Wait-free Byzantine Consensus

Paul Attie

College of Computer Science, Northeastern University and

MIT Laboratory for Computer Science

Keywords: distributed computing, fault tolerance, wait-freedom

1 Introduction

Byzantine consensus has traditionally been studied in the context of message passing models of distributed computing, with various asynchrony/timing assumptions on message transit times and/or process speeds [2, 4]. In an asynchronous shared memory model, Byzantine consensus has been considered "uninteresting" because a single Byzantine process can repeatedly corrupt the entire memory, thereby overwhelming any consensus algorithm. One way to address this problem is to impose assumptions on the relative speeds of processes, even Byzantine ones, so that a process can do only a limited amount of damage within a fixed time interval. Another possibility is to limit the amount of memory that each process can access, e.g., through operating system level mechanisms such as access control matrices, access control lists, or capability lists [8, 10, 11]. We consider the latter approach in this paper.

A consensus protocol must have some degree of fault-tolerance (otherwise the problem is trivial). Among the attributes of a fault are: (1) the entity suffering the fault, e.g., the memory itself, or a process that accesses the memory, and (2) the type of the fault, e.g., crash (entity stops responding), omission (entity responds to only some inputs), or Byzantine (entity behaves arbitrarily). We restrict attention in this paper to faults that affect the processes, and assume that shared memory is reliable. A particularly useful type of fault-tolerance is *wait-freedom* [5]. A principal advantage of wait-free protocols is that the progress of each participating process is independent of that of the other processes. In particular, a process can progress even if all other processes crash. Thus, a wait-free protocol with n processes tolerates up to $n-1$ crash failures.

In this paper, we study the problem of devising wait-free shared memory consensus protocols that tolerate Byzantine processes (i.e., processes that have suffered a Byzantine fault). We show that weak wait-free Byzantine consensus can be achieved, but only by using "nonresettable" shared objects, that is, objects whose state cannot be reset to an initial state (by any sequence of permissible operations). With resettable objects, even a single Byzantine process is enough to prevent any solution to weak wait-free Byzantine consensus. Our impossibility result holds even if we can limit every process (including Byzantine processes) to invoking objects in a particular

fixed set (i.e., impose access-control). Although this limited-access model restricts the amount of damage a Byzantine process can do, the restriction is not powerful enough to enable a solution using resettable objects.

In the technical report version of this paper [1] we also show that wait-free Byzantine consensus cannot be achieved at all, in the presence of even one Byzantine process, and we give a straightforward protocol for weak wait-free Byzantine consensus that uses a nonresettable object (a sticky register—a generalization of Plotkin's sticky bit [9] to more than three values).

Related work. Malkhi et al. [3] consider shared-object systems in which the processes are subject to Byzantine faults. They provide universal constructions and a consensus algorithm. None of their constructions and algorithms are wait-free. They also exploit the idea of limiting the access of each object to a specified part of memory by using access control lists. Jayanti et al. [6] studies the "dual" problem of implementing wait-free shared objects in a setting where processes are reliable but shared objects are subject to faults (including arbitrary responses): they exploit redundancy by implementing a single shared object using many "internal" shared objects.

The paper is as follows. Section 2 defines our model of computation. Section 3 defines the wait-free Byzantine consensus problem and its weak variant. Section 4 presents our impossibility result for weak wait-free Byzantine consensus. Section 5 discusses directions for further research and concludes.

2 Model of Computation

A *concurrent program* $P = (P_1 \parallel \cdots \parallel P_n, O_1, \ldots, O_m, \mathcal{A})$ consists of n sequential *processes* that execute concurrently and interact by performing operations on *shared data objects* $O_1, \ldots, O_m$. An object O_ℓ is a nondeterministic state-machine with a nonempty set of initial local states, and transitions of the form $(v_\ell, \texttt{op}, \texttt{resp}, w_\ell)$: when in local state v_ℓ, object O_ℓ can respond to an invocation of operation `op` (by some process) by moving to local state w_ℓ and producing the response `resp`. The local state of an object is not visible to the processes except via its responses to operation invocations. Each shared object O_ℓ has an *interface*: a defined set of legal operations that can be invoked on O_ℓ. We require that, from any local state, a transition exists for every legal operation (c.f., the input-enabling assumption of I/O automata [7, chapter 13]). We also assume that all objects are reliable, i.e., that they always behave in accordance with their transition relation.

A process P_i is a nondeterministic state-machine with a nonempty set of initial local states, and two types of transitions. An *internal transition* (s_i, t_i) takes P_i from local state s_i to local state t_i. An *invocation/response transition* $(s_i, O_\ell, \texttt{op}, \texttt{resp}, t_i)$ takes P_i from local state s_i to local state t_i, provided that `resp` is the response that object O_ℓ returns for the invocation of operation `op`. The local state of a process is not visible to other processes. A *global state* is a tuple $\langle s_1, \ldots, s_n, v_1, \ldots, v_m \rangle$,

where s_i, $1 \leq i \leq n$, is the local state of process P_i, and v_ℓ, $1 \leq \ell \leq m$, is the local state of object O_ℓ. *Global transitions* are of two types, as follows: (1) If $\langle s_1, \ldots, s_i, \ldots, s_n, v_1, \ldots, v_m \rangle$ is a global state, and (s_i, t_i) is an internal transition of P_i, then $\langle s_1, \ldots, t_i, \ldots, s_n, v_1, \ldots, v_m \rangle$ is a possible next global state, and (2) If $\langle s_1, \ldots, s_i, \ldots, s_n, v_1, \ldots, v_\ell, \ldots, v_m \rangle$ is a global state, $(s_i, O_\ell, \texttt{op}, \texttt{resp}, t_i)$ is an invocation/response transition of P_i, and $(v_\ell, \texttt{op}, \texttt{resp}, w_\ell)$ is a transition of O_ℓ, then $\langle s_1, \ldots, t_i, \ldots, s_n$, $v_1, \ldots, w_\ell, \ldots, v_m \rangle$ is a possible next global state. In either case, we say that P_i makes a *step*, and that this step *takes state s to state t*. Thus, a step of process P_i is either an internal transition (s_i, t_i) of P_i, or an invocation/response transition $(s_i, O_\ell, \texttt{op}, \texttt{resp}, t_i)$ of P_i.

$\mathcal{A} \subseteq \{P_1, \ldots, P_n\} \times \{O_1, \ldots, O_m\}$ is an *access pattern*: P_i can perform operations on O_ℓ only if $(P_i, O_\ell) \in \mathcal{A}$. This access restriction remains in force even if P_i is Byzantine. A *nonfaulty process* is one that behaves according to its transition relation. A *faulty process* is one that behaves arbitrarily, i.e., it can invoke any operation on any object that it has access to (according to $\mathcal{A}$). If the operation is not legal, according to the object's interface, then we assume that the object does not change state. Also, even a Byzantine process cannot invoke operations on objects that it does not have access to. We assume that such invocations would be blocked by, e.g., the operating system, and so the targeted object would be unaffected. We extend our definition of process step to allow for such arbitrary steps if the process is faulty. We also assume that a Byzantine process cannot change the code of another process. Since process code also resides in memory, this assumption is necessary. Without it, a single Byzantine process can corrupt the entire system.

An *execution fragment* is a (finite or infinite) alternating sequence of global states and steps (starting with a state) such that each step takes the state preceding it to the state following it. An *execution* is an execution fragment that starts in an initial global state. Hence, we model concurrency by the nondeterministic interleaving of internal transitions, executed by a single process, and invocation/response transitions, executed jointly by one process and one object. A *reachable state* is a state lying on some execution.

Note that objects have "high" atomicity in that invocations and responses occur "together" in one atomic transition. Our impossibility results carry over to a model in which invocations and responses are separate events, since such a model would still admit executions in which every invocation is immediately followed by its matching response. Hence, the proofs in this paper would still apply, since they only rely on the ability to construct certain executions.

An infinite execution is *fair* if and only if every nonfaulty process takes an infinite number of steps in the execution. We assume (without further statement) that every infinite execution is fair. We also assume that a nonfaulty process can always take a step, i.e, is always enabled. This is reasonable, since our model does not permit operations such as "await some condition," which hide busy waiting or conditional waiting within their implementation. Any busy waiting or conditional

waiting must be programmed explicitly into the transition relation of a process, e.g., as a repeated "polling" of some object until a particular response is observed. Hence, in our model, if a process enters a local state from which it currently has no enabled actions, then the process is stuck forever in that state.

2.1 Notation and Technical Definitions

If α is an execution, then $\alpha|P_i$ is obtained by taking the subsequence of steps that P_i executes along α. Also, $\alpha|O_\ell$ is the subsequence of steps along α that involve O_ℓ. This extends to a set of objects in the obvious way: $\alpha|(O_1, \ldots, O_\ell)$ is the subsequence of steps along α that each involve one object in $O_1, \ldots, O_\ell$. Finally, these notations can be nested, so that $(\alpha|P_i)|O_\ell$ denotes the subsequence of all process P_i's steps along α that involve O_ℓ (i.e., that arise from an invocation by P_i on O_ℓ). If $s = \langle s_1, \ldots, s_n, v_1, \ldots, v_m \rangle$ is a global state, then $s(P_i)$ is s_i, i.e., P_i's local state in s, and $s(O_\ell)$ is v_ℓ, i.e., O_ℓ's local state in s. If s and t are two global states, we say $s \overset{i}{\sim} t$ if and only if $s(P_i) = t(P_i)$ and $s(O_\ell) = t(O_\ell)$ for all objects O_ℓ such that $(P_i, O_\ell) \in \mathcal{A}$, i.e., all objects that P_i has access to.

3 The Wait-free Byzantine Consensus problem

We specify the wait-free Byzantine consensus problem as follows. Each process starts with an initial value from a fixed set V (part of its initial local state), and each nonfaulty process eventually decides on some value in V. The correctness requirements are as follows (agreement and validity are from [7, chapter 6]):

Agreement No two nonfaulty processes decide on different values.

Validity If all nonfaulty processes start with the same initial value $val \in V$, then val is the only possible decision value for a nonfaulty process.

Wait-free Termination If P_i is nonfaulty in an infinite fair execution α, then P_i decides exactly once in α.

Uniform-initial-state Every combination of initial local state for each process and initial local state for each shared object is a possible initial global state.

We impose the uniform-initial-state requirement in order to rule out trivial solutions in which each process has a "shared" object that only it can access, and whose initial value gives a correct decision value. The wait-free termination requirement embodies the wait-freedom aspect of the problem, since it requires all nonfaulty processes to decide, regardless of how many other processes fail. In the sequel, we assume without loss of generality that $\{0, 1\} \subseteq V$.

We extend the definition of a process so that a subset of the internal transitions are *decision transitions*. A decision transition decides on some value in V.

Proposition 1 *The wait-free termination requirement implies:*

if P_i is a nonfaulty process and s is a reachable global state in which P_i has not yet decided, then there exists an execution fragment α starting in s such that:

1. *α consists only of steps of P_i, and*
2. *P_i decides in α.*

Proof. Let P_i be a nonfaulty process, and let s be an arbitrary reachable state in which P_i has not decided. Since a nonfaulty process is (by assumption) always enabled, s lies along at least one infinite execution. Since one possible behavior of a Byzantine process is to emulate a nonfaulty process up to some point, and then do nothing, there exists an infinite execution α' containing s such that α' can be split into two parts: a prefix α'' ending in s such that every process except P_i is faulty in α'', and a suffix α''' starting from s, such that no process other than P_i takes a step along α'''. By the wait-free termination requirement, P_i decides along α'''. □

We also define the weak wait-free Byzantine consensus problem, in which the Validity requirement is replaced by the Weak Validity requirement:

Weak Validity If there are no faulty processes and all processes start with the same initial value $val \in V$, then val is the only possible decision value.

4 Impossibility of Weak Wait-free Byzantine Consensus using Resettable Objects

We show that weak wait-free Byzantine consensus cannot be solved when each of the shared objects may be reset to some initial state. We define:

Definition 1 *An object is* resettable *if, for each of its reachable local states, there exists a sequence of operations that takes the object back to some initial local state. We call such a sequence of operations a* reset sequence.

Note that our definition allows an object to be reset to several initial states from a given reachable state v. The object cannot however, be reset to an initial state that is not reachable from v. An extreme case is when every initial state is reachable from v, and so the object could be reset to any initial state (from v).

Resettable objects are more desirable than nonresettable objects, because they can be easily reused (e.g., for successive instances of consensus).

We show that there is no solution to weak wait-free Byzantine consensus using only resettable objects, even if only a single process is Byzantine.

Proposition 2 *Let $P = (P_1 \parallel \cdots \parallel P_n, O_1, \ldots, O_m, \mathcal{A})$ be a protocol for weak wait-free Byzantine consensus. In any execution of P in which P_i, $1 \leq i \leq n$, runs alone and decides before any other process takes a single step, P_i must decide its own initial value.*

Proof. Let s be an arbitrary initial global state, and let α_s be an arbitrary execution starting in s in which P_i first runs alone until it decides. Let the initial value of P_i in s be val_i. Let t be the global initial state which is identical to s except that all processes have initial value val_i (thus all shared objects have the same initial local states in s as in t). t exists by the uniform-initial-state requirement. Starting in state t, it is possible for P_i to execute the same sequence of steps (up to the point that it decides) that it executes in α_s, since P_i's local state and all object states are the same in s as in t. Let α_t be the resulting execution, and furthermore assume that no process is faulty in α_t (it is clear that such an α_t exists). By weak validity, P_i decides val_i in α_t. Hence, P_i also decides val_i in α_s since it executes the same steps in both executions, up to the point that it decides. Since α_s was chosen arbitrarily, the proposition follows. □

We first establish the impossibility result for three processes. The generalization to n processes is easily done using the wait-free termination and uniform-initial-state requirements.

Theorem 3 *There is no solution to the weak wait-free Byzantine consensus problem for three processes if one of them can be Byzantine and all shared objects are resettable.*

Proof. By contradiction. Suppose a solution $P = (P_1 \parallel P_2 \parallel P_3, O_1, \ldots, O_m, \mathcal{A})$ exists. Let $\mathcal{O}_{ij} \subseteq \{O_1, \ldots, O_m\}$ be the (possibly empty) set of objects that can be accessed by P_i and P_j, and no other process, $i, j \in \{1, 2, 3\}, i \neq j$ (all according to the access pattern $\mathcal{A}$). Also let $\mathcal{O}_{123} \subseteq \{O_1, \ldots, O_m\}$ be the (possibly empty) set of objects that can be accessed by P_1, P_2, and P_3, again according to $\mathcal{A}$. Thus, these sets of objects are all pairwise disjoint. We construct two executions α_1 and α_2 of P as follows (Figure 1 shows how α_1 and α_2 are "alternately" constructed from each other in a well-defined manner).

The first part of α_1 is as follows. P_1 is Byzantine, P_2 and P_3 have initial value 1, and $\mathcal{O}_{23}$ is in some initial state $\mathcal{V}_{23}$.[1] First, P_2 runs alone until it decides. By Proposition 2, P_2 decides 1. Next, P_1 invokes reset sequences for all objects in $\mathcal{O}_{13}$ and $\mathcal{O}_{123}$. Let the resulting initial states of $\mathcal{O}_{13}, \mathcal{O}_{123}$ be $\mathcal{V}_{13}, \mathcal{V}_{123}$ respectively.

[1] For brevity, we discuss the sets of objects $\mathcal{O}_{ij}, \mathcal{O}_{123}$ as if they were single objects. For example, $\mathcal{V}_{23}$ is actually a tuple of initial object states.

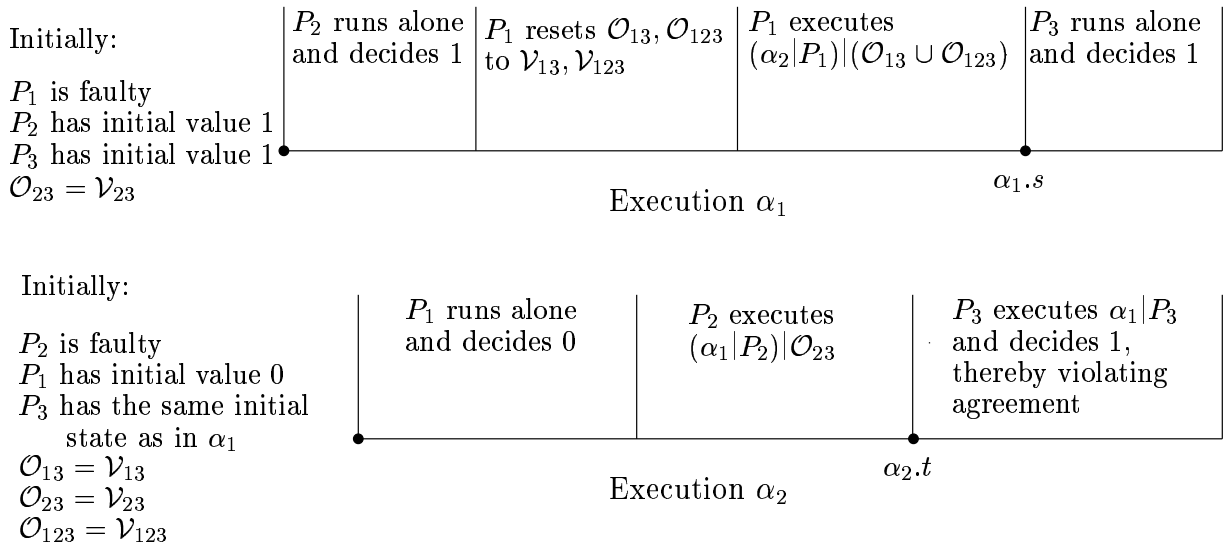

Figure 1: Executions α_1 and α_2 in the proof of Theorem 3.

α_2 is as follows. P_2 is Byzantine, P_1 is nonfaulty and has initial value 0, and P_3 is nonfaulty and has the same initial state as in α_1 (and therefore has initial value 1). The initial states of $\mathcal{O}_{13}, \mathcal{O}_{23}, \mathcal{O}_{123}$ are $\mathcal{V}_{13}, \mathcal{V}_{23}, \mathcal{V}_{123}$ respectively. By the uniform-initial-state requirement, these initial conditions must exist for some execution. First, P_1 runs alone until it decides. By Proposition 2, P_1 decides 0. Next, P_2 executes $(\alpha_1|P_2)|\mathcal{O}_{23}$ (since P_2 is Byzantine and has access to $\mathcal{O}_{23}$, it can invoke any sequence of operations on $\mathcal{O}_{23}$). Let the global state at this point be $\alpha_2.t$. Finally, P_3 runs alone until it decides. By agreement, P_3 must decide 0, since P_1 decided 0, and P_1, P_3 are nonfaulty.

The last part of α_1 is as follows. P_1 executes $(\alpha_2|P_1)|(\mathcal{O}_{13} \cup \mathcal{O}_{123})$—since P_1 is Byzantine and has access to $\mathcal{O}_{13} \cup \mathcal{O}_{123}$, it can execute any sequence of operations on $\mathcal{O}_{13} \cup \mathcal{O}_{123}$. Let the global state at this point be $\alpha_1.s$. Finally, P_3 runs alone until it decides. By agreement, P_3 must decide 1, since P_2 decided 1, and P_2, P_3 are nonfaulty.

We now establish $\alpha_1.s \overset{3}{\sim} \alpha_2.t$. In $\alpha_1.s$, the state of $\mathcal{O}_{23}$ results from the sequence $(\alpha_1|P_2)|\mathcal{O}_{23}$ applied to the initial state $\mathcal{V}_{23}$ of $\mathcal{O}_{23}$ (by definition of $|$). In $\alpha_2.t$, the state of $\mathcal{O}_{23}$ results from the same sequence, namely $(\alpha_1|P_2)|\mathcal{O}_{23}$, applied to the same initial state $\mathcal{V}_{23}$ (by construction of α_2). Hence, we can select the local transitions of all the objects in $\mathcal{O}_{23}$ so that $\alpha_1.s(\mathcal{O}_{23}) = \alpha_2.t(\mathcal{O}_{23})$.

In $\alpha_1.s$, the states of $\mathcal{O}_{13}, \mathcal{O}_{123}$ result from the sequence $(\alpha_2|P_1)|(\mathcal{O}_{13} \cup \mathcal{O}_{123})$ applied to the states $\mathcal{V}_{13}, \mathcal{V}_{123}$ respectively (because of the reset sequences invoked by P_1 in α_1, the actual initial states of $\mathcal{O}_{13}, \mathcal{O}_{123}$ don't matter). In $\alpha_2.t$, the states of $\mathcal{O}_{13}, \mathcal{O}_{123}$ also result from the sequence

$(\alpha_2|P_1)|(\mathcal{O}_{13} \cup \mathcal{O}_{123})$ applied to the (initial) states $\mathcal{V}_{13}, \mathcal{V}_{123}$ respectively (by definition of $|$). Hence we can choose the local transitions of all the objects in $\mathcal{O}_{13} \cup \mathcal{O}_{123}$ so that $\alpha_1.s(\mathcal{O}_{13}) = \alpha_2.t(\mathcal{O}_{13})$ and $\alpha_1.s(\mathcal{O}_{123}) = \alpha_2.t(\mathcal{O}_{123})$. Finally, the local state of P_3 in $\alpha_1.s$ is its initial state in α_1, since P_3 takes no steps in the prefix of α_1 up to $\alpha_1.s$. Likewise the local state of P_3 in $\alpha_2.t$ is its initial state in α_2. But these initial states are the same, by construction of α_2. Hence $\alpha_1.s(P_3) = \alpha_2.t(P_3)$. Since P_3's state and the states of all objects that P_3 can access are the same in $\alpha_1.s$ as in $\alpha_2.t$, we conclude $\alpha_1.s \overset{3}{\sim} \alpha_2.t$.

In α_1, P_3 runs alone from $\alpha_1.s$ until it decides. In α_2, P_3 runs alone from $\alpha_2.t$ until it decides. Since $\alpha_1.s \overset{3}{\sim} \alpha_2.t$, and P_3 runs alone from $\alpha_1.s$, $\alpha_2.t$ until it decides, it is possible for P_3 to execute the same sequence (i.e., $\alpha_1|P_3$) of steps in α_2 as in α_1. Let this then be the sequence that P_3 actually executes in α_2. Hence, P_3 must decide the same value in α_2 as in α_1. But we showed above that P_3 decides 1 in α_1 and 0 in α_2. Hence the desired contradiction. □

Theorem 4 *There is no solution to the weak wait-free Byzantine consensus problem for $n \geq 3$ processes if one of them can be faulty and all shared objects are resettable.*

Proof. We assume a solution exists and derive a contradiction exactly as in the proof of Theorem 3. From the wait-free termination and uniform-initial-state requirements, and Proposition 1, it is clear that the executions α_1, α_2 (with their initial states extended arbitrarily to $P_4, \ldots, P_n$) constructed in the aforementioned proof are executions of a solution for any $n \geq 3$ (simply delay $P_4, \ldots, P_n$ until α_1, α_2 have been fully executed). □

We note that if an access pattern is not imposed, then the proof of Theorem 3 becomes trivial: let P_1 be nonfaulty and run alone until it decides; then let P_2 be faulty and reset all objects in the system; finally let P_3 be nonfaulty and run alone until it decides. Invoking Proposition 2 then allows us to conclude that P_1 and P_3 must both decide on their initial values, since P_3's view of the execution is indistinguishable from its view of some execution in which it runs alone and decides before any other process takes a step.

5 Further Work and Conclusions

We studied the problem of wait-free consensus in the presence of Byzantine faults. We showed that weak wait-free Byzantine consensus is achievable only if we use nonresettable objects (and, in the technical report version of this paper, that wait-free Byzantine consensus is not achievable at all). Our impossibility results hold in the presence of only a single Byzantine process, and even if we can impose an "access pattern" that, for each process, fixes the objects that the process has access to. Our results suggest that the "majority voting" considerations that give the usual bound of less than one-third faulty processes ($n \geq 3f + 1$) do not come into play when wait-freedom is

required. We also remark that our results are easily extended to low-atomicity objects, i.e., where invocations and responses are separate events: simply construct the executions used in the proofs above so that every response immediately follows its matching invocation.

The access patterns considered in this paper either grant a process access to all of the operations defined by a particular shared object, or to none of them. A more refined notion would grant different processes access to different subsets of the defined operations, e.g., like an access matrix [8, 10]. Thus, some processes may only have access to "read" operations. The proof ideas we used in this paper do not carry over to this more refined model. It would be worthwhile to investigate whether our impossibility results themselves do carry over.

As mentioned in the introduction, the issue of Byzantine consensus in a shared memory system is rendered nontrivial by introducing some restriction on the amount of damage a Byzantine process can inflict. We considered a restriction based on access control in this paper. Other restrictions could be based on timing, so that a Byzantine process can only inflict a limited amount of damage within a fixed time interval (i.e., the "rate of damage" is finite). A first step in further investigation is to categorize the restrictions on a Byzantine process that could be reasonably imposed in practice, and then to study the existence (or not) of wait-free consensus, or consensus generally, c.f. [3], subject to one (or more) of the restrictions.

References

[1] P. C. Attie. Wait-free byzantine agreement. Technical Report NU-CCS-00-02, College of Computer Science, Northeastern University, Boston, Massachusetts, May 2000. Available on-line at http://www.ccs.neu.edu/home/attie/pubs.html.

[2] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41(1):122–152, Jan. 1994.

[3] M. Reiter D. Malkhi, M. Merritt and G. Taubenfeld. Objects shared by byzantine processes. In *Proceedings of the 14th International Symposium on DIStributed Computing (DISC 2000)*, Toledo, Spain, Oct. 2000.

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

[5] M. Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 11(1):124–149, Jan. 1991.

[6] P. Jayanti, T.D. Chandra, and S. Toueg. Fault-tolerant wait-free shared objects. *J. ACM*, 45(3):451–500, May 1998.

[7] N. A. Lynch. *Distributed Algorithms.* Morgan-Kaufmann, San Francisco, California, USA, 1996.

[8] C.P. Pfleeger. *Security in Computing.* Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1989.

[9] S. Plotkin. Sticky bits and the universality of consensus. In *8'th ACM Symposium on the Principles of Distributed Computing (PODC)*, Edmonton, Alberta, Canada, Aug. 1989.

[10] A. Silberschatz and P. Galvin. *Operating System Concepts.* Addison-Wesley, Reading, Massachusetts, USA, 1994.

[11] A. S. Tanenbaum. *Operating Systems, Design and Implementation.* Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1987.