and k_s, which is explained by the fact that producers only have to process one signature in ERA-NBART, and two signatures in LRA-NBART (one for the SUMMARY messages and another for a single VALUE message). If there are Byzantine producers, then in the worst scenario for LRA-NBART, the same producer may receive $f+1$ REQUEST messages, which forces that producer to process $O(f)$ signatures. Therefore, if $k_v \simeq k_s$, the PC of LRA-NBART is $O(f)$, whereas if $k_v \gg k_s$, the PC is $O(1)$ in this algorithm. The PC of producers in ERA-NBART is always $O(1)$.

Regarding consumers, the PC of both algorithms is $O(N)$, since consumers always receive up to N SUMMARY messages from non-Byzantine producers, and this complexity is always asymptotically greater than the complexity of processing up to $f+1$ VALUE messages.

4.4.3 Packet Transmission Complexity

The PTC of the algorithms is summarized in Table 4.13.

	Scenario	**ERA-NBART**	**LRA-NBART**
Producers	Optimistic Scenario	$O(fl_v + Nl_s)$	$O(l_v + Nl_s)$
	Crash-Faults Scenario	$O(fl_v + Nl_s)$	$O(fl_v + Nl_s)$
	Arbitrary-Faults Scenario	$O(fl_v + Nl_s)$	$O(fl_v + Nl_s)$
Consumers	Optimistic Scenario	$O(fl_v + Nl_s)$	$O(l_v + Nl_s)$
	Crash-Faults Scenario	$O(fl_v + Nl_s)$	$O(l_v + Nl_s)$
	Arbitrary-Faults Scenario	$O(fl_v + Nl_s)$	$O(l_v + Nl_s)$

Table 4.13: Summary of the packet transmission complexity of ERA-NBART and LRA-NBART.

As we can see, the PTC of producers is $O(N)$ in both algorithms if $l_v \simeq l_s$, that is, each producer must send N messages with the signature of the value. However, if we consider that $l_v \gg l_s$, then the PTC of LRA-NBART in the optimistic scenario is $O(1)$, in contrast with the PTC in other scenarios or in ERA-NBART, which is $O(f)$. This is due to the fact that in LRA-NBART producers only send the value in reply to a REQUEST message. In the optimistic scenario, each producer only receives one message of that type, whereas in the other scenarios there may be up to $f+1$ consumers which are unable to obtain the value from any other producer and send requests to the same producer, forcing it to transmit $f + 1$ messages with the value. In ERA-NBART, each producer always has to send the value to all consumers of *consumerset*, i.e., to $f + 1$ consumers.

Concerning the PTC of consumers, in ERA-NBART, each consumer may receive $f+1$ VALUE messages from non-Byzantine producers, whereas in LRA-NBART, consumers only receive one message with the value from non-Byzantine producers. Yet, consumers always receive up to N SUMMARY messages. Hence, if $l_v \gg l_s$, then the PTC of LRA-NBART is $O(1)$ and the PTC of ERA-NBART is $O(f)$, while if $l_v \simeq l_s$, then the PTC of both algorithms is $O(N)$.

4.4.4 Storage Complexity

The SC of ERA-NBART and LRA-NBART are depicted in Table 4.14.

The SC of ERA-NBART in producers is always $O(1)$, as producers only have to store the value and its signature during one round. On the other hand, the SC of producers in LRA-

	Scenario	**ERA-NBART**	**LRA-NBART**
Producers	Optimistic Scenario	$O(k_{sv} + k_{sh})$	$O(fk_{sv} + k_{sh})$
	Crash-Faults Scenario	$O(k_{sv} + k_{sh})$	$O(fk_{sv} + k_{sh})$
	Arbitrary-Faults Scenario	$O(k_{sv} + k_{sh})$	$O(fk_{sv} + k_{sh})$
Consumers	Optimistic Scenario	$O(k_{sv} + Nk_{sh})$	$O(fk_{sv} + Nfk_{sh})$
	Crash-Faults Scenario	$O(k_{sv} + Nk_{sh})$	$O(fk_{sv} + Nfk_{sh})$
	Arbitrary-Faults Scenario	$O(fk_{sv} + Nk_{sh})$	$O(fk_{sv} + Nfk_{sh})$

Table 4.14: Summary of the storage complexity of ERA-NBART and LRA-NBART.

NBART is always $O(f)$, since they have to store the value during $f + 2$ rounds.

Regarding consumers, if we consider that $k_{sv} \simeq k_{sh}$, then the SC of consumers in LRA-NBART is always $O(Nf)$, while the SC in LRA-NBART is $O(N)$. This is due to the fact that although in both algorithms each consumer only has to store up to N signatures, in ERA-NBART consumers only store that information until round 2, while in LRA-NBART consumers are forced to store it during $f+2$ rounds. In the other case where $k_{sv} \gg k_{sh}$, the SC of consumers in LRA-NBART is always $O(f)$, as they may obtain the value in round 2 and then store it during $f + 1$ rounds. In ERA-NBART, consumers only store the received values during round 1, and they only receive one different value in the scenarios where faults are non-arbitrary, while in the worst possible scenario they may receive $f + 1$ different values. Therefore, the SC of consumers in ERA-NBART is $O(1)$ in the optimistic and crash-faults scenarios, but it is identical to the SC of LRA-NBART in the arbitrary-faults scenario, that is, $O(f)$.

4.4.5 Conclusions

We showed 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, PTC, and SC are summarized in Tables 4.15, 4.16, 4.17, and 4.18.

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

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 4.15: 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 4.16: Comparison of the processing complexity.

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. 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.

	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 4.17: Comparison of the packet transmission complexity.

4.5 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 from the number of consumers ($N_{\mathcal{C}}$). For instance, in the last transfer, only one non-Byzantine consumer may be necessary. Nevertheless, for any instance of NBART, it is required that $N_{\mathcal{P}} \geq 2f_{\mathcal{P}} + 1$, where $f_{\mathcal{P}}$ 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 reception of the correct value and therefore to certify the behaviour of producers. Hence, $N_{\mathcal{C}} \geq f_{\mathcal{C}} + 1$, where $f_{\mathcal{C}}$ is the upper bound on the number of Byzantine consumers. Though, if a set of consumers is expected to behave as the set of producers of the next instance of NBART, then it is required that $N_{\mathcal{C}} \geq 2f_{\mathcal{C}} + 1$.

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

	Scenario	**Conditions**	**ERA-NBART**	**vs**	**LRA-NBART**
Producers	Optimistic Scenario		$O(1)$	$<$	$O(f)$
	Crash-Faults Scenario		$O(1)$	$<$	$O(f)$
	Arbitrary-Faults Scenario		$O(1)$	$<$	$O(f)$
Consumers	Optimistic Scenario	$k_{sv} \gg k_{sh}$	$O(1)$	$<$	$O(f)$
		$k_{sv} \simeq k_{sh}$	$O(N)$	$<$	$O(Nf)$
	Crash-Faults Scenario	$k_{sv} \gg k_{sh}$	$O(1)$	$<$	$O(f)$
		$k_{sv} \simeq k_{sh}$	$O(N)$	$<$	$O(Nf)$
	Arbitrary-Faults Scenario	$k_{sv} \gg k_{sh}$	$O(f)$	$=$	$O(f)$
		$k_{sv} \simeq k_{sh}$	$O(N)$	$<$	$O(Nf)$

Table 4.18: Comparison of the storage complexity.

among all producers. These restrictions can be stated more precisely as follows:

$$\forall_{c_j \in \mathcal{C}} \#producerset_j = f_P + 1 \tag{4.2}$$

$$\forall_{p_i, p_k \in \mathcal{P}} |\#consumerset_i - \#consumerset_k| \leq 1 \tag{4.3}$$

The following definitions of *producerset* and *consumerset* are correct according to Equation 4.2 and Inequality 4.3:

- $\forall_{c_j \in \mathcal{C}} producerset_j = \{p_i | i \in [s \ldots (s + f_{\mathcal{P}}) \ mod \ N_{\mathcal{P}}], s = j(f_{\mathcal{P}} + 1) \ mod \ N_{\mathcal{P}}\}$
- $\forall_{p_i \in \mathcal{P}} consumerset_i = \{c_j | p_i \in producerset_j\}$

The underlying intuition of the above definition is that each consumer is associated to $f_{\mathcal{P}}+1$ producers whose identifier varies from p_s to $p_{s+f_{\mathcal{P}} \ mod \ N_{\mathcal{P}}}$, i.e., the $f_{\mathcal{P}}+1$ consecutive producers whose identifiers follow p_s. This can be seen as an iterative process, where for each iteration j, a different sequence of $f_{\mathcal{P}}+1$ producers is assigned to the consumer c_j. If during iteration j producer p_i is assigned to the n-th consumer of $consumerset_i$, then all the other producers have been assigned to at least $n-1$ consumers.

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:

$$\forall_{c_j \in \mathcal{C}} \# producerseq_j = f_P + 1 \tag{4.4}$$

$$\forall_{p_i, p_k \in \mathcal{P}} |\# consumerseq_i - \# consumerseq_k| \leq 1 \tag{4.5}$$

$$\forall_{p_i, p_k \in \mathcal{P}} \forall_{r \in [0 \ldots f]} |\# consumerseq_i[r] - \# consumerseq_k[r]| \leq 1 \tag{4.6}$$

We now provide possible definitions of *consumerseq* and *producerseq* that fulfil Equation 4.4, and Inequalities 4.5 and 4.6. Let $d = GCD(f_{\mathcal{P}} + 1, N_{\mathcal{P}})$ be the Greatest Common Divisor and $g = LCM(f_{\mathcal{P}} + 1, N_{\mathcal{P}})$ be the Least Common Multiple of $f_{\mathcal{P}}$ and $N_{\mathcal{P}}$:

- $\forall_{c_j \in \mathcal{C}} producerset_j = [r \rightarrow p_i | r \in [0 \ldots f_{\mathcal{P}}], i = (s + r) \ mod \ N_{\mathcal{P}}, s = (j' + \delta) \ mod \ N_{\mathcal{P}}, j' = j(f_{\mathcal{P}} + 1), \delta = (j' \ div \ d) \ mod \ g]$
- $\forall_{p_i \in \mathcal{P}} consumerseq_i = [r \rightarrow c_j | r \in [0 \ldots f_P], (r, p_i) \in producerseq_j]$

These definitions ensure that all processes have a balanced number of consumers associated to them through *consumerseq* and that each consumer is associated to exactly $f_{\mathcal{P}} + 1$ producers through *producerseq*. Therefore, the LRA-NBART algorithm now operates in $f_{\mathcal{P}} + 5$ rounds. In addition, these definitions guarantee that in any round, if a producer p_i is associated to $n \geq 1$ consumers, then any other producer $p_k \neq p_i$ is associated to at least $n - 1$ consumers.

To better understand these definitions, we introduce an example for $f_{\mathcal{P}} = 2$, $f_{\mathcal{C}} = 1$, $N_{\mathcal{P}} = 6$ and $N_{\mathcal{C}} = 4$, where the data structures *producerset*, *consumerset*, *producerseq*, and *consumerseq* are defined as depicted in Tables 4.19 and 4.20 for ERA-NBART and LRA-NBART, respectively. In Table 4.19, relations among producers are represented using the symbol $\times$, whereas in Table 4.20, those relations are marked with the round number in which each consumer sends a request to the related producer.

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

	c_0	c_1	c_2	c_3	*consumerset*
p_0	$\times$		$\times$		$\{c_0, c_2\}$
p_1	$\times$		$\times$		$\{c_0, c_2\}$
p_2	$\times$		$\times$		$\{c_0, c_2\}$
p_3		$\times$		$\times$	$\{c_1, c_3\}$
p_4		$\times$		$\times$	$\{c_1, c_3\}$
p_5		$\times$		$\times$	$\{c_1, c_3\}$
producerset	$\{p_0, p_1, p_2\}$	$\{p_3, p_4, p_5\}$	$\{p_0, p_1, p_2\}$	$\{p_3, p_4, p_5\}$	

Table 4.19: ERA-NBART: Definition of *producerset* and *consumerset* for $N_{\mathcal{P}} \neq N_{\mathcal{C}}$.

	c_0	c_1	c_2	c_3	*consumerseq*
p_0	0			2	$[c_0, \perp, c_3]$
p_1	1		0		$[c_2, c_0, \perp]$
p_2	2		1		$[\perp, c_2, c_0]$
p_3		0	2		$[c_1, \perp, c_2]$
p_4		1		0	$[c_3, c_1, \perp]$
p_5		2		1	$[\perp, c_3, c_1]$
producerseq	$[p_0, p_1, p_2]$	$[p_3, p_4, p_5]$	$[p_1, p_2, p_3]$	$[p_4, p_5, p_0]$	

Table 4.20: LRA-NBART: Definition of *producerseq* and *consumerseq* for $N_{\mathcal{P}} \neq N_{\mathcal{C}}$.

- *hasProduced(evidence, p_i)* is true if the following condition holds: there are at least $N_{\mathcal{C}} - f_{\mathcal{C}}$ consumers $c_k \in \mathcal{C}$: $evidence[c_k][p_i] = \langle h(v), s_{p_i}(h(v)) \rangle$. 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 $\#correctset_j \geq N_{\mathcal{P}} - f_{\mathcal{P}}$ 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.

In this new scenario, the proofs of correctness of ERA-NBART and LRA-NBART are identical to the proofs provided for the scenario where $N_{\mathcal{P}} = N_{\mathcal{C}}$, as each consumer is associated with $f_{\mathcal{P}}+1$ producers, which ensures that each non-Byzantine consumer obtains the correct value and at least $f_{\mathcal{P}} + 1$ signatures of the hash of that value ($N_{\mathcal{P}} \geq 2f_{\mathcal{P}} + 1$), therefore ensuring NBART 1-4 properties. Also, the existence of $N_{\mathcal{C}} - f_{\mathcal{C}} \geq 1$ non-Byzantine consumers ($N_{\mathcal{C}} \geq f_{\mathcal{C}} + 1$) guarantees that if a producer p_i follows the protocol, then $N_{\mathcal{C}} - f_{\mathcal{C}} > 0$ consumers will certify p_i. In addition, given that $N_{\mathcal{P}} \geq 2f_{\mathcal{P}} + 1$, each consumer obtains the signed hash of the value from $N_{\mathcal{P}} - f_{\mathcal{P}} \geq f_{\mathcal{P}} + 1$ non-Byzantine producers, for which *hasProduced* is true. Hence, the property of *consumer certification* is also ensured.

Furthermore, the principles used to prove that ERA-NBART and LRA-NBART provide a Nash-equilibrium also hold in this new scenario. Each producer p_i cannot omit a VALUE message to none of the consumers of *consumerset*, or else in the worst scenario less than $N_{\mathcal{C}} - f_{\mathcal{C}}$ consumers certify p_i. The same applies for producers that do not send the hash and its signature to all consumers. Risk-averse consumers cannot fail to store any received value as they may not be able to retrieve the correct value from any producer. Also, if a consumer c_j does not store all the received hashes and signatures that it receives, it is possible that less than $N_{\mathcal{P}} - f_{\mathcal{P}}$ hashes received by c_j are from producers for which *hasProduced* is true.

4.6 Summary

This chapter presented two solutions for the NBART problem, namely, ERA-NBART and LRA-NBART. It has been proven that both algorithms fulfil the NBART properties in the absence of Rational behaviour. We also have proven that both algorithms provide a Nash equilibrium and claimed that they are dominant strategies. Therefore, all Rational processes should follow the algorithms. We performed a complexity analysis of both algorithms, which allowed us to conclude that ERA-NBART is more suited for systems were the execution time is more critical, while LRA-NBART provides lower bit complexity at the expense of higher time complexity. Finally, an extension has been proposed for an environment where it may be true that $N_{\mathcal{P}} \neq N_{\mathcal{C}}$.

5 Conclusions and Future Work

5.1 Conclusions

This thesis identified the NBART problem, which is the problem of reliably transferring a value v from a set $\mathcal{P}$ of $N_{\mathcal{P}}$ producers to a set $\mathcal{C}$ of $N_{\mathcal{C}}$ consumers, in the BAR model. NBART was defined by a set of seven properties: four base properties ensure safety and liveness in the presence of Byzantine processes, and the three additional properties guarantee that Rational processes have incentives to follow the solutions to NBART. It has been shown that previous work does not solve this problem in an environment where processes do not interact repeatedly.

Two solutions for the NBART problem were proposed for an environment where Rational processes do not incur in any risks, namely ERA-NBART and LRA-NBART. It was proven that both solutions are correct (fulfil the NBART properties) in the presence of up to f Byzantine processes in each of the sets $\mathcal{P}$ and $\mathcal{C}$, assuming that $N = N_{\mathcal{P}} = N_{\mathcal{C}}$, $N \geq 2f + 1$, the system is synchronous, and communication channels are reliable. Using the concepts of Game Theory, both algorithms were modelled as strategic games and it was proven that they provide a Nash equilibrium. Given that the strategy of following these solutions is also a dominant strategy, that implies that Rational processes should obey our algorithms, therefore preserving the NBART properties. Both algorithms have been compared in terms of time, bit, message, processing, packet transmission, and storage complexity. ERA-NBART has a constant time complexity unlike LRA-NBART, which execution time is linear to f. Regarding message and processing complexity, both algorithms are very similar. The most significant difference occurs for most scenarios in which the bit complexity of LRA-NBART is linear to Nl_v while the bit complexity of ERA-NBART is always Nfl_v, where N is number of processes and l_v is the size of the value. Therefore, ERA-NBART is more suited for systems where time complexity is more relevant than communication costs or the value is small. For any other scenario, LRA-NBART obtains a better performance.

NBART has practical relevance in many distinct areas. We have described a volunteer computing system named BARRAGE and a data backup system named BARCKUP that use NBART. In BARRAGE, Mappers may need to transfer data directly to Reducers, in which case the execution time is more critical. Therefore, ERA-NBART should be used in this context. This algorithm should also be used both in BARRAGE and BARCKUP when the value is not very large. In any other scenario, LRA-NBART should be used. This is particularly true for BARCKUP, since the most important factor in a data backup operation is the availability of the data, not the execution time of the transfer.

5.2 Future Work

The proposed solutions were modelled as strategic games. This is only adequate for scenarios where processes interact only once or if at least the influence of knowledge obtained from previous instances of the game is negligible. One possible direction of future research is to study possible mechanisms of forming the sets of processes in order to introduce some periodicity in the interaction among processes. This could allow a more realistic Game Theoretical analysis by modelling the algorithms as repeated games, for instance. Another interesting direction of future work would be to extend ERA-NBART to an asynchronous environment. Actually, this algorithm is almost asynchronous, since consumers only have to wait for $f + 1$ messages with the signature of the same hash of the value v along with a VALUE message containing v. Then, consumers may immediately consume v and send the signatures to TO. In order to certify all producers, consumers may have to process their signatures and send them to TO, even after consuming the value. A mechanism, such as forcing each consumer to periodically send data with fixed size containing signatures or special null values, could be used to guarantee that consumers certify all producers.

It would also be interesting to find solutions for NBART in an environment where Rational processes are risk-seekers, that is, are willing to incur in risks to maximize their expected utility. At the time of writing this thesis, two alternative strategies were being explored. One provides extra benefits as an incentive for Rational producers to send all the messages specified in ERA-NBART and for Rational consumers to process all the received messages. This requires some modifications on ERA-NBART in order to prevent Rational processes from cheating. The most important modification is to force consumers to prove that they have received the value for each producer they certify. The alternative strategy is to fix the benefits and vary the costs according to the information provided to TO. That is, the TO forces processes to incur in extra costs to balance costs, either by sending penance messages, performing extra computations, or storing special values. Notice however that one of the main challenges of NBART is to provide incentives for processes to transfer the value without using cost balancing. Therefore, it may be feasible to balance the costs of transferring small messages, such as hashes and signatures, but it is infeasible to force producers to transfer a penance message with a cost identical to the one of transferring an arbitrarily large value. Hence, it is not possible to force producers to transfer the value or signed hash to all consumers. Instead, it is sufficient that they transfer the value to the consumers of *consumerset*. Then, consumers exchange the received hashes, allowing them to determine which of the received values is correct.

Bibliography

Abraham, I., D. Dolev, R. Gonen, & J. Halpern (2006, July). Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'06, Denver, CO, USA, pp. 53–62. ACM.

Aiyer, S., L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, & C. Porth (2005, October). BAR fault tolerance for cooperative services. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, SOSP'05, Brighton, United Kingdom, pp. 45–58. ACM.

Alvisi, L., D. Malkhi, E. Pierce, M. Reiter, & R. Wright (2000, June). Dynamic Byzantine quorum systems. In *Proceedings of the 30th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN'00, New York, NY, USA, pp. 283–292. IEEE.

Anderson, D. (2004, November). Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID'04, Pittsburgh, PA, USA, pp. 4–10. IEEE.

Anderson, D., J. Cobb, E. Korpela, M. Lebofsky, & D. Werthimer (2002, November). SETI@home: an experiment in public-resource computing. *Communications of the ACM 45*(11), 56–61.

Asokan, N., B. Baum-waidner, M. Schunter, & M. Waidner (1998, December). Optimistic synchronous multi-party contract signing. Technical Report RZ3089, IBM Research Division.

Asokan, N., M. Schunter, & M. Waidner (1997, April). Optimistic protocols for multi-party fair exchange. In *Proceedings of the 4th ACM Conference on Computer and communications security*, CCS'97, Zurich, Switzerland, pp. 7–17. ACM.

Aumann, Y. & Y. Dombb (2010, October). Pareto efficiency and approximate pareto efficiency in routing and load balancing games. In *Proceedings of the 3rd International Conference on Algorithmic Game Theory*, SAGT'10, Athens, Greece, pp. 66–77. Springer.

Avizienis, A., J.-C. Laprie, B. Randell, & C. Landwehr (2004, January). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable Security Computing 1*(1), 11–33.

Axelrod, R. (1984). *The Evolution of Cooperation.* New York: Basic Books.

Bahreman, A. & J. D. Tygar (1994, February). Certified electronic email. In *Proceedings of the Symposium on Network and Distributed Systems Security*, NDSS'94, San Diego, CA, United States, pp. 3–19. Internet Society.

Baldoni, R., J.-M. Helary, M. Raynal, & L. Tanguy (2003, April). Consensus in Byzantine asynchronous systems. *Journal of Discrete Algorithms 1*(2), 185–210.

Bao, F., R. Deng, K. Q. Nguyen, & V. Varadharajan (1999, August). Multi-party fair exchange with an off-line trusted neutral party. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, DEXA'99, Florence, Italy, pp. 858–. Springer.

Bessani, A., M. Correia, & P. Sousa (2010, October). Active quorum systems. In *Proceedings of the 6th Workshop on Hot Topics*, HotDep'10, Vancouver, Canada, pp. 1–8. USENIX Association.

Birman, K. P., M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, & Y. Minsky (1999, May). Bimodal multicast. *ACM Transactions on Computer Systems 17*(2), 41–88.

Bracha, G. & S. Toueg (1985, October). Asynchronous consensus and broadcast protocols. *Journal of the ACM 32*(4), 824–840.

Castro, M. & B. Liskov (2002, November). Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems 20*(4), 398–461.

Castro, M., R. Rodrigues, & B. Liskov (2003, August). Base: Using abstraction to improve fault tolerance. *ACM Transactions on Computer Systems 21*(3), 236–269.

Chandra, T. D., R. Griesemer, & J. Redstone (2007, August). Paxos made live: an engineering perspective. In *Proceedings of the 26th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'07, Portland, OR, USA, pp. 398–407. ACM.

Chandra, T. D., V. Hadzilacos, & S. Toueg (1996, July). The weakest failure detector for solving consensus. *Journal of ACM 43*(4), 685–722.

Chandra, T. D. & S. Toueg (1990, September). Time and message efficient reliable broadcasts. In *Proceedings of the 4th International Workshop on Distributed Algorithms*, WDAG'90, Bari, Italy, pp. 289–303. Springer.

Chang, J.-M. & N. F. Maxemchuk (1984, August). Reliable broadcast protocols. *ACM Transactions on Computer Systems 2*(3), 251–273.

Chow, M. & R. van Renesse (2010, April). A middleware for gossip protocols. In *Proceedings of the 9th International Conference on Peer-to-peer Systems*, IPTPS'10, San Jose, CA, USA, pp. 8–8. USENIX Association.

Clement, A., M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, & T. Riche (2009, August). Upright cluster services. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, SOSP'09, Big Sky, MT, USA, pp. 277–290. ACM.

Clement, A., H. Li, J. Napper, J.-P. Martin, L. Alvisi, & M. Dahlin (2008, June). BAR primer. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN'08, Anchorage, AK, USA, pp. 287–296. IEEE.

Clement, A., J. Napper, H. Li, J.-P. Martin, L. Alvisi, & M. Dahlin (2007, August). Theory of BAR games. In *Proceedings of the 26th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'07, Portland, OR, USA, pp. 358–359. ACM.

Coffey, T. & P. Saidha (1996, January). Non-repudiation with mandatory proof of receipt. *SIGCOMM Computer Communication Review 26*(1), 6–17.

Cohen, B. (2003, June). Incentives build robustness in bittorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, P2PEcon'03, Berkeley, CA, USA.

Correia, M., N. Neves, L. C. Lung, & P. Veríssimo (2005, March). Low complexity Byzantine-resilient consensus. *Distributed Computing 17*(3), 237–249.

Costa, P., M. Pasin, A. Bessani, & M. Correia (2011, November). Byzantine fault-tolerant MapReduce: Faults are not just crashes. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science*, CloudCom'11, Athens, Greece. IEEE.

Cowling, J., D. Myers, B. Liskov, R. Rodrigues, & L. Shrira (2006, November). Hq replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'06, Seattle, WA, USA, pp. 177–190. USENIX Association.

Cox, L., C. Murray, & B. Noble (2002, December). Pastiche: making backup cheap and easy. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'02, Boston, MA, USA, pp. 285–298. USENIX Association.

Cox, L. & B. D. Noble (2003, October). Samsara: Honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, SOSP'03, Bolton Landing, NY, USA, pp. 120–132. ACM.

Dean, J. & S. Ghemawat (2004, December). MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'04, San Francisco, CA, USA, pp. 107–113. USENIX Association.

Distler, T. & R. Kapitza (2011, April). Increasing performance in Byzantine fault-tolerant systems with on-demand replica consistency. In *Proceedings of the 6th ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys'11, Salzburg, Austria, pp. 91–106. ACM.

Dolev, D., C. Dwork, & L. Stockmeyer (1987, January). On the minimal synchronism needed for distributed consensus. *Journal of ACM 34*(1), 77–97.

Dolev, D. & H. Strong (1983, June). Authenticated algorithms for Byzantine agreement. *SIAM Journal of Computing 12*(4), 656–666.

Dwork, C., N. Lynch, & L. Stockmeyer (1988, April). Consensus in the presence of partial synchrony. *Journal of ACM 35*(2), 288–323.

Eliaz, K. (2002, August). Fault-tolerant implementation. *Review of Economic Studies 69*(3), 589–610.

Elnikety, S., M. Lillibridge, M. Burrows, & W. Zwaenepoel (2002, January). Peer-to-peer cooperative backup system. In *Proceedings of the USENIX Conference on File and Storage Technologies*, FAST'02, Monterey, CA, USA.

Fischer, M. J. (1983, August). The consensus problem in unreliable distributed systems (a brief survey). In *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, FCT'83, Borgholm, Sweden, pp. 127–140. Springer.

Fischer, M. J., N. A. Lynch, & M. S. Paterson (1985, April). Impossibility of distributed consensus with one faulty process. *Journal of ACM 32*(2), 374–382.

Fraigniaud, P. (1992, November). Asymptotically optimal broadcasting and gossiping in faulty hypercube multicomputers. *IEEE Transactions on Computers 41*(11), 1410–1419.

Gafni, E. & L. Lamport (2003, February). Disk paxos. *Distributed Computing 16*(1), 1–20.

Garay, J. A. & P. D. MacKenzie (1999, September). Abuse-free multi-party contract signing. In *Proceedings of the 13th International Symposium on Distributed Computing*, DISC'99, Bratislava, Slovak Republic, pp. 151–165. Springer.

Golding, R. A. & D. D. E. Long (1992, October). Design choices for weak-consistency group communication. Technical Report UCSC–TR–92–45, University of California at Santa Cruz, Santa Cruz, CA, USA.

Guerraoui, R. & L. Rodrigues (2006). *Introduction to Reliable Distributed Programming*. Secaucus, NJ, USA: Springer.

Gupta, R. & A. K. Somani (2004, August). Reputation management framework and its use as currency in large-scale peer-to-peer networks. In *Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing*, P2P'04, Zurich, Switzerland, pp. 124–132. IEEE.

Hadzilacos, V. & S. Toueg (1994, May). A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Department of Computer Science, Ithaca, NY, USA.

Han, Y. (1996, June). Investigation of non-repudiation protocols. In *Proceedings of the 1st Australasian Conference on Information Security and Privacy*, ACISP'96, Wollongong, NSW, Australia, pp. 38–47. Springer.

Hardin, G. (1968, December). The tragedy of the commons. *Science 162*(3859), 1243–47.

Hayrapetyan, A., É. Tardos, & T. Wexler (2006, May). The effect of collusion in congestion games. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, STOC'06, Seattle, WA, USA, pp. 89–98. ACM.

Hughes, D., G. Coulson, & J. Walkerdine (2005, June). Free riding on gnutella revisited: The bell tolls? *IEEE Distributed Systems Online 6*(6), 1–.

Johansen, H., A. Allavena, & R. van Renesse (2006, April). Fireflies: scalable support for intrusion-tolerant network overlays. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys'06, Leuven, Belgium, pp. 3–13. ACM.

Johnsson, S. & C.-T. Ho (1989, September). Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers 38*(9), 1249–1268.

Kaashoek, M. F. & A. S. Tanenbaum (1996, May). An evaluation of the Amoeba group communication system. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, ICDCS'96, pp. 436–. IEEE.

Kaashoek, M. F., A. S. Tanenbaum, & S. F. Hummel (1989, October). An efficient reliable broadcast protocol. *Operating Systems Review 23*(4), 5–19.

Keidar, I., R. Melamed, & A. Orda (2006, July). Equicast: Scalable multicast with selfish users. In *Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'06, Denver, CO, USA, pp. 63–71. ACM.

Khill, I., J. Kim, I. Han, & J. Ryou (2001, May). Multi-party fair exchange protocol using ring architecture model. *Computers & Security 20*(5), 422–439.

Kremer, S. & O. Markowitch (2000a, August). A multi-party non-repudiation protocol. In *Proceedings of the 15th Annual Working Conference on Information Security for Global Information Infrastructures*, IFIP TC11, Bejing, China, pp. 271–280. Springer.

Kremer, S. & O. Markowitch (2000b, May). Optimistic non-repudiable information exchange. In *Proceedings of the 21th Symposium on Information Theory in the Benelux*, WICSP'00, Enschede, The Netherlands, pp. 139–146. Werkgemeenschap Informatie en Communicatietheorie.

Kremer, S., O. Markowitch, & J. Zhou (2002, November). An intensive survey of fair non-repudiation protocols. *Computer Communications 25*(17), 1606–1621.

Lamport, L. (1978, July). Time clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*(7), 558–565.

Lamport, L. (1998, May). The part-time parliament. *ACM Transactions on Computer Systems 16*(2), 133–169.

Lamport, L. (2006a, October). Fast paxos. *Distributed Computing 19*(2), 79–103.

Lamport, L. (2006b, October). Lower bounds for asynchronous consensus. *Distributed Computing 19*(2), 104–125.

Lamport, L. (2011, September). Byzantizing Paxos by refinement. In *Proceedings of the 25th International Symposium on Distributed Computing*, DISC'11, Rome, Italy, pp. 211–224. Springer.

Lamport, L. & M. Massa (2004, June). Cheap paxos. In *Proceedings of the 34th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN'04, Florence, Italy, pp. 307–. IEEE.

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

Lee, S. & K. Shin (1994, May). Interleaved all-to-all reliable broadcast on meshes and hypercubes. *Parallel and Distributed Systems, IEEE Transactions on 5*(5), 449–458.

Li, H., A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, & M. Dahlin (2008, December). Flightpath: Obedience vs choice in cooperative services. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'08, San Diego, CA, USA, pp. 355–368. USENIX Association.

Li, H., A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, & M. Dahlin (2006, November). BAR gossip. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'06, Seattle, WA, USA, pp. 191–204. USENIX Association.

Liang, J., R. Kumar, & K. W. Ross (2005, October). The kazaa overlay: A measurement study. *Computer Networks Journal 49*(6).

Liben-Nowell, D., H. Balakrishnan, & D. Karger (2002, July). Analysis of the evolution of peer-to-peer systems. In *Proceedings of the 21st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'02, Monterey, CA, USA, pp. 233–242. ACM.

Mailath, G. J. (1998, September). Do people play Nash equilibrium? Lessons from evolutionary game theory. *Journal of Economic Literature 36*(3), 1347–1374.

Malkhi, D. & M. Reiter (1997, May). Byzantine quorum systems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC'97, El Paso, TX, USA, pp. 569–578. ACM.

Malkhi, D., M. Reiter, & A. Wool (1997, August). The load and availability of Byzantine quorum systems. In *Proceedings of the 16th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'97, Santa Barbara, CA, United States, pp. 249–257. ACM.

Mari, F., I. Melatti, I. Salvo, L. Alvisi, A. Clement, & H. Li (2008, November). Model checking Nash equilibria in mad distributed systems. In *Proceedings of the 8th Conference on Formal*

Methods in Computer-Aided Design, FMCAD'08, Portland, OR, USA, pp. 1–8. IEEE.

Mari, F., I. Melatti, I. Salvo, E. Tronci, L. Alvisi, A. Clement, & H. Li (2009, November). Model checking coalition Nash equilibria in mad distributed systems. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS'09, Lyon, France, pp. 531–546. Springer.

Markowitch, O. & S. Kremer (2000, December). A multi-party optimistic non-repudiation protocol. In *Proceedings of the 3rd International Conference on Information Security and Cryptology*, ICISC'00, Seoul, Korea, pp. 109–122. ACM.

Markowitch, O. & Y. Roggeman (1999, September). Probabilistic non-repudiation without trusted third party. In *Proceedings of the 2nd Conference on Security in Communication Networks*, SCN'99, Amalfi, Italy, pp. 1–12. Springer.

Markowitch, O. & S. Saeednia (2001, February). Optimistic fair exchange with transparent signature recovery. In *Proceedings of the 5th International Conference on Financial Cryptography*, FC'01, pp. 339–350. Springer.

Martin, J.-P. & L. Alvisi (2006, July). Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing 3*(3), 202–215.

Martin, J.-P., L. Alvisi, & M. Dahlin (2002, October). Minimal Byzantine storage. In *Proceedings of the 16th International Symposium on Distributed Computing*, DISC'02, Toulouse, France, pp. 311–325. Springer.

Martin, O. & R. Ariel (1994). *A Course in Game Theory.* MIT Press.

Mokhtar, S., A. Pace, & V. Quéma (2010, October). FireSpam: Spam resilient gossiping in the BAR model. In *Proceedings of the 29th IEEE International Symposium on Reliable Distributed Systems*, SRDS'10, New Delhi, India, pp. 225–234. IEEE.

Moscibroda, T., S. Schmid, & R. Wattenhofer (2006, July). On the topologies formed by selfish peers. In *Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'06, Denver, CO, USA, pp. 133–142. ACM.

Nash, J. (1951, September). Non-cooperative games. *The Annals of Mathematics 54*(2), 286–295.

Onieva, J. A., J. Zhou, & J. Lopez (2008, December). Multiparty nonrepudiation: A survey. *ACM Computers Survey 41*(1), 5:1–5:43.

Pease, M., R. Shostak, & L. Lamport (1980, April). Reaching agreement in the presence of faults. *Journal of the ACM 27*(2), 228–234.

Plank, J. S. (1997, September). A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Software – Practice & Experience 27*(9), 995–1012.

Rabin, M. O. (1983, October). Transaction protection by beacons. *Journal of Computer and System Sciences 27*(2), 256–267.

Schneider, F. B. (1984, May). Byzantine generals in action: implementing fail-stop processors. *ACM Transactions on Computer Systems 2*(2), 145–154.

Schneider, F. B. (1990, December). Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computers Survey 22*(4), 299–319.

Shneidman, J., D. C. Parkes, & L. Massoulié (2004, August). Faithfulness in internet algorithms. In *Proceedings of the ACM SIGCOMM Workshop on Practice and theory of incentives in networked systems*, PINS'04, Portland, Oregon, USA, pp. 220–227. ACM.

Thambidurai, P. & Y. keun Park (1988, October). Interactive consistency with multiple failure modes. In *Proceedings of the 7th IEEE International Symposium on Reliable Distributed Systems*, SRDS'88, Columbus, OH, USA, pp. 93–100. IEEE.

Veronese, G. S., M. Correia, A. Bessani, & L. C. Lung (2010, November). EBAWA: efficient Byzantine agreement for wide-area networks. In *Proceedings of the 12th IEEE International High Assurance Systems Engineering Symposium*, HASE'10, San Jose, CA, USA, pp. 10–19. IEEE.

Vilaça, X., J. Leitão, M. Correia, & L. Rodrigues (2011, December). N-party BAR transfer. In *Proceedings of the 15th International Conference On Principles Of Distributed Systems (to appear)*, OPODIS'11, Toulouse, France.

Vilaça, X., J. Leitão, & L. Rodrigues (2011a, September). N-party BAR Transfer: Motivation, definition, and challenges. In *Proceedings of the 3rd Workshop on Theoretical Aspects on Dynamic Distributed Systems (in conjunction with DISC 2011)*, TADDS'11, Rome, Italy, pp. 18–22. Springer.

Vilaça, X., J. Leitão, & L. Rodrigues (2011b, September). Transferência de dados entre grupos de processos no modelo BAR. In *Actas do Terceiro Simpósio de Informática*, INForum'11, Coimbra, Portugal.

Walsh, K. & E. G. Sirer (2006, May). Experience with an object reputation system for peer-to-peer file sharing. In *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation*, NSDI'06, San Jose, CA, USA, pp. 1–14. USENIX Association.

Wensley, J., L. Lamport, J. Goldberg, M. Green, K. Levitt, P. Melliar-Smith, R. Shostak, & C. Weinstock (1978, October). Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE 66*(10), 1240–1255.

Wong, E. L., J. B. Leners, & L. Alvisi (2010, September). It's on me! the benefit of altruism in BAR environment. In *Proceedings of the 25th International Symposium on Distributed Computing*, DISC'10, Cambridge, USA, pp. 406–420. Springer.

Zhang, N. & Q. Shi (1996, November). Achieving non-repudiation of receipt. *The Computer Journal 39*(10), 844–853.

Zhou, J., R. H. Deng, & F. Bao (1999, April). Evolution of fair non-repudiation with ttp. In *Proceedings of the 4th Australasian Conference on Information Security and Privacy*, ACISP'99, London, UK, pp. 258–269. Springer.