Byzantine consensus in asynchronous message-passing systems: A survey

Article *in* International Journal of Critical Computer-Based Systems · January 2011

DOI: 10.1504/IJCCBS.2011.041257 · Source: DBLP

4 authors:

Miguel Correia
Instituto Superior Técnico University of Lisbon
327 PUBLICATIONS **8,407** CITATIONS
SEE PROFILE

Giuliana Veronese
Stefanini IT Solutions
17 PUBLICATIONS **820** CITATIONS
SEE PROFILE

Nuno Ferreira Neves
University of Lisbon
192 PUBLICATIONS **4,225** CITATIONS
SEE PROFILE

Paulo Veríssimo
University of Luxembourg
363 PUBLICATIONS **12,611** CITATIONS
SEE PROFILE

Byzantine Consensus in Asynchronous Message-Passing Systems: a Survey

Miguel Correia*, Giuliana Santos Veronese, Nuno Ferreira Neves and Paulo Verissimo

Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749-016 Lisboa, Portugal
E-mail: mpc@di.fc.ul.pt
E-mail: giuliana@lasige.di.fc.ul.pt
E-mail: nuno@di.fc.ul.pt
E-mail: pjv@di.fc.ul.pt
*Corresponding author

Abstract

Consensus is a classical distributed systems problem with both theoretical and practical interest. Asynchronous Byzantine consensus is currently at the core of some solutions for the implementation of highly-resilient computing services. This paper surveys Byzantine consensus in message-passing distributed systems, by presenting the main theoretical results in the area, the main classes of algorithms and by discussing important issues like the performance and resilience of these algorithms.

Keywords: distributed algorithms; distributed systems; consensus; Byzantine faults; arbitrary faults; asynchronous model; message passing; Byzantine consensus

Reference to this paper should be made as follows: Correia, M., Veronese, G. S., Neves, N. F. and Verissimo, P. (xxxx) ‘Byzantine Consensus in Asynchronous Message-Passing Systems: a Survey’, *Int. J. Critical Computer-Based Systems*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Miguel Correia is an assistant professor in the Department of Informatics at the Faculty of Sciences at the University of Lisboa, and an adjunct faculty member of the Carnegie Mellon Information Networking Institute. More information about his research can be found at http://www.di.fc.ul.pt/∼mpc.

Giuliana Santos Veronese is a PhD student in the Department of Informatics at the Faculty of Sciences at the University of Lisboa. She received a MSc degree in Distributed Systems from Pontifícia Universidade Católica do Paraná in 2005.

Nuno Neves received the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 1998. He is an associate professor in the Department of Informatics at the Faculty of Sciences at the University of Lisboa. More information about his research can be found at http://www.di.fc.ul.pt/∼nuno.

Paulo Verissimo is a professor in the Department of Informatics (DI) at the Faculty of Sciences at the University of Lisboa (http://www.di.fc.ul.pt/~pjv), and the director of LASIGE. He is a fellow of the IEEE and the ACM.

1 Introduction

This paper presents a survey on Byzantine consensus algorithms for distributed systems. Consensus is a classical distributed systems problem, first introduced in (Pease et al., 1980), with both theoretical and practical interest (Lynch, 1996; Guerraoui and Schiper, 1997; Guerraoui et al., 2000). The problem can be stated informally as: how to ensure that a set of distributed processes achieve agreement on a value or an action despite a number of faulty processes.

From a theoretical point of view, the relevance of the consensus problem derives from several other distributed systems problems being reducible or equivalent to it. Examples are atomic broadcast (Hadzilacos and Toueg, 1994; Chandra and Toueg, 1996; Correia et al., 2006), non-blocking atomic commit (Guerraoui and Schiper, 2001), group membership (Guerraoui and Schiper, 2001), and state machine replication (Schneider, 1990). The relations between consensus and other distributed problems are important because consensus is a deeply studied problem and many results stated for consensus automatically apply to these other problems.

From a systems perspective, replication of components and services is an important paradigm that can be employed for information protection in critical applications, and consensus plays a fundamental role in many replication algorithms. Therefore, it is essential that one has a good understanding on the tradeoffs and difficulties in the design and implementation of these algorithms. Some example solutions based on these ideas are: Bessani et al. (2008) and Yin et al. (2003) use replication to implement fault- and intrusion-tolerant firewall devices; Cachin et al. (2001), Castro and Liskov (2002), Chun et al. (2007) and Veronese et al. (2009b) propose replication algorithms to implement highly-resilient services, like data historians or DNS. Some of these can be employed for the protection of critical information infrastructures, which are used to control services of a high societal importance, like power, water and gas (Verissimo et al., 2008; Hauser et al., 2008).

Algorithms that solve consensus vary much depending on the assumptions that are made about the system. This paper will consider message-passing algorithms for systems that may experience *Byzantine* (or arbitrary) faults in *asynchronous* settings (i.e., without timing assumptions). Algorithms based on this system model are being used as important building blocks in critical applications, often under designations such as intrusion tolerance and Byzantine fault tolerance. In more detail:

The *message-passing* model is the natural choice for algorithms that are executed by various components spread through the infrastructure, which use messages to exchange information, transmitted through some kind of network (e.g., Ethernet or Wi-Fi LAN). An alternative system model would be shared-memory (Attie, 2002; Friedman et al., 2002; Malkhi et al., 2003; Alon et al., 2005; Bessani et al., 2009), but in the system architecture we are considering the shared memory itself has to be implemented using message-passing algorithms. For shared memory, consensus has been shown to be universal, i.e., to be enough to implement any shared memory object, both with crash (Herlihy, 1991) and Byzantine faults (Malkhi et al., 2003).

Arbitrary faults, usually called *Byzantine* after the paper by Lamport et al. (1982), do not put any constraints on how processes fail. This sort of assumption or, better said, of non-assumption about how processes fail, is specially adequate for systems where malicious attacks and intrusions can occur. For instance, an attacker might modify the behaviour of a process that he/she controls in order to change the outcome of the consensus algorithm, eventually causing the rest of the system to act in an erroneous way. Interestingly, assuming Byzantine faults, instead of the more typical assumption of crash faults, leads to more complex and challenging algorithms.

Asynchrony might also be better described as a non-assumption about timing properties (i.e., there is no need to make assumptions about the processing speeds of nodes and delays on message transmission). This (non-) assumption is important because attackers can often violate some timing properties by launching denial-of-service attacks against processes or communications. For instance, the attacker might delay the communication of a process for an interval, breaking some assumption about the timeliness of the system.

This system model – Byzantine faults and asynchrony – is very generic. Algorithms in this model have to deal with uncertainty on two independent planes: faults and time. This leads to algorithms that besides being tolerant to adverse environments, have the virtue of also running correctly in more benign conditions, like those in which only crash faults occur or in which delay bounds are attained. These algorithms are however bounded by an impossibility result, which says that consensus can not be deterministically solved in an asynchronous system if a single process can crash (often called the Fischer-Lynch-Paterson, FLP, result (Fischer et al., 1985)). This result has led to a large number of works that attempt to circumvent it, i.e., to slightly modify the system model in such a way that consensus becomes solvable. Examples include randomization (Rabin, 1983; Ben-Or, 1983), failure detectors (Chandra and Toueg, 1996; Malkhi and Reiter, 1997), partial-synchrony (Dwork et al., 1988; Dolev et al., 1987), and hybridization/wormholes (Correia et al., 2005; Neves et al., 2005).

The paper is organized as follows. There are several definitions of consensus in the literature, so Section 2 presents those that are widespread and more useful. Section 3 presents the FLP impossibility result, which is a core theoretical result in the area. Section 4 describes several ways of circumventing this result: by sacrificing determinism, adding time to the model, augmenting the system model with an oracle, using hybridization and modifying the problem. Section 5 discusses the performance, scalability and resilience of consensus algorithms. Section 6 addresses the aspect of related and equivalent problems, like atomic broadcast and membership. Section 7 concludes the paper.

2 Byzantine Consensus Definitions

This section defines the consensus problem and several of the variations considered in the literature. We say that a process is *correct* if it follows its algorithm until completion, otherwise it is said to be *faulty*.

The consensus problem is typically defined for a set of n known processes. Another important parameter, often designated with the letter f (or t), is the maximum number of faulty processes. There is a recent trend of research on dynamic systems in which the number of involved processes is unknown (Mostefaoui et al., 2005; Aguilera, 2004; Alchieri et al., 2008), but the first Byzantine consensus algorithms for this system model are still starting to appear (Alchieri et al., 2008).

A *binary consensus* algorithm aims to achieve consensus on a binary value $v \in \{0,1\}$ (or {false, true} or {against, in favour}). Each process proposes its *initial value* (binary) and decides on a value v. The problem can be formally defined in terms of three properties:

- *Validity:* If all correct processes propose the same value v, then any correct process that decides, decides v.
- *Agreement:* No two correct processes decide differently.
- *Termination:* Every correct process eventually decides.

The first two properties are *safety* properties, i.e., properties that say that some "bad thing" cannot happen, while the last is a *liveness* property, i.e., a property that states "good things" that must happen (Alpern and Schneider, 1984).

Multi-valued consensus is apparently similar to binary consensus, except that the set of values has arbitrary size, i.e., $v \in V$ and $|V| > 2$. However, multi-valued consensus algorithms were studied in the literature using different Validity properties, while the Agreement and Termination properties remained essentially the same (with minor exceptions for Termination that we will see later). Some papers use the following Validity property (Dwork et al., 1988; Malkhi and Reiter, 1997; Kihlstrom et al., 2003):

- *Validity 1:* If all correct processes propose the same value v, then any correct process that decides, decides v.

Others use the following (Doudou et al., 1999, 2002; Baldoni et al., 2003):

- *Validity 2:* If a correct process decides v, then v was proposed by some process.

Both properties are somewhat weak. Validity 1 does not say anything about what is decided when the correct processes do not propose all the same v, while Validity 2 does not say anything about what is the value decided (e.g., is it the value proposed by the correct processes if all of them propose the same, or a value proposed by a faulty process?). A definition that gives more detail about what is decided has also been proposed (Correia et al., 2006). The definition has three Validity properties:

- *Validity 1:* If all correct processes propose the same value v, then any correct process that decides, decides v.
- *Validity 2A:* If a correct process decides v, then v was proposed by some process or $v = \perp$.
- *Validity 3:* If a value v is proposed only by faulty processes, then no correct process decides v.

The first two are essentially the Validity properties already introduced, except that Validity 2A allows the protective decision of a value $\perp \notin V$. The third property is inspired by the original definition in the context of the "Byzantine Generals" metaphor used in the classical paper by Lamport et al. (1982). The definition was "(1) All loyal generals decide upon the same plan of action; (2) A small number of traitors cannot cause the loyal generals to adopt a bad plan.". That paper however, considered a synchronous system, i.e., a system in which there are known delay bounds for processing and communication.

This concern about the practical interest of multi-valued consensus defined in terms of Validity 1 or Validity 2 has also led to the definition of *vector consensus* (Doudou et al., 1999). The difference in relation to the previous versions of the problem is again the Validity property. The decision is no longer a single value but a vector with some of the initial values of the processes, at least $f + 1$ of which (the majority) are from correct processes. The validity property is now stated as:

- *Vector validity:* Every correct process that decides, decides on a vector V of size n:
  - $\forall_{p_i}$: if p_i is correct, then either *V[i]* is the value proposed by p_i or $\perp$;
  - at least $(f + 1)$ elements of V were proposed by correct processes.

Vector consensus is a variation for asynchronous systems of the classical *interactive consistency* problem (Pease et al., 1980). Interactive consistency is a consensus on a vector with values from all correct processes. However, in asynchronous systems a correct process can be very slow so it is not possible to guarantee that values from all correct processes are obtained and still ensure Termination. Therefore, vector consensus ensures only that $f + 1$ of the values in the vector are from correct processes. This is clearly more interesting than multi-valued consensus since it tells much more about the initial values of the correct processes. Vector consensus was proved to be equivalent to multi-valued consensus defined with the validity properties Validity 1, Validity 2A and Validity 3 (Correia et al., 2006).

Some other variations of consensus were studied in the literature. For instance, the *k-set consensus* problem in which the correct processes can decide at most k different values (Chaudhuri, 1993; de Prisco et al., 1999). The *Byzantine generals with alternative plans* problem takes into account the fact that processes may have several views about what decisions/actions are acceptable and unacceptable (Correia et al., 2008). Each process has a set of good decisions and a set of bad decisions. The problem is to make all correct processes agree on good decisions proposed by a correct process, and never on a bad decision.

A somewhat different kind of definition is the one used in the Paxos algorithms (Lamport, 1998, 2001; Lampson, 2001; Zielinski, 2004; Martin and Alvisi, 2005;

Variation	**Characterization**	**Properties that define the variation**		
binary	agreement about a binary value	Validity	Agreement	Termination
multi-valued	agreement about a value of a set with arbitrary size	Validity 1 or Validity 2 or Validity 1 + Validity 2A + Validity 3	Agreement	Termination
vector	agreement about a vector with values from some of the processes	Vector Validity	Agreement	Termination
Paxos	multi-valued consensus with processes with different roles	Safety 1 + Safety 2 + Safety 3		Liveness 1 + Liveness 2

Table 1 Summary of variations of Byzantine consensus and properties used to define them.

Cowling et al., 2006). The idea is that processes play one or more of the following roles: *proposers* (propose values), *acceptors* (choose the value to be decided) and *learners* (learn the chosen value). The problem can be defined in terms of five properties (Lamport, 2001; Martin and Alvisi, 2005):

- *Safety 1:* Only a value that has been proposed may be chosen.
- *Safety 2:* Only a single value may be chosen.
- *Safety 3:* Only a chosen value may be learned by a correct learner.
- *Liveness 1:* Some proposed value is eventually chosen.
- *Liveness 2:* Once a value is chosen, correct learners eventually learn it.

The first three properties are safety properties, while the last two are liveness properties. This definition is interesting because it allows a simple implementation of state machine replication in the crash failure model (Schneider, 1990; Lamport, 2001). However, in the Byzantine failure model this is more complicated because faulty processes can deviate from the algorithm in several ways (Castro and Liskov, 2002; Martin and Alvisi, 2005).

A summary of the different variations of Byzantine consensus and of the properties that are used to define them can be found in Table 1.

3 FLP Impossibility

The most cited paper about consensus is probably the one that proves the impossibility of solving consensus deterministically in an asynchronous system if a single process can crash (Fischer et al., 1985). This result is often called the FLP impossibility result, after its authors' names, Fischer, Lynch and Paterson. The consensus definition used to prove the result is even weaker than the first definition in Section 2: validity is more relaxed and termination is only required for a single process.

The idea is that the uncertainty in terms of timeliness (asynchrony) combined with the uncertainty in terms of failure (even if failures are only crashes and only one process can fail) does not allow any deterministic algorithm to guarantee that a decision is reached. More precisely, the reason for the impossibility is that in an asynchronous system it is impossible to differentiate a crashed process from another that is simply slow (or connected by a slow network link). In the years that followed the statement and proof of this result, a few alternative proofs have been given (a discussion on these related results can be found in (Lynch, 1989)).

The FLP result indicates when consensus is not solvable. However, from a practical point of view it is more important to know when it can be solved. A first detailed study of this issue was presented by Dolev, Dwork and Stockmeyer for crash faults (Dolev et al., 1987). The paper identified five relevant parameters that affect solvability: synchrony/asynchrony of the processes; synchrony/asynchrony of the communication; ordered/unordered message delivery; broadcast/point-to-point communication; and atomic/not-atomic receive and send. This lead to 32 different models. The paper showed that different degrees of synchronism allow deterministic algorithms to tolerate different numbers of crash faults (there is no equivalent study for Byzantine faults).

To solve consensus, an algorithm has to *circumvent* the FLP impossibility result. This word, circumvent, is quite imprecise so it is important to discuss its meaning. The idea is to slightly modify either the system model or the problem definition considered in (Fischer et al., 1985) to allow the problem to be solvable. These modifications change the conditions in which FLP was proven so, to be precise, the result simply no longer applies. However, researchers in the area prefer to call it "circumventing" the result, to pass the idea that the conditions are close to those in which the result applies.

An observation about FLP with interesting practical consequences is that if we discard one of the properties that define consensus, we can enforce the two others. This observation lead researchers to design consensus algorithms in the following way:

- the algorithm solves consensus if the technique used to circumvent FLP works as expected;
- the algorithm always satisfies the safety properties, even if the technique used to circumvent FLP does not work as assumed (Guerraoui, 2000; Guerraoui and Raynal, 2004).

The idea is that if something bad happens, like an additional timing assumption not being satisfied, the algorithm may not terminate, but Validity and Agreement properties will always be enforced. This notion has been called *indulgence* in the context of system models augmented with eventual failure detectors (Guerraoui, 2000; Guerraoui and Raynal, 2004), but almost all consensus algorithms satisfy it. The only exceptions that we are aware of are the randomized algorithms in (Toueg, 1984; Canetti and Rabin, 1993), which always terminate but only satisfy Agreement with a certain probability.

There are several ways to look into the techniques to circumvent FLP. We use a classification in five types of techniques, which we present in more detail in the following section:

- techniques that sacrifice determinism, leading to probabilistic algorithms;
- techniques that add time to the model;

Technique	Sub-technique	Positive	Negative	References
sacrifice determinism	randomization w/local coin	no timing assumptions	high expected number of rounds	Ben-Or (1983); Bracha (1984)
	randomization w/shared coin	no timing assumptions, low number of rounds	expensive cryptography	Rabin (1983); Toueg (1984); Ben-Or (1985); Canetti and Rabin (1993); Cachin et al. (2000); Friedman et al. (2005)
add time	partial synchrony	efficient if the network is stable	termination is delayed if network is unstable	Dwork et al. (1988)
	timed asynchrony		not studied for Byzantine consensus	Cristian and Fetzer (1998); Fetzer and Cristian (1995)
add oracle	failure detectors	efficient algorithms		Chandra and Toueg (1996); Malkhi and Reiter (1997); Kihlstrom et al. (2003); Doudou et al. (1999); Baldoni et al. (2003); Doudou et al. (2005); Friedman et al. (2005); Correia et al. (2010)
hybridization	wormhole	efficient algorithms, lower number of processes	wormhole has to be tamper-proof	Verissimo (2006); Correia et al. (2005); Neves et al. (2004, 2005); Correia et al. (2010)
change the problem	condition based approach		different problem	Mostefaoui et al. (2003b, 2004); Friedman et al. (2002)

Table 2 Summary of the techniques used to circumvent the FLP impossibility result in asynchronous Byzantine consensus algorithms.

- techniques that enrich the system model with an oracle;
- techniques that use hybridization;
- techniques that enrich the problem definition.

A summary of their characteristics can be found in Table 2.

4 Circumventing FLP

The following sections introduce the techniques to circumvent FLP and algorithms that use them.

4.1 Sacrificing Determinism

The FLP impossibility result applies to *deterministic* algorithms, therefore a solution to circumvent it is by using *randomization* to design *probabilistic* algorithms. More specifically, the idea is to substitute one of the properties that define consensus by a similar property that is satisfied only with a certain probability. For the reasons mentioned above when discussing indulgence, almost all algorithms choose to modify the Termination property, which becomes:

- *P-Termination:* Every correct process eventually decides with probability 1.

The single exceptions that we are aware of, already mentioned above, do not modify Termination but Agreement, so agreement on the decided value is reached with a certain probability (Toueg, 1984; Canetti and Rabin, 1993).

Randomized Byzantine consensus algorithms have been around since Ben-Or's and Rabin's seminal papers (Ben-Or, 1983; Rabin, 1983). Virtually all randomized consensus algorithms are based on a random operation, *tossing a coin*, which returns values 0 or 1 with equal probability.

These algorithms can be divided in two classes, depending on how the tossing operation is done: there are those that use a *local coin* mechanism in each process (starting with Ben-Or's work (Ben-Or, 1983)), and those based on a *shared coin* that gives the same values to all processes (initiated with Rabin's work (Rabin, 1983)).

Typically, local coin algorithms are simpler but terminate in an expected exponential number of communication steps (Ben-Or, 1983; Bracha, 1984), while shared coin algorithms require an additional coin sharing scheme but can terminate in an expected constant number of steps (Ben-Or, 1985; Cachin et al., 2000; Canetti and Rabin, 1993; Friedman et al., 2005; Rabin, 1983; Toueg, 1984). The original Rabin algorithm required a trusted dealer to distribute key shares before the execution of the algorithm (Rabin, 1983). This unpractical operation is no longer needed in newer algorithms (Canetti and Rabin, 1993; Cachin et al., 2000).

Randomized consensus algorithms have often been assumed to be inefficient due to their high expected message and time complexities, so they have been largely overlooked as a solution for the deployment of fault-tolerant distributed systems. Nevertheless, two important points have been chronically ignored. First, consensus algorithms are not usually executed in oblivion, but are run in the context of a higher-level problem (e.g., atomic broadcast) that can provide a friendly environment for faster termination (e.g., many processes proposing the same value can lead to a quick termination). Second, for the sake of theoretical interest, the proposed adversary models usually assume a strong adversary that completely controls the scheduling of the network and decides which processes receive which messages and in what order. In practice, a real adversary usually does not possess this ability, but if it does, it will probably perform simpler attacks like blocking the communication entirely. Therefore, in practice, the network scheduling can be "nice" and lead to a fast termination. Two papers show that this is true and that these algorithms can be practical (Moniz et al., 2006b,a).

A flavour of how randomized binary consensus works can be obtained from Algorithm 1. This is a somewhat informal presentation of one of the two seminal algorithms of the kind, which solves binary consensus defined in terms of the properties Validity, Agreement and P-Termination. Remember that $V = \{0, 1\}$, that n is the number of processes that execute the algorithm and that f is the maximum of those that can be faulty. This algorithm requires $n \geq 5f + 1$, which is sub-optimal. It assumes authenticated channels, i.e., that correct processes can not be impersonated. The algorithm uses a local coin, which is tossed in line 18. The basic idea of the algorithm is that processes disseminate an estimate of the value to be decided, and change of estimate or make a decision when they receive enough identical estimates from enough processes. Details about the algorithm can be found in the original paper that presents it (Ben-Or, 1983).

Algorithm 1 A randomized binary consensus algorithm by Ben-Or (1983).

```
estimate = my proposal
loop
    /* phase 1 */
    send estimate to all processes
    wait until messages from $n - f$ processes are received
    if there are more than $(n + f)/2$ messages with the same estimate $v$ then
        estimate_to_send = $v$
    else
        estimate_to_send = "no estimate"
    /* phase 2 */
    send estimate_to_send to all processes
    wait until messages from $n - f$ processes are received
    if there are at least $f + 1$ messages with the same estimate $v \in V$ then
        estimate = $v$
    if there are more than $(n + f)/2$ messages with the same estimate $v \in V$ then
        decide $v$
    else
        estimate = 0 or 1 with probability 1/2
```

4.2 Adding Time to the Model

The notion of *partial synchrony* was introduced by Dwork, Lynch and Stockmeyer in (Dwork et al., 1988). A partial synchrony model captures the intuition that systems can behave asynchronously (i.e., with variable/unknown processing/communication delays) for some time interval, but that they eventually stabilize and start to behave (more) synchronously. Therefore, the idea is to let the system be mostly asynchronous but to make assumptions about timing properties that are eventually satisfied. Algorithms based on this model are typically guaranteed to terminate only when these timing properties are satisfied.

Dwork et al. introduced two partial synchrony models, each one extending the asynchronous model with a timing property:

- *M1:* For each execution, there is an *unknown* bound on the message delivery time Δ, which is always satisfied.
- *M2:* For each execution, there is an unknown global stabilization time GST, such that a *known* bound on the message delivery time Δ is always satisfied from GST onward.

Chandra and Toueg proposed a third model, which is similar but weaker (Chandra and Toueg, 1996):

- *M3:* For each execution, there is an unknown global stabilization time GST, such that an *unknown* bound on the message delivery time Δ is always satisfied from GST onward.

Two Byzantine consensus algorithms based on M1 and M2 are presented in the original paper by Dwork et al. (1988). These algorithms are based on a *rotating coordinator*, meaning that in each round there is a special process (the coordinator) that tries to impose the value to be decided. In Algorithm 1 in every phase all processes would send their estimate to all others. On the contrary, in algorithms based on a rotating coordinator the communication pattern can differ depending on the phase. This difference in clear in the abstract representation of Dwork et al.'s algorithms that can be found in Algorithm 2. These algorithms manage to progress and terminate when the system becomes stable, i.e., when the system starts to behave synchronously. There is still no algorithm or proof that M3 allows Byzantine consensus to be solved, although it has been shown to be enough to solve crash-tolerant consensus (Chandra and Toueg, 1996).

Algorithm 2 Structure of the partially synchronous consensus algorithms of Dwork et al. (1988).

```
initialization
loop
  /* trying phase 1 – all-to-all */
  send estimate to all processes
  wait for a certain time that messages from $n - f$ processes are received
  /* trying phase 2 – coordinator-to-all */
  if I am the coordinator of the round then
    send estimate to all processes
  wait for a certain time that the coordinator's message is received
  /* trying phase 3 – all-to-coordinator */
  if the coordinator's message is received and the estimate found acceptable then
    send "estimate accepted" to the coordinator
  if I am the coordinator of the round then
    wait for a certain time that messages from $n - f$ processes are received
  /* lock-release phase – all-to-all */
  send estimate to all processes
  wait for a certain time that messages from $n - f$ processes are received
```

The timed asynchronous model enriches the asynchronous system model with hardware clocks that can be used to detect the violation of time bounds (Cristian and Fetzer, 1998). Cristian and Fetzer have shown that it is possible to solve consensus in this model, although the problem of Byzantine consensus has not been studied (Fetzer and Cristian, 1995).

4.3 Augmenting the System Model with an Oracle

The original idea of circumventing FLP using oracles was introduced by Chandra and Toueg (Chandra and Toueg, 1996). The oracle in that case is a *failure detector*, i.e., a component that gives hints about which processes are failed or not failed. Remember that the FLP result derives from the impossibility of distinguishing if a process is faulty or simply very slow. Therefore, intuitively, having a hint about the failure/crash of a process may be enough to circumvent FLP. Notice however that augmenting the system model with a failure detector is equivalent to modifying the time model since (useful) failure detectors cannot be implemented in asynchronous systems. In fact, timing assumptions, like those made in partial synchrony models, are usually necessary (e.g., Chandra and Toueg show that the weaker failure detector to solve consensus can be implemented in model M3). The single exception, is the implementation of failure detectors based on some order pattern in the messages exchanged that was proposed in (Mostefaoui et al., 2003a).

Next we present failure detectors. Other types of oracles have been presented in the literature, but they have not been used with Byzantine faults. Examples include the Ω detector, which provides hints about who is the leader process (Chandra et al., 1996), and ordering oracles, which provide hints about the order of messages broadcasted (Pedone et al., 2002).

The original idea of failure detectors was to detect or, more precisely, to suspect the crash of a process. Each process has attached a failure detector module and the set of all these modules formed the failure detector.

Several works have been applying the idea of Byzantine failure detectors to solve consensus (Baldoni et al., 2003, 2008; Correia et al., 2010; Doudou et al., 1999, 2005; Friedman et al., 2005; Kihlstrom et al., 2003; Malkhi and Reiter, 1997). The main differences in relation to crash failure detectors is that (1) Byzantine failure detectors can neither be made completely independent of the algorithm in which they are used (Doudou et al., 2002), nor (2) detect all Byzantine faults, only certain subsets (Kihlstrom et al., 2003).

Malkhi and Reiter presented a binary consensus algorithm based on a rotating coordinator. The leader/coordinator waits for a number of proposals from the others, chooses a value to be broadcasted and then waits for enough acknowledgments to decide (Malkhi and Reiter, 1997). If the leader is suspected by the failure detector, a new one is chosen and the same procedure is applied. The same paper also described a hybrid algorithm combining randomization and an unreliable failure detector. The algorithm by Kihlstrom et al. also solves the same type of consensus but requires weaker communication primitives and uses a failure detector that detects more Byzantine failures, such as invalid and inconsistent messages (Kihlstrom et al., 2003).

Doudou and Schiper presented an algorithm for crash fault-tolerant vector consensus based on a *muteness failure detector*, which detects if a process stops sending messages to another one (Doudou et al., 1999). This algorithm is also based on a rotating coordinator that proposes an estimate that the others broadcast and accept, if the coordinator is not suspected. This muteness failure detector was used to solve Byzantine multi-valued consensus (Doudou et al., 2002). Another efficient algorithm based on a muteness failure detector was presented by Friedman et al. (2005).

Baldoni et al. described a vector consensus algorithm based on two failure detectors (Baldoni et al., 2003). One failure detector detects if a process stops sending messages (muteness) while the other detects other Byzantine failures. This latter detector is implemented using an interesting solution based on a finite-state automaton that monitors the behaviour of the algorithm.

All algorithms based on failure detectors that we are aware of are indulgent, i.e., they satisfy the safety properties of consensus (Validity and Agreement) even if the failure detector does not behave “nicely”. Examples of undesirable behaviour of a failure detector are not detecting a subset of Byzantine behaviour or the muteness of a process.

4.4 Hybridization and Wormholes

Wormholes are extensions to a system model with stronger properties than the rest of the system. Wormholes are materialized as enhanced components that provide processes with a means to obtain a few simple privileged functions with "good" properties otherwise not guaranteed by the normal environment (Verissimo, 2003, 2006). For example, a wormhole can provide timely or secure functions in, respectively, asynchronous or Byzantine systems. The Trusted Platform Module is a commercial component that can be considered to be a (secure) wormhole (Trusted Computing Group, 2006). This way of modelling systems contrasts with work on failure detectors, which tries to abstract the minimum requirements on hints about failures to solve consensus. The idea is more generic and has to do with what are the distributed system models that allow to have desirable levels of predictability in systems that are mostly uncertain in terms of properties like time and security (Verissimo, 2006).

Wormholes are closely related to the notion of *architectural hybridization*, a well-founded way to substantiate the provisioning of those "good" properties on "weak" environments. In the case that we are interested in here, we assume that the system is essentially asynchronous and Byzantine, so when implementing the model we should not simply postulate that parts of it behave in a timely or secure fashion, or these assumptions might naturally fail. Instead, those parts should be built in a way that our claim is guaranteed with high confidence.

The first paper that presented a consensus algorithm based on a wormhole (Correia et al., 2005) used a specific wormhole, a device called *Trusted Timely Computing Base* (TTCB) (Correia et al., 2002). Technically, the TTCB is a secure real-time and fail-silent distributed component. Applications implementing the consensus algorithm run in the normal system, i.e., in the asynchronous Byzantine system. However, the TTCB is locally accessible to any process, and at certain points of the algorithm the processes can use it to execute correctly (small) crucial steps. The consensus algorithm relies mostly on a TTCB service called Trusted Block Agreement Service, which essentially makes an agreement on small values proposed by a set of processes. The idea is to use this service to make agreement on the hash of the value proposed by the majority of the processes. Later, a simpler multi-valued consensus algorithm and a vector consensus based on the TTCB were also proposed (Neves et al., 2004, 2005).

Another version of the TTCB was used to implement atomic broadcast and state machine replication with only $2f + 1$ replicas (Correia et al., 2004). Although that paper did not present an algorithm for solving consensus, a multi-valued consensus with Validity 2 can be trivially implemented on top of that atomic broadcast: each process proposes a value by atomic broadcasting it; the first value delivered is the result of the consensus. It is also simple to see that the scheme can not be used to implement consensus based on Validity 1 or vector consensus.

More recently, Chun et al., proposed the *Attested Append-Only Memory* (A2M), another wormhole used to implement state machine replication with only $2f + 1$ replicas (Chun et al., 2007). Like the TTCB, the A2M has to be tamper-proof, but it is local to the computers, not distributed. Replicas using the A2M are forced to commit to a single, monotonically increasing sequence of operations. Since the sequence is externally verifiable, faulty replicas can not present different sequences to different replicas. Veronese et al. presented an even simpler wormhole that also allows implementing state machine replication with only $2f + 1$ replicas, the *Unique Sequential Identifier Generator* (USIG) (Veronese et al., 2009a,b). This component contains only a counter and a few cryptographic functions that are used to associate sequence numbers to certain operations done by the replicas, e.g., producing a signed certificate that proves unequivocally that the number is assigned to that message. Levin et al. proposed a similar component, TrInc, which might also be used to implement state machine replication (Levin et al., 2009). All these state machine replication algorithms might be used to implement consensus in the way explained in the previous paragraph.

4.5 Modifying the Problem

"If you can't beat them join them". Last but not the least, this section describes how FLP can be circumvented by weakening the very definition of consensus, i.e., by modifying it. Currently, we are aware of a single type of algorithm that fits in this category in the system model that we consider in the paper: algorithms based on the *condition based approach* (Mostefaoui et al., 2003b, 2004; Friedman et al., 2002). For crash faults, there is also k-set consensus, which allows k different values to be decided (Chaudhuri, 1993).

Consensus algorithms based on the condition-based approach terminate if the initial values of the processes satisfy certain conditions, but satisfy the safety properties – Validity and Agreement – even if the conditions are not valid. Let us define the *input vector* for an execution of a consensus algorithm as the vector I in which each $I[i]$ is the initial value of process p_i. The condition based approach identifies sets of input vectors for which the consensus algorithm terminates (besides satisfying Validity and Agreement). Conditions on input vectors were shown to be directly related to error correcting codes. In fact, crash failures correspond to erasure errors in the context of error correcting codes, while Byzantine failures correspond to corruption errors (Friedman et al., 2002).

An argument in favour of this sort of trade-off between Termination and conditions on input vectors is made in (Friedman et al., 2002). A first reason is that it makes sense to use the approach to efficiently solve consensus problems in which the initial values really satisfy the conditions, but to guarantee safety even if this assumption does not hold. A second reason is that the conditions can serve as a guideline that allows the designer to augment the system (modifying the system model) with the minimum synchrony needed to ensure the solvability of the problem.

The single paper about the condition based approach that we are aware of that deals with Byzantine failures is (Friedman et al., 2002). This paper presents simple algorithms to solve multi-valued and k-set consensus.

5 Evaluating Consensus Algorithms

Byzantine distributed algorithms have been evaluated using several different metrics. Ultimately, the objectives are to understand how an algorithm works and how it behaves in practice:

- How will it *perform*? Or, more precisely, what will be its latency (time needed to run) and throughput (number of executions per unit of time)?
- How will it *scale*, i.e., what is the relation between its performance and the number of processes executing it?
- What will be its *resilience*, i.e., how many faulty processes will it tolerate?

5.1 Performance and Scalability

The first two parameters are usually evaluated theoretically in terms of time, message and communication complexities. In asynchronous systems, time complexity is usually measured in terms of the maximum number of *asynchronous steps*. An asynchronous step involves a process sending a message and receiving one or more messages sent by the other processes. The message complexity is measured by the *number of messages sent* and the communication complexity by the *number of bits sent*. Cryptographic operations often have some impact in the processing time, especially public-key cryptography operations, so the evaluation should also take into account, e.g., the number of signatures made and evaluated. It has been shown that the minimum number of asynchronous steps for Paxos consensus is two (Dutta et al., 2005; Martin and Alvisi, 2005).

These metrics are not so simple to assess as it may seem, since they usually depend on the occurrence of faults. Therefore, the evaluations should consider at least two cases: failure-free executions and executions in which the maximum number of processes (f) is faulty (Byzantine). Other aspects, like the correct processes having the same initial value, can influence the performance evaluation and should also be taken into account.

For randomized algorithms, these parameters can only be stated probabilistically, so often the metrics considered are the *expected* number of asynchronous steps, messages sent, and bits sent. The literature usually assesses these values in the worst case, i.e., with the most unfavourable combination of initial values, failures and network scheduling of the messages.

Despite the importance of these theoretical metrics, it has been argued that they may not reflect correctly the behaviour of the algorithms in practice (Keidar, 2002). Some authors have shown that randomized binary consensus algorithms that in theory run in high numbers of steps, in practice may execute in only a few communication steps under realistic conditions (Moniz et al., 2006b).

5.2 Resilience

The third parameter above, resilience, can be assessed precisely for an algorithm. The optimal resilience for Byzantine consensus in all system models without wormholes that we are aware of is $n/3$, i.e., less than $n/3$ out of n processes can fail for the algorithm to run correctly (Lamport et al., 1982; Bracha, 1984; Dwork et al., 1988; Correia et al., 2006). Baldoni et al. present an algorithm that assumes $f \leq min(\lfloor (n-1)/2 \rfloor, C)$, where C is the maximum number of faulty processes allowed by the certification algorithm (Baldoni et al., 2003). However, they point out that known certification techniques assume $n - C = \lceil \frac{2n+1}{3} \rceil$, so their algorithm also requires $n \geq 3f + 1$.

As mentioned in the Section 4.4, using wormholes it is possible to design Byzantine consensus algorithms with better resilience, $n/2$, or $n \geq 2f + 1$. That section mentioned a few state machine replication algorithms that might be used to implement consensus with this resilience. However, to the best of our knowledge, (Correia et al., 2010) is the only work that presents a complete asynchronous Byzantine consensus algorithms with this resilience. The paper shows that the impossibility of improving the resilience of consensus from $n/3$ to $n/2$ without a wormhole comes from an important component of most of these algorithms, a *reliable broadcast* algorithm. The reliable broadcast problem consists essentially in guaranteeing that when a process sends a message, all processes deliver that message, or possibly no message at all if the sender is faulty (Bracha, 1984). The paper shows that using a variation of the USIG wormhole it is possible to design a reliable broadcast algorithm that imposes no bounds on the number of faulty processes, unlike previous existing algorithms that require $n \geq 3f + 1$ (Bracha, 1984; Reiter, 1994). Using this reliable broadcast algorithm it is possible to implement consensus with only $n = 2f + 1$ processes. The same paper presents a methodology to transform asynchronous *crash* consensus algorithms into asynchronous Byzantine consensus algorithms with different characteristics keeping the number of processes of $n \geq 2f + 1$.

In relation to resilience, it is important to note that there is no point in making assumptions about the maximum number of processes that can be faulty if there are common modes of failure, i.e., if some faults can affect all processes (Powell, 1992). For the Byzantine failure model, common modes of failure are caused by identical bugs or vulnerabilities in all (or several) processes (Verissimo et al., 2009). Some degree of independence of failure of processes can be enforced by using diversity of design (Littlewood and Strigini, 2004; Obelheiro et al., 2006; Deswarte et al., 1998).

6 Related and Equivalent Problems

In the introduction, we mentioned that there are several distributed systems problems equivalent to consensus. In this section we give more details about this issue.

Given two distributed problems A and B, a *transformation* from A to B is an algorithm that converts any algorithm that solves A into an algorithm that solves B (Hadzilacos and Toueg, 1994). Problems A and B are said to be *equivalent* if there is a transformation from A to B and a transformation from B to A. An equivalence is always proven considering a certain system model, and may not exist if the model is modified.

The first equivalences and transformations were established for the crash failure model. In this model, multi-valued consensus has been proved to be equivalent to atomic (or total order) broadcast (Hadzilacos and Toueg, 1994; Chandra and Toueg, 1996). Transformations from consensus to several problems were also presented: non-blocking atomic commit (Guerraoui and Schiper, 2001), group membership (Guerraoui and Schiper, 2001), and state machine replication (Schneider, 1990). Only some of these equivalences/transformations extend to the Byzantine failure model. For instance, non-blocking atomic commit commits a transaction if all resources say ‘commit’ and aborts it one or more say ‘abort’. With Byzantine failure model, a faulty process can simply abort all transactions preventing the system from working as expected, so clearly there is no transformation from consensus to non-blocking atomic commit.

The equivalence of (Byzantine) atomic broadcast and consensus has been first proved for systems with signatures in (Cachin et al., 2001). A similar result but without the requirement of signatures has been proved in (Correia et al., 2006). Both proofs are independent of the technique used to circumvent FLP. Atomic broadcast, or total order broadcast, is the problem of delivering the same messages in the same order to all processes.

We are not aware of transformations from Byzantine consensus to other distributed systems problems. However, there is probably a transformation from vector consensus to group membership. A group membership algorithm makes agreement about a sequence of views, which are numbered events with the identifiers of the members of a group of processes (see, e.g., the survey in (Chockler et al., 2001)). The view can be modified by events like the addition of members to a group, the removal of failed members, and the removal of members by their own initiative. The Byzantine-resilient membership algorithms available give this intuition that a transformation might be defined (Reiter, 1996; Kihlstrom et al., 2001; Correia et al., 2007).

Several transformations from a variation of (Byzantine) consensus to another were presented in the literature. Turpin and Coan presented a transformation from binary to multi-valued consensus for synchronous systems (Turpin and Coan, 1984). Toueg and Cachin et al. presented similar transformations for asynchronous systems, both requiring signatures (Toueg, 1984; Cachin et al., 2001). Transformations from binary to multi-valued consensus, and from multi-valued to vector consensus, without signatures, were presented in (Correia et al., 2006).

7 Conclusion

Consensus is an important problem in distributed systems since it can be considered to be the "greatest common sub-problem" of several others (Mostefaoui et al., 2000). In this paper a short survey about research on consensus in asynchronous message-passing systems prone to Byzantine faults was provided. The paper started by discussing the more common variations and definitions of Byzantine consensus found in the literature. Then, it presented the FLP impossibility result, which albeit being a negative result, ended up being a driving force of research in area. The following section presented several classes of consensus algorithms classifying them in terms of the way in which they circumvent FLP. The paper ended with a discussion about the performance, scalability and resilience of consensus algorithms, and the relation between consensus and other problems like atomic broadcast.

Algorithms that solve the several variations of this problem and the equivalent problem of atomic broadcast are currently being used as fundamental building blocks in secure and intrusion-tolerant applications. Like previously crash fault-tolerant consensus, Byzantine fault-tolerant consensus is becoming an important component of distributed systems. This, in fact, is the current and probably major future trend in the area: the design of new algorithms and the adaptation of previous ones to solve real, practical problems.

Acknowledgments

We thank Michel Raynal and David Powell for their comments on preliminary drafts of this survey that greatly assisted us in improving it. This work was partially supported by the EC through NoE IST-4-026764-NOE (ReSIST) and the Alban scholarship E05D057126BR, and by the FCT through the Multiannual and the CMU-Portugal Programmes, the project PTDC/EIA-EIA/100581/2008 (REGENESYS) and the project PTDC/EIA-EIA/100894/2008 (DIVERSE).

References

Aguilera, M. (2004). A pleasant stroll through the land of infinitely many creatures. *ACM SIGACT News*, 35(2):36–59.

Alchieri, E. A. P., Bessani, A. N., Fraga, J. S., and Greve, F. (2008). Byzantine consensus with unknown participants. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*, pages 22–40.

Alon, N., Merrit, M., Reingold, O., Taubenfeld, G., and Wright, R. (2005). Tight bounds for shared memory systems acessed by Byzantine processes. *Distributed Computing*, 18(2):99–109.

Alpern, B. and Schneider, F. (1984). Defining liveness. Technical report, Department of Computer Science, Cornell University.

Attie, P. C. (2002). Wait-free Byzantine consensus. *Information Processing Letters*, 83(4):221–227.

Baldoni, R., Hélary, J.-M., and Piergiovanni, S. T. (2008). A methodology to design arbitrary failure detectors for distributed protocols. *Journal of Systems Architecture*, 54(7):619–637.

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

Ben-Or, M. (1983). Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30.

Ben-Or, M. (1985). Fast asynchronous Byzantine agreement. In *Proceedings of the 4th ACM Symp. on Principles of Distributed Computing*, pages 149–151.

Bessani, A. N., Correia, M., Fraga, J. S., and Lung, L. C. (2009). Sharing memory between Byzantine processes using policy-enforced tuple spaces. *IEEE Transactions on Parallel and Distributed Systems*, 20(3):419–432.

Bessani, A. N., Sousa, P., Correia, M., Neves, N. F., and Verissimo, P. (2008). The CRUTIAL way of critical infrastructure protection. *IEEE Security & Privacy*, pages 44–51.

Bracha, G. (1984). An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 154–162.

Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. (2001). Secure and efficient asynchronous broadcast protocols (extended abstract). In Kilian, J., editor, *Advances in Cryptology: CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer-Verlag.

Cachin, C., Kursawe, K., and Shoup, V. (2000). Random oracles in Contanstinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 123–132.

Canetti, R. and Rabin, T. (1993). Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 42–51.

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

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

Chandra, T. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

Chaudhuri, S. (1993). More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158.

Chockler, G., Keidar, I., and Vitenberg, R. (2001). Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):427–469.

Chun, B.-G., Maniatis, P., Shenker, S., and Kubiatowicz, J. (2007). Attested append-only memory: making adversaries stick to their word. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, pages 189–204.

Correia, M., Bessani, A. N., and Verissimo, P. (2008). On Byzantine generals with alternative plans. *Journal of Parallel and Distributed Computing*, 68(9):1291–1296.

Correia, M., Neves, N. F., Lung, L. C., and Verissimo, P. (2005). Low complexity Byzantine-resilient consensus. *Distributed Computing*, 17(3):237–249.

Correia, M., Neves, N. F., Lung, L. C., and Verissimo, P. (2007). Worm-IT – a wormhole-based intrusion-tolerant group communication system. *Journal of Systems and Software*, 80(2):178–197.

Correia, M., Neves, N. F., and Verissimo, P. (2004). How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, pages 174–183.

Correia, M., Neves, N. F., and Verissimo, P. (2006). From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *Computer Journal*, 41(1):82–96.

Correia, M., Verissimo, P., and Neves, N. F. (2002). The design of a COTS real-time distributed security kernel. In *Proceedings of the Fourth European Dependable Computing Conference*, pages 234–252.

Correia, M., Veronese, G. S., and Lung, L. C. (2010). Asynchronous Byzantine consensus with 2f+1 processes. In *Proceedings of the 25th Annual ACM Symposium on Applied Computing*, pages 475–480.

Cowling, J., Myers, D., Liskov, B., Rodrigues, R., and Shrira, L. (2006). HQ-Replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Proceedings of 7th Symposium on Operating Systems Design and Implementations*, pages 177–190.

Cristian, F. and Fetzer, C. (1998). The timed asynchronous system model. In *Proceedings of the 28th IEEE International Symposium on Fault-Tolerant Computing*, pages 140–149.

de Prisco, R., Malki, D., and Reiter, M. (1999). On k-set consensus problems in asynchronous systems. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, pages 257–265.

Deswarte, Y., Kanoun, K., and Laprie, J. C. (1998). Diversity against accidental and deliberate faults. In *Computer Security, Dependability, & Assurance: From Needs to Solutions*. IEEE Press.

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

Doudou, A., Garbinato, B., and Guerraoui, R. (2002). Encapsulating failure detection: From crash-stop to Byzantine failures. In *International Conference on Reliable Software Technologies*, pages 24–50.

Doudou, A., Garbinato, B., and Guerraoui, R. (2005). Tolerating arbitrary failures with state machine replication. In Diab, H. B. and Zomaya, A. Y., editors, *Dependable Computing Systems Paradigms, Performance Issues, and Applications*, chapter 2, pages 27–56. Wiley.

Doudou, A., Garbinato, B., Guerraoui, R., and Schiper, A. (1999). Muteness failure detectors: Specification and implementation. In *Proceedings of the Third European Dependable Computing Conference*, pages 71–87.

Dutta, P., Guerraoui, R., and Vukolic, M. (2005). Best-case complexity of asynchronous Byzantine consensus. Technical Report 200499, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

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

Fetzer, C. and Cristian, F. (1995). On the possibility of consensus in asynchronous systems. In *Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems*.

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

Friedman, R., Mostefaoui, A., Rajsbaum, S., and Raynal, M. (2002). Distributed agreement and its relation with error-correcting codes. In *Proceedings of the 16th International Conference on Distributed Computing*, pages 63–87.

Friedman, R., Mostefaoui, A., and Raynal, M. (2005). Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56.

Guerraoui, R. (2000). Indulgent algorithms. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 289–298.

Guerraoui, R., Hurfin, M., Mostefaoui, A., Oliveira, R., Raynal, M., and Schiper, A. (2000). Consensus in asynchronous distributed systems: A concise guided tour. In Krakowiak, S. and Shrivastava, S., editors, *Advances in Distributed Systems*, number 1752 in Lecture Notes in Computer Science, pages 33–47. Springer-Verlag.

Guerraoui, R. and Raynal, M. (2004). The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453.

Guerraoui, R. and Schiper, A. (1997). Consensus: the big misunderstanding. In *Proceedings of the IEEE International Workshop on Future Trends in Distributed Computing Systems*.

Guerraoui, R. and Schiper, A. (2001). The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41.

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

Hauser, C. H., Bakken, D. E., Dionysiou, I., Gjermundrød, K. H., Irava, V. S., and Bose, A. (2008). Security, trust and QoS in next-generation control and communication for large power systems. *International Journal of Critical Infrastructures*, 4(1/2).

Herlihy, M. (1991). Wait-free synchronization. *ACM Transactions on Programing Languages and Systems*, 13(1):124–149.

Keidar, I. (2002). Challenges in evaluating distributed algorithms. In *Proceedings of the International Workshop on Future Directions in Distributed Computing.*

Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (2001). The SecureRing group communication system. *ACM Transactions on Information and System Security*, 4(4):371–406.

Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (2003). Byzantine fault detectors for solving consensus. *The Computer Journal*, 46(1):16–35.

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

Lamport, L. (2001). Paxos made simple. *SIGACT News*, 32(4):51–58.

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

Lampson, B. (2001). The ABCD's of Paxos. In *Proceedings of the 20th annual ACM Symposium on Principles of Distributed Computing*, page 13.

Levin, D., Douceur, J. R., Lorch, J. R., and Moscibroda, T. (2009). TrInc: small trusted hardware for large distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 1–14.

Littlewood, B. and Strigini, L. (2004). Redundancy and diversity in security. In Samarati, P., Rian, P., Gollmann, D., and Molva, R., editors, *Computer Security – ESORICS 2004, 9th European Symposium on Research Computer Security*, LNCS 3193, pages 423–438. Springer.

Lynch, N. (1989). A hundred impossiblility proofs for distributed computing. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing.*

Lynch, N. (1996). *Distributed Algorithms.* Morgan Kaufmann, San Mateo, CA.

Malkhi, D., Merrit, M., Reiter, M., and Taubenfeld, G. (2003). Objects shared by Byzantine processes. *Distributed Computing*, 16(1):37–48.

Malkhi, D. and Reiter, M. (1997). Unreliable intrusion detection in distributed computations. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 116–124.

Martin, J. P. and Alvisi, L. (2005). Fast Byzantine consensus. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, pages 402–411.

Moniz, H., Neves, N. F., Correia, M., and Verissimo, P. (2006a). Experimental comparison of local and shared coin randomized consensus protocols. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, pages 235–244.

Moniz, H., Neves, N. F., Correia, M., and Verissimo, P. (2006b). Randomized intrusion-tolerant asynchronous services. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 568–577.

Mostefaoui, A., Mourgaya, E., and Raynal, M. (2003a). Asynchronous implementation of failure detectors. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, pages 351–360.

Mostefaoui, A., Rajsbaum, S., and Raynal, M. (2003b). Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922–954.

Mostefaoui, A., Rajsbaum, S., Raynal, M., and Roy, M. (2004). Condition based consensus solvability: A hierarchy of conditions and efficient protocols. *Distributed Computing*, 17:1–20.

Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and Abbadi, A. E. (2005). From static distributed systems to dynamic systems. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 109–118.

Mostefaoui, A., Raynal, M., and Tronel, F. (2000). From binary consensus to multivalued consensus in asynchronous message-passing systems. *Information Processing Letters*, (73):207–212.

Neves, N. F., Correia, M., and Verissimo, P. (2004). Wormhole-aware Byzantine protocols. In *2nd Bertinoro Workshop on Future Directions in Distributed Computing: Survivability - Obstacles and Solutions*.

Neves, N. F., Correia, M., and Verissimo, P. (2005). Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1120–1131.

Obelheiro, R. R., Bessani, A. N., Lung, L. C., and Correia, M. (2006). How practical are intrusion-tolerant distributed systems? DI-FCUL TR 06–15, Dep. of Informatics, Univ. of Lisbon.

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

Pedone, F., Schiper, A., Urbán, P., and Cavin, D. (2002). Solving agreement problems with weak ordering oracles. In *Proceedings of the Fourth European Dependable Computing Conference*, pages 44–61.

Powell, D. (1992). Fault assumptions and assumption coverage. In *Proceedings of the 22nd IEEE International Symposium of Fault-Tolerant Computing*.

Rabin, M. O. (1983). Randomized Byzantine generals. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 403–409.

Reiter, M. (1994). Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80.

Reiter, M. K. (1996). A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42.

Schneider, F. B. (1990). Implementing faul-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

Toueg, S. (1984). Randomized Byzantine agreements. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 163–178.

Trusted Computing Group (2006). Trusted computing platform module main specification. version 1.2, revision 94. http://www.trustedcomputinggroup.org.

Turpin, R. and Coan, B. A. (1984). Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters*, 18(2):73–76.

Verissimo, P. (2003). Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 108–113. Springer-Verlag.

Verissimo, P. (2006). Travelling through wormholes: A new look at distributed systems models. *SIGACT News*, 37(1):66–81.

Verissimo, P., Bessani, A. N., Correia, M., Neves, N. F., and Sousa, P. (2009). Designing modular and redundant cyber architectures for process control: Lessons learned. In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences*.

Verissimo, P., Neves, N. F., and Correia, M. (2008). The CRUTIAL reference critical information infrastructure architecture: a blueprint. *International Journal of System of Systems Engineering*, 1(1/2):78–95.

Veronese, G. S., Correia, M., Bessani, A. N., C., L., and Verissimo, P. (2009a). Minimal Byzantine fault tolerance: Algorithm and evaluation. DI/FCUL TR 09–15, Department of Computer Science, University of Lisbon.

Veronese, G. S., Correia, M., Bessani, A. N., and Lung, L. C. (2009b). Highly-resilient services for critical infrastructures. In *Proceedings of the Embedded Systems and Communications Security Workshop*.

Yin, J., Martin, J., Venkataramani, A., Alvisi, L., and Dahlin, M. (2003). Separating agreement from execution for Byzantine fault tolerant services. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 253–267.

Zielinski, P. (2004). Paxos at war. Technical Report UCAM-CL-TR-593, University of Cambridge Computer Laboratory, Cambridge, UK.