On the Impossibility of Group Membership*

Tushar Deepak Chandra† Vassos Hadzilacos‡ Sam Toueg§

Bernadette Charron-Bost¶

Abstract

We prove that the primary-partition group membership problem cannot be solved in asynchronous systems with crash failures, even if one allows the removal or killing of non-faulty processes that are erroneously suspected to have crashed.

1 Introduction

The problem of *group membership* has been the focus of much theoretical and experimental work on fault-tolerant distributed systems. A group membership protocol manages the formation and maintenance of a set of processes called a *group*. For example, a group may be a set of processes that are cooperating towards a common task (e.g., the primary and backup servers of a database), a set of processes that share a common interest (e.g., clients that subscribe to a particular newsgroup), or the set of all processes in the system that are currently deemed to be operational. In general, a process may *leave* a group because it failed, it voluntarily requested to leave, or it is forcibly expelled by other members of the group. Similarly, a process may *join* a group; for example, it may have been selected to replace a process that has recently left the group. A group membership protocol must manage such dynamic changes in some coherent way. In this paper we consider group membership protocols that ensure processes agree on the current membership of the group.

The group membership problem was first defined for synchronous systems by [Cri91]. Since then, the group membership problem for asynchronous systems has also been the subject of intense investigation (e.g., [KT91, MPS91, RB91, VVR92, JFR93, vRBC+93, BDGB94, BS94, DMS94, EMS95, MSMA94]). Yet, despite the wide interest that it has attracted and the numerous publications on this subject, the group membership problem for asynchronous systems is far from being understood: In particular, there is no agreed definition for this problem, and some of the most referenced formal definitions are unsatisfactory [ACBMT95].

Despite their differences, all versions of the group membership problem require some form of process agreement in systems with failures. Another well-known problem requiring agreement in spite of failures is *Consensus*. This problem, however, cannot be solved in asynchronous systems even if commu-

*Research partially supported by NSF grant CCR-9402894, DARPA/NASA Ames grant NAG-2-593, and a grant from the Natural Sciences and Engineering Research Council of Canada.

†H2-L10, IBM T. J. Watson Research Center, 30 Saw Mill Road, Hawthorne, NY 10532, USA.

‡Computer Systems Research Institute, University of Toronto, 6 King's College Road, Toronto, Ontario, Canada M5S 1A1.

§Department of Computer Science, Upson Hall, Cornell University, Ithaca, NY 14853, USA. This author worked on this paper while he was at INRIA (Project REFLECS) Rocquencourt, France.

¶Laboratoire d'Informatique LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, FRANCE.

nication is reliable, only one process may fail, and it can do so only by *crashing*, i.e., if it stops executing steps [FLP85]. Since the purpose of group membership is to ensure some kind of agreement among processes (on the membership of a dynamically changing set), the potential for running into a similar impossibility result is obvious. On the other hand, group membership is different from Consensus in at least two ways:

- In group membership, a process that is suspected to have crashed can be *removed* from the group, or even *killed*, even if this suspicion is actually incorrect (e.g., the suspected process was only very slow). The [FLP85] model does not speak about process removals, and it does not directly model process killing (i.e., program-controlled crashes).
- Consensus requires progress in *all* runs, while group membership allows runs that "do nothing" (for instance, "doing nothing" is desirable when no process wishes to join or leave the group, and no process crashes).

These differences appear to make group membership weaker than Consensus, and in fact the first one has been widely cited as a reason why group membership is solvable in asynchronous systems while Consensus is not [RB91, ADKM92, DKM93, DMS94, EMS95]. In this paper we prove that this is not so: We define a problem called *WGM* (for *Weak Group Membership*) that allows the removal of erroneously suspected processes from the group, and is subsumed by any reasonable definition of group membership, and show that WGM cannot be solved in asynchronous systems with failures. We first show the impossibility of WGM in systems where communication is reliable and where at most one process may crash, exactly as in [FLP85]. We then extend this negative result to systems that allow program-controlled process crashes.

One of the reasons why WGM is weak is that it makes no attempt to "track" failures: In contrast to most existing group membership services whose goal is to maintain a set of processes that are deemed to be operational (e.g., [RB91, JFR93, BDGB94, DMS94, MSMA94]), WGM does not link the membership of the group with actual or suspected crashes. Indeed, the only WGM requirement that ties the membership of the group to "reality" is a very weak and natural one: If a single process p requests to leave the group, a WGM protocol should not preclude the possibility that p is indeed the only process removed from the group.

It is important to note that our result applies only to group membership services that attempt to maintain a *single* agreed view of the current membership of a group (e.g., [RB91, KT91, MPS91, MSMA94, HS95]). These are known as *primary-partition* group membership services and are intended for systems with no network partitions, or for systems that allow the group membership to change in at most one network partition, the "primary partition". So-called *partitionable* group membership services, which allow *multiple* views of the group to coexist, have also been proposed [JFR93, vRBC+93, BDGB94, DMS94, DMS95, EMS95]. Some remarks about such services are given at the end of this paper.

The rest of this paper is organized as follows. In Section 2, we define WGM and explain why it is weak. In Section 3, we show the impossibility of WGM in the [FLP85] model of asynchronous systems. The proof is patterned after the one given in [FLP85] with the exception of the starting lemma. In Section 4 we strengthen this impossibility result in various ways, and in Section 5 we extend it to systems that allow program-controlled crashes. Section 6 concludes the paper with some observations on group membership services.

2 The WGM Problem

We now describe WGM, a problem that captures a small fragment of what a realistic (primary-partition) group membership service should be able to provide. The impossibility of solving WGM in asynchronous systems with failures implies the impossibility of implementing more realistic group membership services in such systems. The informal specification of WGM given here is sufficient to understand the scope and meaning of our result; a formal specification in terms of the model of [FLP85] is given in the next section.

Consider a system with $n \geq 4$ processes $p_1, p_2, \ldots, p_n$, that can fail only by *crashing*, i.e., by stopping to execute steps. If a process never crashes, we say it is *correct*, otherwise it is *faulty*. We assume that initially *all* processes are in the same group G. More precisely, the *first view of G* of every process is the set $V_{init} = \{p_1, p_2, \ldots, p_n\}$.

At the beginning, some processes may wish to (voluntarily) leave group G. This, as well as actual or suspected crashes, may cause the membership of G to change: For any $V \subseteq V_{init}$, we say that a process *installs new view* V if it decides that V is its second view of G (such a decision is *irrevocable*). WGM satisfies the following properties:

1. If p_1, or p_2, or both (and no other process) wish to leave G, then at least one process eventually installs a new view, and no process installs a different new view.
2. For both $i \in \{1, 2\}$, if p_i is the only process that wishes to leave G, then it is possible that some process installs new view $V_i = V_{init} - \{p_i\}$. In other words, there is at least one execution in which p_i is the only process that wishes to leave G and also the only one removed from G.

Compared to a full-fledged group membership service, WGM is very weak. It imposes restrictions only on the *first* view change, and only if p_1 or p_2 initially wish to leave the group (it says nothing about subsequent view changes, or about what should happen if any process other than p_1 or p_2 wishes to leave G). Even in this case, only one process is required to install a new view. More importantly:

- WGM does not attempt to track actual (or even suspected) failures.
- The second requirement of WGM is natural for any reasonable group membership service. To see this, consider executions in which p_i is the only process that wishes to leave G, and in which no failures or failure suspicions occur. It stands to reason that in at least one such execution, p_i and only p_i is removed from G.
- Except for the two executions mentioned in the previous item, there is no restriction on the size or content of the new view: In a single view change WGM is allowed to remove from group G an arbitrary set of processes, including correct processes that did not wish to leave G. This is compatible with any group membership service that is allowed to remove from the group processes that are erroneously suspected to have crashed.
- There are no requirements on requests to *join* the group.
- There are no requirements on the broadcast or multicast of messages. In other words, there are no restrictions whatsoever on message delivery, and in particular on the way such deliveries interleave with view changes.[1]

3 Impossibility of WGM in the [FLP85] Model

We now formally define the WGM problem and show that it cannot be solved in the [FLP85] model of distributed computing, namely, in asynchronous systems with reliable communication where at most one process may crash. The proof is very similar to the one of Fischer, Lynch and Paterson in [FLP85]. Essentially, the only difference stems from the fact that Consensus requires "termination" in all executions, while WGM does not. This affects principally one lemma — the only one whose proof we give in detail here.

The main features of the [FLP85] model are informally highlighted below.

- The system consists of n processes $p_1, p_2, \ldots, p_n$, that communicate by sending messages over a completely connected point-to-point network. We assume $n \geq 4$.
- Processes execute steps at arbitrary speeds that can vary over time.
- At most one process may fail and can fail only by crashing.
- Communication links are reliable but asynchronous: any message sent to a process that does not crash is received with a finite but arbitrary delay.

We expect readers to be familiar with the formalization of this model given in [FLP85]. We therefore do not repeat the definitions of terms already defined in that paper. Instead, we use boldface to indicate the first occurrence of a technical term borrowed from [FLP85]. When the definitions need to be adjusted to the new context (WGM instead of Consensus) we explain the necessary modifications.

[1] Most group membership services have such requirements. For example, the *Virtual Synchrony* property of ISIS requires the total ordering of multicasts with respect to view changes [BJ87].

Each process is modelled as an automaton. The **internal state** of each process p contains, among other things, an *input register* x_p and an *output register* y_p. The former holds a binary value indicating whether p initially wishes to leave the group G or not ($x_p = 1$ or $x_p = 0$, respectively). (In [FLP85] x_p indicates p's input value for Consensus.) The range of the output register y_p is a subset of $V_{init} = \{p_1, \ldots, p_n\}$. Initially, $y_p = V_{init}$, indicating that p's *first view of* G contains every process. After its initial setting, y_p may be written at most once. When (and if) p writes $V \subseteq V_{init}$ into y_p, we say that p *installs new view* V. (In [FLP85], the output register y_p is initially "blank" and is written at most once with process p's decision value of the Consensus.)

An **initial configuration** can be uniquely identified by a bit vector of length n that gives the value of x_{p_i} for each p_i, $1 \leq i \leq n$. Recall that WGM specifies the behavior of the system only when p_1, p_2 (or both) are the only processes that initially wish to leave G. Thus, the only initial configurations of interest are the ones corresponding to these three scenarios, namely, $C_{10} = \langle 100 \ldots 0 \rangle$, $C_{01} = \langle 010 \ldots 0 \rangle$, and $C_{11} = \langle 110 \ldots 0 \rangle$.

The WGM problem in asynchronous systems is formally defined as follows.

1. For every $C \in \{C_{10}, C_{01}, C_{11}\}$, and every **admissible run** that starts from C, there is at least one process that installs a new view, and no process installs a different new view.

2. Starting from C_{10} (respectively, C_{01}) there is at least one **run** such that some process installs new view $V_1 = V_{init} - \{p_1\}$ (respectively, $V_2 = V_{init} - \{p_2\}$).

Let C be a **configuration** and let $\mathcal{V}_C$ be the set of views installed in configurations **reachable** from C. Formally, $\mathcal{V}_C = \{V | \exists C', \exists p : C'$ is reachable from C, and in C' $y_p = V\}$. We say that C is *bivalent* if $|\mathcal{V}_C| \geq 2$. (In [FLP85], C is bivalent if the set of decision values of configurations reachable from C contains two elements.)

The impossibility proof of [FLP85] is structured as follows. It first shows that a bivalent initial configuration exists. Then, starting from that bivalent configuration, it constructs an admissible run in which no process decides — contradicting the termination requirement of Consensus (which requires that every admissible run be a deciding run). This line of argument is not directly applicable here because in WGM not every admissible run is required to terminate: only those that start from C_{10}, C_{01} or C_{11} must result in the installation of a new view. Thus, it is not sufficient to show that *some* initial configuration is bivalent, but that one of C_{10}, C_{01} or C_{11} is.

Lemma 1: C_{10}, C_{01}, or C_{11} is bivalent.

Proof: If C_{10} or C_{01} is bivalent, we are done. Now suppose that neither one of C_{10} or C_{01} is bivalent. We show that in this case C_{11} must be bivalent.

By the second property of WGM, there is a run that starts from C_{10} such that a process installs V_1. Consider any admissible run R that starts from C_{10} in which p_2 takes no **steps**. By the first property of WGM, some process installs a new view V in R. Since C_{10} is not bivalent, $V = V_1$. Let S be the **schedule** corresponding to R. Since S contains no steps of p_2 and the only difference between C_{10} and C_{11} is the value of p_2's input register, S is also **applicable** to C_{11}. Hence, from C_{11} there is a reachable configuration in which some process installs new view V_1. By a symmetric argument, from C_{11} there is a reachable configuration in which some process installs new view V_2. Therefore C_{11} is bivalent. □

Theorem 1: WGM is not solvable in asynchronous systems with crash failures, even if communication is reliable and at most one process may fail.

Proof (Sketch): Let C be a bivalent configuration in $\{C_{01}, C_{10}, C_{11}\}$ (by Lemma 1 such a C exists). We can now apply the techniques of [FLP85] to construct an infinite admissible run that starts from bivalent C and remains bivalent forever: In this run, no process ever installs a new view — a contradiction to the first requirement of WGM. □

4 Strengthening the Impossibility Result

Our specification of WGM allows processes to voluntarily request their removal from the group. This is a natural requirement which is compatible with the specification of the *Strong Group Membership Problem* (*S-GMP*) given in [RB91, Ric93, RB94]. In S-GMP a process p may execute event $faulty_p(q)$ indicating that p suspects that q crashed. If this occurs, S-GMP requires that eventually either p or q are excluded from the group view. The possibility that $p = q$ is explicitly allowed, and hence p executing $faulty_p(p)$ amounts to a request for self-removal.

In any case, self-removal is *not* necessary to our impossibility result. To avoid self-removals, we can simply: (1) reinterpret the meaning of the initial values, namely, p_1 has initial value 1 iff p_1 requests the removal of p_2 (say because p_1 suspects that p_2 has crashed), and p_2 has initial value 1 iff p_2 requests the removal of p_1; (2) redefine V_1 and V_2 to be $V_{init} - \{p_2\}$ and $V_{init} - \{p_1\}$, respectively. The impossibility proof does not change.

The first property of WGM requires that no two processes, whether correct or not, disagree on the new view. This is akin to the *Uniform Agreement* property of [NT90]. One may wonder whether our impossibility result hinges on this uniformity requirement. It is easy to show that WGM remains unsolvable even if we only require that *correct* processes do not disagree.

The second property of WGM postulates the existence of two runs, one installing new view $V_1 = V_{init} - \{p_1\}$, and the other installing $V_2 = V_{init} - \{p_2\}$. This is a reasonable requirement but it can be weakened without affecting the impossibility result. An examination of the proof shows that all that is necessary is the existence of two runs (one starting from C_{10} and one from C_{01}) that install *distinct* views, not necessarily V_1 and V_2.

5 Impossibility of WGM Despite Process Killing

Some group membership services have the ability to kill processes that are not faulty. For example, a process that is suspected of being faulty (say because it is very slow) can be removed from the group view and instructed to kill itself, even if this process is operational and did not actually fail. Alternatively, a process that has not succeeded in communicating with other processes for a long time, may decide to kill itself. It may appear that such "program-controlled" crashes of processes that hinder the progress of the algorithm make it easier for the remaining processes to install a new group view.

An example of program-controlled crashes appears in the *S-GMP* group membership protocol given in [RB91, Ric93]. In this protocol, if the process p in charge of coordinating the update of group views comes to believe that a majority of the processes are faulty, it executes an event called *crash*. This causes p to actually crash, i.e., to stop executing steps. This program-controlled crash is indistinguishable from a "genuine" crash that is due to, say, a hardware failure.

The model of [FLP85] does not quite capture such program-controlled crashes: in that model a process does not have the ability to stop taking steps if it wants to. Since we use the [FLP85] model to prove the impossibility of WGM (Theorem 1), one may ask whether our negative result still holds if program-controlled crashes are allowed.

To see why this question is pertinent, consider the proof of impossibility of WGM based on the [FLP85] model. This proof constructs an admissible run in which every process takes an infinite number of steps but never installs a new view. It is crucial for this construction that *every* process be capable of taking the next step at any point in time. This is true in the [FLP85] model, but is not true if we allow program-controlled crashes: once a process has executed a crash event, it cannot be required to take a next step. Thus, the argument of [FLP85] cannot be directly applied to systems that allow program-controlled crashes.

It turns out that the impossibility of WGM is not limited to the [FLP85] model: this negative result holds even if we allow program-controlled crashes. To show this, we must first model asynchronous distributed systems with program-controlled crashes. Let $\mathcal{M}$ denote the model of [FLP85]. Recall that in $\mathcal{M}$ each process is an automaton, and at most one process can stop executing steps at any state. Consider a model of computation denoted $\mathcal{M}_c$ that is identical to $\mathcal{M}$, except that the automaton associated with any process is now allowed to have a special state, called *die*, such that if a process enters that state, it stops executing steps. Thus, as in $\mathcal{M}$, model $\mathcal{M}_c$ allows at most one process to stop at any state (this models a genuine hardware crash), but unlike $\mathcal{M}$, in $\mathcal{M}_c$ any number of processes may stop at the *die* state (this models program-controlled crashes). Since $\mathcal{M}_c$ admits program-controlled crashes, the definition of admissible run is different from the one for $\mathcal{M}$: In $\mathcal{M}$, an infinite run is admissible if at most one process stops executing steps, and every message sent to any process that takes an infinite number of steps is received. In $\mathcal{M}_c$, an infinite run is admissible if at most one process stops executing steps *at a state different from die*, and every message sent to any process that takes an infinite number of steps is received.

Theorem 2: WGM is not solvable in asynchronous systems with crash failures, even if communication is reliable and program-controlled crashes are allowed.

Proof: We show that WGM is not solvable in $\mathcal{M}_c$. The proof is based on Theorem 1 which shows that WGM is not solvable in $\mathcal{M}$.

Let $\mathcal{A}_c$ be any algorithm in model $\mathcal{M}_c$. $\mathcal{A}_c$ consists of a set of automata, one for each process in the system. Transform these automata into automata of $\mathcal{M}$ as follows: replace each *die* state with a *nop* state, such that if a process p enters the *nop* state, p continues to take steps and in every subsequent step that it takes, regardless of the message it receives (including the "null" message), p remains in the *nop* state and does not send any messages. This transformation yields an algorithm $\mathcal{A}$ without *die* states, and so $\mathcal{A}$ is an algorithm in model $\mathcal{M}$.

We say that two runs are *congruent* if they start from the same initial configuration and each process installs a new view in one if and only if it installs the same view in the other. We claim that for each admissible run of $\mathcal{A}$ in $\mathcal{M}$ there is a congruent admissible run of $\mathcal{A}_c$ in $\mathcal{M}_c$, and vice-versa. From this claim and the specification of WGM, $\mathcal{A}_c$ solves WGM in $\mathcal{M}_c$ if and only if $\mathcal{A}$ solves WGM in $\mathcal{M}$. The theorem then follows immediately from Theorem 1. It now remains to show the claim.

Let r be any admissible run of $\mathcal{A}$ in $\mathcal{M}$. Modify r as follows: for each process p that enters state *nop*, replace the step causing p to enter *nop* for the first time into a step causing p to enter the state *die*, and discard all subsequent *nop* steps of p. It is easy to verify that the resulting run is an admissible run of $\mathcal{A}_c$ in $\mathcal{M}_c$ that is congruent to r.

Conversely, let r_c be any admissible run of $\mathcal{A}_c$ in $\mathcal{M}_c$. Let D be the set of processes that enter state *die* in r_c. Let r'_c be the shortest prefix of r_c such that all processes in D have entered *die*, and r''_c be the suffix of r_c that follows r'_c. Modify r_c as follows: In the prefix r'_c, for each process p in D, replace the step causing p to enter *die* into a step causing p to enter state *nop*. In r''_c, after each step, insert a sequence of steps, one for each process in D; the message received in each of these steps is the oldest one sent to the process taking the step, or the "null" message if no such message exists. It is easy to verify that the resulting run is an admissible run of $\mathcal{A}$ in $\mathcal{M}$ that is congruent to r_c.

This completes the proof of the claim and hence of the theorem. □

6 Discussion

A group membership service provides some kind of agreement among processes on the "current" membership of a group. This agreement requirement must be carefully specified so that two potentially conflicting goals are met:

1. It must be weak enough to be solvable.
2. It must be strong enough to simplify the design of fault-tolerant distributed applications. In particular, it should not be satisfied by trivial or useless protocols.

For primary-partition group membership services, i.e., those that maintain agreement on a *single* view of the group at any one time, the impossibility of WGM indicates that these two goals are incompatible in asynchronous systems with failures, even if communication is reliable and processes may only crash. It is important to note that *this impossibility result applies to group membership services that are allowed to remove from the group, and even kill, an arbitrary number of non-faulty processes that did not wish to be removed.* Thus, contrary to a widespread view [RB91, ADKM92, DKM93, DMS94, EMS95], the removal or killing of processes that are suspected to have crashed does not contribute to the solvability of the primary-partition group membership problem.

Note that the proof that WGM cannot be solved in asynchronous systems with failures hinges on the following liveness requirement of WGM: If p_1 or p_2 request to leave the group then a new view is eventually installed by at least one process in the system. Some implementations of primary-partition group membership do not satisfy this liveness requirement: They have runs that "block" forever, or remove or kill all processes.[2] It is not clear, however, what liveness property (if any) such implementations do satisfy [ACBMT95].

[2]For example, in the implementation of *S-GMP* [RB91, Ric93] it is possible for *all* processes to kill themselves before any new view is installed.

In a recent paper, Neiger proposes a primary-partition group membership specification that is weak enough to be solvable in asynchronous systems, and strong enough to prevent trivial implementations [Nei96]. With this specification, however, a single process crash is allowed to block all processes. So the group membership service that it defines is not fully fault-tolerant: when a crash occurs safety properties are guaranteed, but liveness can be lost.

The impossibility of WGM in asynchronous systems does not necessarily imply that one cannot implement a primary-partition group membership that works well "in practice" or "most of the time". After Fischer, Lynch and Paterson proved the impossibility of Consensus, researchers looked for ways to "circumvent" this negative result. First among these were the randomized algorithm of Ben-Or [Ben83], the probabilistic algorithm of Bracha and Toueg [BT83], and the algorithm by Rabin that assumes a shared sequence of random bits [Rab83]. These algorithms solve Consensus with probability 1: even though they admit non-terminating runs, such runs are shown to have probability 0, and Consensus is reached within a finite expected number of steps (for a survey of randomized Consensus algorithms see [CD89]). A different way to "circumvent" the impossibility of Consensus, is the addition of (unreliable) failure detectors to the asynchronous model of computation: [CT91] shows that Consensus can be solved using failure detectors that can make an infinite number of mistakes. Since the proof of the impossiblity of WGM is almost identical to the one for Consensus, it is very likely that the techniques that were used to "circumvent" the impossibility of Consensus can also be applied to WGM. In fact, we conjecture that it is possible to provide a realistic primary-partition group membership service, with precise liveness guarantees, if one uses randomization (as in [Ben83, Rab83]), or makes well-defined probability assumptions on the behavior of the environment (as in [BT83]), or uses failure detectors with well-defined properties (as in [CT91]).[3]

In contrast to primary-partition group membership services, partitionable ones allow processes to disagree on the current membership of the group, i.e., several different views of the membership of the group may evolve concurrently and independently from each other [JFR93, vRBC+93, BDGB94, DMS94, DMS95, EMS95]. In particular, there may be several disjoint subsets of processes such that processes in each subset agree that *they* are the current members of the group. In other words, such group membership services allow *group splitting* (e.g., when the network partitions) and *group merging* (e.g., when communication between partitions is restored).

By allowing disagreement, such group membership services escape from the impossibility result of this paper. However, they run into another fundamental problem: their specification must be strong enough to rule out useless group membership protocols, and yet it should be weak enough to remain solvable. The design of such a specification is still a subject of active research [ACBMT95, BDM95, DMS96].

[3] The group membership protocol of Melliar-Smith *et al.* makes probability assumptions on the environment [MSMA94].

Acknowledgements

We are grateful to Lorenzo Alvisi and Prasad Jayanti for their insightful comments on an earlier version of this paper. We also thank Roy Friedman, Gerard LeLann, Pascale Minet, and Robbert van Renessee for their helpful suggestions.

References

[ACBMT95] Emmanuelle Anceaume, Bernadette Charron-Bost, Pascale Minet, and Sam Toueg. On the formal specification of group membership services. Technical Report 95-1534, Computer Science Department, Cornell University, Ithaca, New York 14853, August 1995.

[ADKM92] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Membership algorithms for multicast communication groups. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG - 6), (LCNS, 647)*, pages 292–312, November 1992.

[BDGB94] Özalp Babaoğlu, Renzo Davoli, Luigi-Alberto Giachini, and Mary Gray Baker. *RELACS: a communications infrastructure for constructing reliable* *applications in large-scale distributed systems*. BROADCAST Project deliverable report, 1994. Department of Computing Science, University of Newcastle upon Tyne, UK.

[BDM95] Özalp Babaoğlu, Renzo Davoli, and Alberto Montresor. Failure detectors, group membership and view-synchronous communication in partitionable asynchronous systems (preliminary version). Technical Report UBLCS-95-18, Department of Computer Science, University of Bologna, Bologna, Italy, November 1995.

[Ben83] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 27–30, August 1983.

[BJ87] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th Annual Symposium Operating Systems Principles*, pages 123–138, Austin, TX, November 1987.

[BS94] Özalp Babaoğlu and André Schiper. On group communication in large scale distributed systems. In *Proceedings of the ACM SIGOPS European Workshop*, Dagstuhl, Germany, September 1994.

[BT83] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 12–26, August 1983. An extended and revised version appeared as "Asynchronous consensus and broadcast protocols" in the *Journal of the ACM*, 32(4):824-840, October 1985.

[CD89] Benny Chor and Cynthia Dwork. Randomization in byzantine agreement. *Advances in Computer Research (JAI Press Inc.)*, 4:443–497, 1989.

[Cri91] Flaviu Cristian. Reaching agreement on processor group membership in synchronous distributed systems. *Distributed Computing*, 4(4):175–187, April 1991.

[CT91] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for asynchronous systems. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 325–340. ACM Press, August 1991. To appear in the *Journal of the ACM*. An extended and revised version is also available by anonymous ftp from `ftp.cs.cornell.edu` in `pub/chandra/failure.detectors.algorithms.dvi.Z`

[DKM93] Danny Dolev, Shlomo Kramer, and Dalia Malki. Early delivery totally ordered multicast in asynchronous environment. In *Proceedings of the 23th Annual International Symposium on Fault-Tolerant Computing*, pages 544–553, Toulouse, June 1993.

[DMS94] Danny Dolev, Dalia Malki, and Ray Strong. An asynchronous membership protocol that tolerates partitions. Technical Report CS94-6, Institute of Computer Science, The Hebrew University of Jerusalem, 1994.

[DMS95] Danny Dolev, Dalia Malki, and Ray Strong. A framework for partitionable membership service. Technical Report CS95-4, Institute of Computer Science, The Hebrew University of Jerusalem, 1995.

[DMS96] Danny Dolev, Dalia Malki, and Ray Strong. A framework for partitionable membership service. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, Philadephia, PA, 1996.

[EMS95] Paul D. Ezhilchelvan, Raimundo A. Macêdo, and Santosh K. Shrivastava. Newtop: a fault-tolerant group communication protocol. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, Vancouver, BC, Canada, June 1995.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[HS95] Matti A. Hiltunen and Richard D. Schlichting. Properties of membership services. In *Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ, April 1995.

[JFR93] Farnam Jahanian, Sameh Fakhouri, and Ragunathan Rajkumar. Processor group membership protocols: specification, design and implementation. In *Proceeding of the 12th IEEE Symposium on Reliable Distributed Systems*, pages 2–11, Princeton, October 1993.

[KT91] M. Frans Kaashoek and Andrew S. Tanenbaum. Group communication in the amoeba distributed operating system. In *Proceedings of the 11th International Conference on Distributed Computer Systems*, pages 222–230, Arlington, TX, May 1991.

[MPS91] Shivakant Mishra, Larry L. Peterson, and Richard D. Schlichting. A membership protocol based on partial order. In *Proceedings of the IEEE International Working Conference on Dependable Computing For Critical Applications*, pages 137–145, Tucson, AZ, February 1991.

[MSMA94] P.M. Melliar-Smith, Louise Moser, and Vivek Agrawala. Processor membership in asynchronous distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):459–473, May 1994.

[Nei96] Gil Neiger. A new look at membership services. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, Philadephia, PA, 1996.

[NT90] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, September 1990.

[Rab83] Michael Rabin. Randomized Byzantine generals. In *Proceedings of the Twenty-Fourth Symposium on Foundations of Computer Science*, pages 403–409. IEEE Computer Society Press, November 1983.

[RB91] Aleta Ricciardi and Kenneth P. Birman. Using process groups to implement failure detection in asynchronous environments. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 341–352, 1991. Also available as technical report 93-1328, Department of Computer Science, Cornell University.

[RB94] Aleta Ricciardi and Kenneth P. Birman. Process membership in asynchronous environments. Available by anonymous ftp from `ftp.cs.cornell.edu` in `pub/aleta/AsyncMembService.ps`, April 1994.

[Ric93] Aleta Ricciardi. *The group membership problem in asynchronous systems.* PhD thesis, Department of Computer Science, Cornell University, USA, January 1993.

[vRBC+93] Robbert van Renesse, Kenneth P. Birman, Robert Cooper, Bradford Glade, and Patrick Stephenson. The horus system. In Kenneth P. Birman and Robbert van Renesse, editors, *Reliable Distributed Computing with the Isis Toolkit*, pages 133–147. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[VVR92] Paulo Verissimo, Werner Vogels, and Luis Rodrigues. Group orientation: a paradigm for modern distributed systems. In *Proceedings of the ACM SIGOPS 1992 Workshop*, Le Mont Saint-Michel, France, 1992.