Byzantine Agreement with Homonyms

Carole Delporte-Gallet [*]
University Paris Diderot

Hugues Fauconnier
University Paris Diderot

Rachid Guerraoui
Ecole Polytechnique Fédérale de Lausanne

Anne-Marie Kermarrec
INRIA Rennes-Bretagne Atlantique

Eric Ruppert
York University

Hung Tran-The
University Paris Diderot

ABSTRACT

So far, the distributed computing community has either assumed that all the processes of a distributed system have distinct identifiers or, more rarely, that the processes are anonymous and have no identifiers. These are two extremes of the same general model: namely, n processes use ℓ different authenticated identifiers, where $1 \leq \ell \leq n$. In this paper, we ask how many identifiers are actually needed to reach agreement in a distributed system with t Byzantine processes.

We show that having $3t + 1$ identifiers is necessary and sufficient for agreement in the synchronous case but, more surprisingly, the number of identifiers must be greater than $\frac{n+3t}{2}$ in the partially synchronous case. This demonstrates two differences from the classical model (which has $\ell = n$): there are situations where relaxing synchrony to partial synchrony renders agreement impossible; and, in the partially synchronous case, increasing the number of *correct* processes can actually make it harder to reach agreement. The impossibility proofs use the fact that a Byzantine process can send multiple messages to the same recipient in a round. We show that removing this ability makes agreement easier: then, $t + 1$ identifiers are sufficient for agreement, even in the partially synchronous model.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*

General Terms

Algorithms, Reliability, Theory

Keywords

agreement, Byzantine failures, consensus, identifiers, lower bounds, synchrony

[*]This work is partially supported by the ERC Starting Grant project 204742 and the ANR VERSO SHAMAN.

1. INTRODUCTION

We consider a distributed system in which ℓ distinct identifiers are assigned to n processes, where $1 \leq \ell \leq n$. Several processes may be assigned the same identifier, in which case we call the processes *homonyms*. The identifiers are authenticated: if a process p receives a message from a process q with identifier i, p knows that the message was not sent by a process with identifier $i' \neq i$, but p does not know whether the message was sent by q or another process q' having the same identifier i. A process cannot direct a message it sends to a particular process, but can direct the message to all processes that have a particular identifier.

This model generalizes the classical scheme where processes have distinct identifiers (i.e., $\ell = n$), and the less classical scheme where processes are anonymous (i.e., $\ell = 1$). Studying systems with homonyms provides a better understanding of the importance of identifiers in distributed computing, and there are two additional motivations for the new model. Firstly, assuming in systems such as Pastry or Chord [19, 22] that all processes have unique (unforgeable) identifiers might be too strong an assumption in practice. We may wish to design protocols that still work if, by a rare coincidence, two processes are assigned the same identifier. This approach is also useful if security is breached and a malicious process can forge the identifier of a correct process, for example by obtaining the correct process's private key. Secondly, in many cases, users of a system may wish to preserve their privacy by remaining anonymous. However, in a fully anonymous system where no identifiers are used, very few problems are solvable. (In particular, Okun observed that Byzantine agreement is impossible in the fully anonymous model [15], even with a single faulty process.) With a limited number of identifiers, more problems become solvable, and some level of anonymity can be preserved by hiding, to some extent, the association between users and identifiers. For example, users of a distributed protocol might use only their domain names as identifiers. Others will see that some user within the domain is participating, but will not know exactly which one. If several users within the same domain participate in the protocol, they will behave as homonyms.

We ask in this paper how many distinct identifiers are needed to reach *agreement* in a system of n processes, up to t of which can be Byzantine. We need only to consider systems where $n > 3t$: this assumption is known to be a requirement for solving Byzantine agreement, even when $\ell = n$ [14, 18], and it thus applies also for systems with homonyms. For the synchronous case, we prove using a scenario argument that $3t + 1$ identifiers are necessary. The

	Synchronous	Partially synchronous
Innumerate processes	$\ell > 3t$	$\ell > \frac{n+3t}{2}$
Numerate processes	$\ell > 3t$ ($\ell > t$ for restricted Byzantine processes)	$\ell > \frac{n+3t}{2}$ ($\ell > t$ for restricted Byzantine processes)

Table 1: Necessary and sufficient conditions for solving Byzantine agreement in a system of n processes using ℓ identifiers and tolerating t Byzantine failures. In all cases, n must be greater than $3t$.

matching synchronous algorithm is obtained by a simulation that transforms any synchronous Byzantine agreement algorithm designed for a system with unique identifiers to one that works in a system with $\ell > 3t$ identifiers. For the partially synchronous case, we prove using a partitioning argument that the lower bound becomes $\ell > \frac{n+3t}{2}$. (Note that $\frac{n+3t}{2}$ is strictly greater than $3t$ because $n > 3t$.) We show that this bound is also tight by giving a new partially synchronous Byzantine agreement algorithm. This bound is somewhat surprising because the number of required identifiers ℓ depends on n as well as t. Counter-intuitively, increasing the number of correct processes can render agreement impossible. For example, if $t = 1$ and $\ell = 4$, agreement is solvable for 4 processes but not for 5. Another difference from the classical situation (where $\ell = n$) is that the condition that makes Byzantine agreement solvable is different for the synchronous and partially synchronous models.

To strengthen our results, we show that (a) both the synchronous and partially synchronous lower bounds hold even if correct processes are *numerate*, i.e., can count the number of processes that send identical messages in a round and (b) the matching algorithms are correct even if processes are *innumerate*. In systems with unique identifiers, senders can append their identifier to all messages, making it trivial for the receiver to count copies of messages. This is not possible in systems with homonyms, so the distinction between numerate and innumerate processes is important.

What has more impact, however, is the ability for a Byzantine process to send multiple messages to a single recipient in a round. In a classical system with unique identifiers, the Byzantine process has no advantage in doing this: algorithms could simply discard such messages. In systems with homonyms, there is a clear advantage. In fact, we prove that if each Byzantine process is restricted to sending a single message per round to each recipient (and processes are numerate), then $t + 1$ identifiers are enough to reach agreement even in a partially synchronous model. We also show this bound is tight using a valency argument: $t + 1$ identifiers are needed even in the synchronous case. The fact that $t + 1$ identifiers are sufficient to reach agreement with restricted Byzantine processes has some practical relevance: In some settings, it is reasonable to assume that Byzantine processes are simply malfunctioning ordinary processes sending incorrect messages, and not malicious processes with the additional power to generate and send more messages than correct processes can.

The results are summarized in Table 1. Section 2 describes our models and recalls the specification of Byzantine agreement. Section 3 considers the synchronous case and Section 4 considers the partially synchronous one. Section 5 gives our results for restricted Byzantine processes. Section 6 provides some concluding remarks. Due to space limitations, details of some proofs and algorithms appear in [8].

2. DEFINITIONS

We consider a distributed message-passing system with $n \geq 2$ processes. Each process has an authenticated identifier from the set $\mathcal{L} = \{1, ..., \ell\}$. We assume that $n \geq \ell$ and that each identifier is assigned to at least one process. Thus, the parameter ℓ measures the number of different identifiers that are actually assigned to processes. In the case where $n > \ell$, one or more identifiers will each be shared by several processes. In the case where $\ell = 1$, all processes have the same identifier, and they are therefore anonymous. We assume algorithms are deterministic. Thus, the actions of a process are entirely determined by the process's initial state and the messages it receives. Processes with the same identifier execute the same code but processes with different identifiers may behave differently. In our proofs, we sometimes refer to individual processes using names like p, but these names cannot be used by the processes themselves in their algorithms.

A *correct* process does not deviate from its algorithm specification. A process that is not correct is called *Byzantine*. The maximum possible number of Byzantine processes is denoted t (where $0 < t < n$). We need only consider systems where $n > 3t$: this assumption is known to be a requirement for solving Byzantine agreement, even when $\ell = n$ [14, 18], and it thus also applies to systems with homonyms. A Byzantine process may choose to send arbitrary messages (or no message) to each other process. However, we assume Byzantine processes cannot forge identifiers: each message is authenticated with its sender's identifier. Given a message m, we denote by $m.val$ its *value* (or content) and by $m.id$ the identifier of the sender. If a correct process receives m, then at least one process p with identifier $m.id$ sent m.

In the *synchronous model*, computation proceeds in rounds. In each round, each process can send a message to each other process and then receive all messages that were sent to it during that round.

For the *partially synchronous model* we use the definition of Dwork, Lynch and Stockmeyer [10]: computation proceeds in rounds, as in the synchronous model, except that in each execution, a finite number of messages might not be delivered to all of their intended recipients. There is no bound on the number of messages that can be dropped. As argued in [10], this basic partially synchronous model is equivalent to other models with partially synchronous communication. More specifically, the model in which message delivery times are eventually bounded by a known constant and the model in which message delivery times are always bounded by an unknown constant can both simulate the basic partially synchronous model. Conversely, each of these models can be simulated by the basic partially synchronous model. Thus, our characterization of the values of n, ℓ and t for which Byzantine agreement can be solved applies to the other models with partially synchronous communication too.

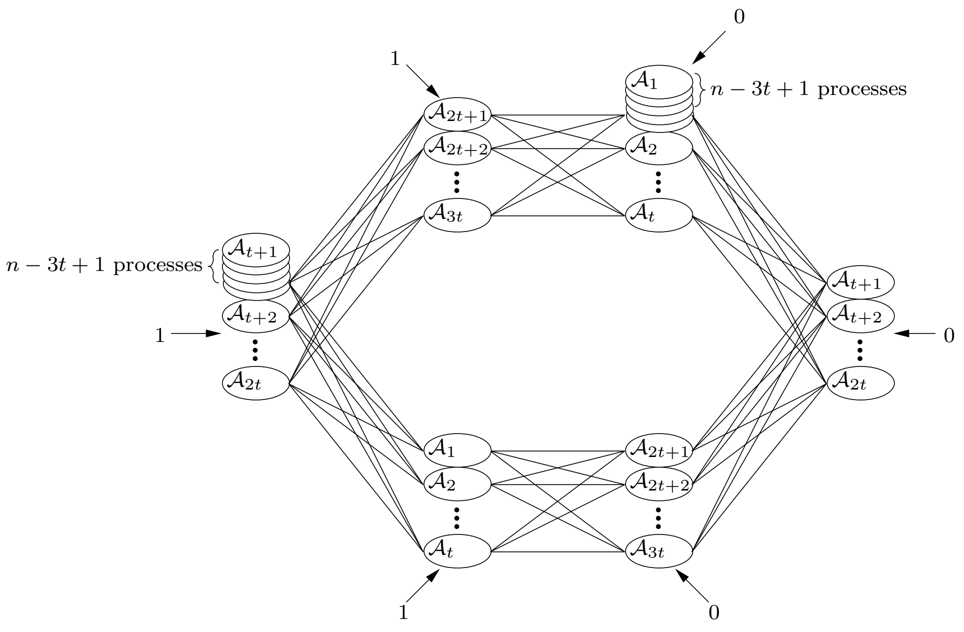

Figure 1: System used in proof of Proposition 1

As mentioned in the introduction, we also consider variants of the models in which each Byzantine process is *restricted* to sending at most one message to each recipient in each round. In general, we consider unrestricted Byzantine processes unless the restriction is explicitly mentioned. We also distinguish the cases where processes are *innumerate* from the case where they are *numerate*. We say that a process is innumerate if the messages it receives in a round form a set of messages: the process cannot count the number of copies of identical messages it receives in the round. We say that a process is numerate if the messages it receives in a round form a multiset of messages: the process can count the number of copies of identical messages it receives in the round. (As we shall show, the numerate model is more powerful than the innumerate model against restricted Byzantine processes.)

The goal of an agreement algorithm is for a set of processes proposing values to decide on exactly one of these values. We consider the classical *Byzantine agreement* problem [11,18], defined by the following three properties.

1. *Validity:* If all correct processes propose the same value v, then no value different from v can be decided by any correct process.

2. *Agreement:* No two correct processes decide different values.

3. *Termination:* Eventually, each correct process decides some value.

An algorithm solves Byzantine agreement in a system of n processes with ℓ identifiers tolerating t failures if these three properties are satisfied in every execution in which at most t processes fail, regardless of the way the n processes are assigned the ℓ identifiers. (Recall that each identifier must be assigned to at least one process.)

3. SYNCHRONOUS MODEL

Here, we prove that having $\ell > 3t$ is necessary and sufficient for solving synchronous Byzantine agreement, regardless of whether the processes are numerate or innumerate. To show that the condition $\ell > 3t$ is sufficient to reach agreement, we design a simulation, where each group of processes with a common identifier cooperatively simulate a single process.

3.1 Impossibility

We prove the condition $\ell > 3t$ is necessary using a scenario argument, in the style of Fischer, Lynch and Merritt [11].

Proposition 1 *Synchronous Byzantine agreement is unsolvable even with numerate processes if $\ell \leq 3t$.*

Proof. It suffices to prove there is no synchronous algorithm for Byzantine agreement when $\ell = 3t$. To derive a contradiction, suppose there was an n-process synchronous algorithm $\mathcal{A}$ for Byzantine agreement when $\ell = 3t$. Let $\mathcal{A}_i(v)$ be the algorithm executed by a process with identifier i when it has input value v.

Imagine setting up a system as shown in Figure 1. Every process correctly executes the algorithm $\mathcal{A}_i$ assigned to it. The two stacks of processes shown in the diagram each have $n - 3t + 1$ processes, so there are a total of $2n$ processes in this system. All processes within a stack have the same identifier, and execute the same algorithm $\mathcal{A}_i$, as shown. Inputs to each of the $2n$ process are indicated by the arrows.

Consider the $n-t$ processes that run $\mathcal{A}_{t+1}(1), \ldots, \mathcal{A}_{3t}(1)$. These $n-t$ processes cannot distinguish this execution from an execution in an n-process system where the remaining identifiers, $1, \ldots, t$ are each assigned to a single Byzantine process. (Here, we use the fact that each Byzantine process can send multiple messages to each correct process in a single round.) By validity, the $n - t$ processes must output 1.

By a symmetric argument, the $n-t$ processes running $\mathcal{A}_1(0),\ldots,\mathcal{A}_{2t}(0)$ must output 0.

Now, consider the $n-2t$ processes that run $\mathcal{A}_1(0),\ldots,\mathcal{A}_t(0)$ and the t processes that run $\mathcal{A}_{2t+1}(1),\ldots,\mathcal{A}_{3t}(1)$. These $n-t$ processes cannot distinguish this execution from an n-process execution where each of the remaining identifiers, $t+1,\ldots,2t$ are each assigned to a single Byzantine process. By agreement, the $n-t$ processes must output the same value, contradicting the previous two paragraphs □

3.2 Algorithm

Next, we present an algorithm that solves Byzantine agreement assuming $\ell > 3t$. Our agreement algorithm is generic: given any synchronous Byzantine agreement algorithm for ℓ processes with unique identifiers (such algorithms exist when $\ell = n > 3t$, e.g., [14]), we transform it into an algorithm for n processes and ℓ identifiers, where $n \geq \ell$. Without loss of generality, we assume that the algorithm to be transformed uses broadcasts: a process sends the same message to all other processes. (If a process wishes to send a message only to specific recipients, it could include the recipient's identifier in the broadcasted message.)

In our transformation, we divide processes into groups according to their identifiers. Each group simulates a single process. If all processes within a group are correct, then they can reach agreement and cooperatively simulate a single process. If any process in the group is Byzantine, we allow the simulated process of that group to behave in a Byzantine manner. The correctness of our simulation relies on the fact that more than two-thirds of the simulated processes will be correct (since $\ell > 3t$), which is enough to achieve agreement.

Proposition 2 *Synchronous Byzantine agreement is solvable even with innumerate processes if $\ell > 3t$.*

Proof sketch. We transform any Byzantine agreement algorithm $\mathcal{A}$ for the classical model with unique identifiers into an algorithm $\mathcal{T}(\mathcal{A})$ for systems with homonyms. Consider any such $\mathcal{A}$ (Figure 2) for a system with ℓ processes $\{p_1,\ldots,p_\ell\}$. $\mathcal{A}$ can be specified by: (1) a set of local process states, (2) a function $init(i,v)$ that encodes the initial state of process p_i when p_i has input value v, (3) a function $M(s,r)$ that determines the message to send in state s in round r, (4) a transition function $\delta(s,r,R)$ that determines the new state to which the process moves from state s after receiving a set of messages R in round r, and (5) a decision function $decide(s)$ which is the decision in state s, or $\perp$ if there is no decision yet (once a correct process has decided in a state s, $decide(s')$ remains equal to this decision in all states s' reachable from s).

Let $G(i)$ be the set of processes with identifier i. We name such a set a *group*. We say that the group $G(i)$ is correct if all processes in $G(i)$ are correct. At most t of the ℓ groups are not correct.

In our new algorithm $\mathcal{T}(\mathcal{A})$, shown in Figure 3, three rounds simulate one round of $\mathcal{A}$. We call these three rounds a *phase*. Each phase consists of a *selection round*, a *deciding round* and a *running round*. In the selection round (line 3 to 5) of a phase r, the processes within each group agree on a state for phase r. For each i, if $G(i)$ is correct, then in each round the selected state will be the same for the processes

```
Code for process $p_i$
Variable:
    $s = init(i,v)$            /* $v$ is the value proposed by $p_i$ */
Main code:
    for all $r$ from 1 to $\infty$
        if $decide(s) \neq \perp$ then decide the value $decide(s)$
        send($M(s,r)$) to all processes
        receive($R$)   /* receive messages sent this round */
        $s = \delta(s,r,R)$
```

Figure 2: Synchronous Byzantine agreement algorithm $\mathcal{A}$ with ℓ processes and ℓ identifiers.

in this group. In deciding rounds (line 6 to 9), if there is a value decided by $t+1$ processes with different identifiers then the process can decide that value. At least one of these identifiers refers to a correct group and gives the decision. The deciding rounds are useful for correct processes that belong to a group with a Byzantine process. In running rounds (line 10 to 15), each process executes one step of algorithm $\mathcal{A}$ with the state chosen in the preceding selection round and the messages received in the round.

Let α_H be an execution of $\mathcal{T}(\mathcal{A})$. For all phases r, at the end of the r-th selection round (line 5), all processes in a correct group $G(i)$ have the same value for the state s, and therefore for $M(s,r)$ and $decide(s)$. Let s_i^r be the value of state s for the processes in group $G(i)$ after the rth selection round. Note that s_i^1 is the initial state of at least one process in $G(i)$.

By induction on r, we prove that there is an execution α_S of $\mathcal{A}$ such that for all r and for all processes in each correct group $G(i)$: $s_i^r = st_i^r$ (and hence $M(s_i^r,r) = M(st_i^r,r)$), where st_i^r is the value of p_i's variable s at the beginning of round r in α_S. In α_S, p_i is correct for all identifiers i such that $G(i)$ is correct in α_H.

We sketch the key idea of the inductive step that proves this claim. In each running round, messages sent by the processes in a correct group $G(i)$ are identical and indistinguishable from a single message from a unique correct process with identifier i. On the other hand, if $G(i)$ is not correct, the processes in $G(i)$ may send different messages to a process p (in which case p ignores the messages at line 14) or they may all send the same (arbitrary) message to p. Either way, their collective behaviour is indistinguishable from a unique Byzantine process with identifier i (which could either send nothing or an arbitrary message to p).

As $\mathcal{A}$ is a synchronous Byzantine agreement algorithm that tolerates t Byzantine failures, all correct processes eventually decide some value v in α_S. It follows from the claim above that in α_H, eventually for all correct groups $G(i)$, s_i^r is a state where $decide(s_i^r)$ is v. As $\ell > 3t$, at least $t+1$ groups $G(i)$ are correct and all processes in these groups eventually send v in the deciding rounds. Thus, each correct process in α_H eventually decides, even if it is in a group with a Byzantine process. Furthermore, if a correct process decides in α_H, it decides the value it received from $t+1$ groups, at least one of which is a correct group, so it must decide v. Thus, the agreement, validity and termination

```
Code for processes with identifier $i$

Variable:
    $s = init(i, v)$                                              /* $v$ is the value proposed by the process */

Main code:
    for all $r$ from 1 to $\infty$
        send($s$) to all processes                                /* get groups to agree on their state */
        receive($R$)                                              /* receive the messages of the round */
        $s$ = deterministic choice of some element $x.val$ such that $x \in R$ and $x.id = i$

        send($decide(s)$) to all processes        /* deciding round replaces decision line of original algorithm */
        receive($R$)                                              /* receive the messages of the round */
        if there is a $v \neq \bot$ such that $|\{d \in R : d.val = v\}| \geq t + 1$
            then decide such a $v$

        send($M(s, r)$) to all processes                          /* almost identical to original algorithm */
        receive($R$)                                              /* receive the messages of the round */
        for all $j$ in $\mathcal{L}$                              /* eliminate messages from known Byzantine groups */
            if there is more than one different message from identifier $j$ in $R$
                then remove all of them from $R$
        $s = \delta(s, r, R)$
```

Figure 3: Synchronous Byzantine agreement algorithm $\mathcal{T}(\mathcal{A})$ with n processes and ℓ identifiers.

properties for α_H follow from the agreement, validity and termination properties for α_S. □

If the algorithm in Figure 2 is known to terminate within k rounds, the algorithm in Figure 3 need only be run for $k + 1$ iterations of the loop. (The extra iteration provides an additional deciding round to ensure correct processes in incorrect groups decide.)

Proposition 1 states that $\ell > 3t$ identifiers are required to solve synchronous Byzantine agreement, even if processes are numerate. Proposition 2 states that $\ell > 3t$ identifiers are sufficient, even if processes are innumerate. Thus, we have the following theorem.

Theorem 3 *Synchronous Byzantine agreement is solvable if and only if $\ell > 3t$.*

4. PARTIALLY SYNCHRONOUS MODEL

Here we prove that having $\ell > \frac{3t+n}{2}$ is necessary and sufficient for solving Byzantine agreement in a partially synchronous system, regardless of whether the processes are numerate or innumerate. Intuitively, this condition means that at least $3t+1$ of the identifiers must each be assigned to a single process (since $2\ell - n > 3t$). We shall see in Section 4.2 that having this many non-homonym processes will be crucial in proving the correctness of the algorithm that we design.

4.1 Impossibility

We prove the necessity of the condition $\ell > \frac{n+3t}{2}$ using a partitioning argument. We show that if there are too few identifiers, and messages between two groups of correct processes are not delivered for sufficiently long, then the Byzantine processes can force processes in the two groups to decide different values.

Proposition 4 *Partially synchronous Byzantine agreement is unsolvable even with numerate processes if $\ell \leq \frac{n+3t}{2}$.*

Proof. Byzantine agreement is impossible when $\ell \leq 3t$, even in the fully synchronous model, by Proposition 1. So, it remains to show that agreement is impossible when $\ell > 3t$ and $\ell \leq \frac{n+3t}{2}$. To derive a contradiction, assume a Byzantine agreement algorithm $\mathcal{A}$ does exist for such a system. In our proof, we construct three executions of this algorithm, α, β and γ.

In α, process identifiers are assigned as shown in the upper left portion of Figure 4. In this diagram, a process labelled $\mathcal{A}_i$ has identifier i and runs the algorithm $\mathcal{A}$ correctly, and a process labelled $\mathcal{B}_i$ has identifier i and is Byzantine. Note that there are n processes in total. The t Byzantine processes send no messages and all messages sent by correct processes are delivered. All correct processes have input 0 in α and must therefore decide 0 by some round r_α.

Execution β is defined similarly, as shown in the upper right portion of Figure 4. Again, the t Byzantine processes send no messages and all messages sent by correct processes are delivered. All correct processes have input 1, and must therefore decide 1 by some round r_β.

In γ, the n processes are assigned identifiers as shown in the bottom half of Figure 4. (Here, we use the assumption that $\ell \leq \frac{n+3t}{2}$, so that $n \geq 2\ell - 3t$.) The inputs to each group of correct processes is also shown in the diagram. The t Byzantine processes $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_t$ send to each correct process with input 0 the same messages as that process receives in α and they send to each correct process with input 1 the same messages as that process receives in β. (This requires the ability of Byzantine process $\mathcal{B}_1$ to send more than one message to each recipient per round.) All messages sent across the edges shown in the diagram are delivered. All other messages are not delivered for the first $r = \max(r_\alpha, r_\beta)$ rounds. The correct processes with input 0 cannot distinguish γ from α for the first r rounds, so they must decide 0 by round r. The correct processes with input 1 cannot distinguish γ from β for the first r rounds, so they must decide 1 by round r. This contradicts the assumption that $\mathcal{A}$ satisfies agreement. □

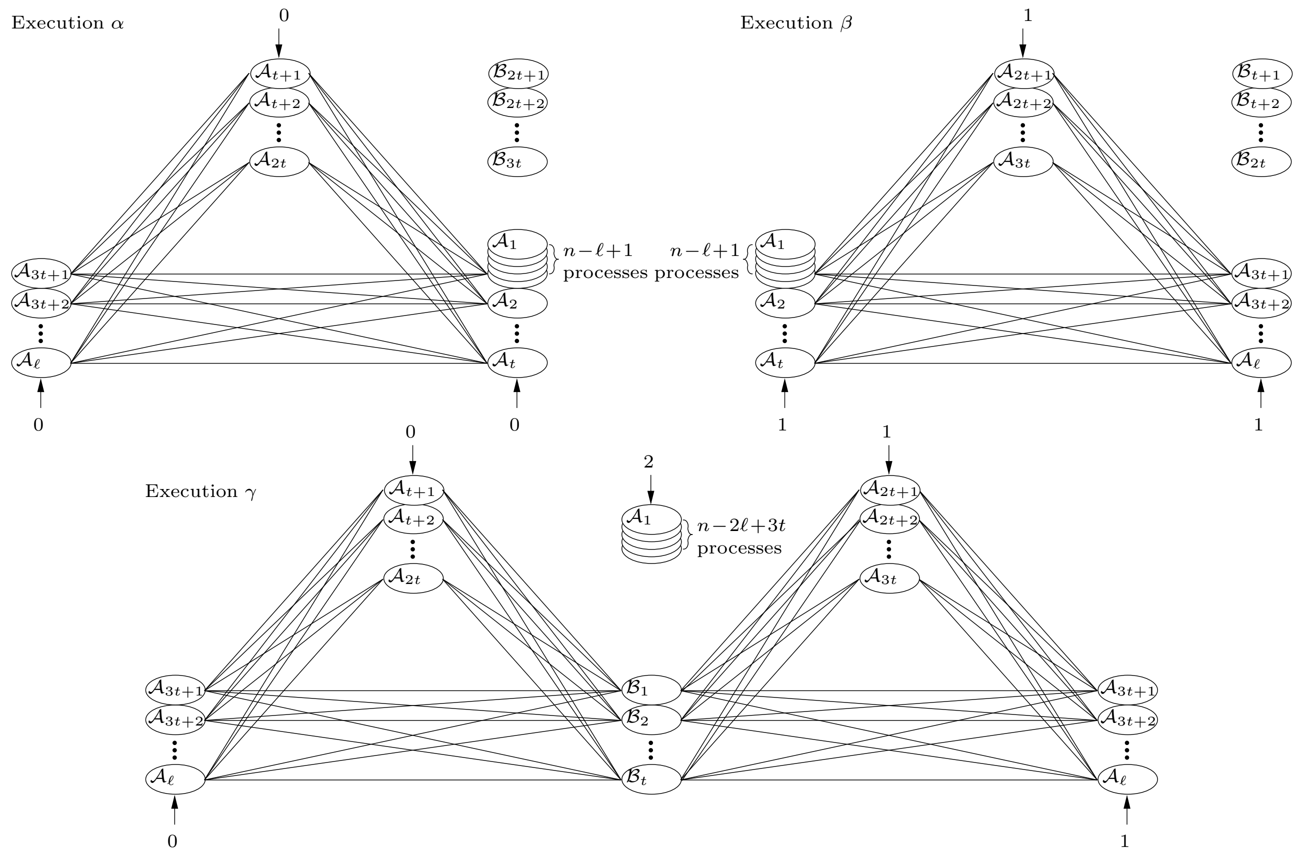

Figure 4: System used in proof of Proposition 4

4.2 Algorithm

We now describe an algorithm that solves Byzantine agreement in the basic partially synchronous model when $\ell > \frac{n+3t}{2}$. Our algorithm is based on the algorithm given by Dwork, Lynch and Stockmeyer [10] for the classical case where $n = \ell$, with several novel features. Generalizing the algorithm is not straightforward. Some of the difficulty stems from the following scenario. Suppose two correct processes share an identifier and follow the traditional algorithm of [10]. They could send very different messages (for example, if they have different input values), but recipients of those messages would have no way of telling apart the messages of the two correct senders, so it could appear to the recipients as if a single Byzantine process was sending out contradictory information. Thus, the algorithm has to guard against inconsistent information coming from correct homonym processes as well as malicious messages sent by the Byzantine processes.

Proposition 5 *Partially synchronous Byzantine agreement is solvable even with innumerate processes if $\ell > \frac{n+3t}{2}$.*

We think of an execution as being divided into superrounds, where each superround consists of two consecutive rounds. Let T be the first superround such that all messages sent during or after superround T are delivered. We begin with an *authenticated broadcast* primitive based on [21]. This primitive allows processes to perform BROADCAST(m) commands. Once a process receives sufficient evidence that a process with identifier i has performed a BROADCAST(m), it performs an ACCEPT(m, i) action. This is guaranteed to happen for broadcasts from correct processes after superround T. (In the case where a process with identifier i is Byzantine, processes will at least eventually agree on which messages to accept from identifier i.) Our version of authenticated broadcast for homonymous systems satisfies the following three properties.

1. *Correctness*: If a process with identifier i performs BROADCAST(m) in superround $r \geq T$, then every correct process performs ACCEPT(m, i) during superround r.

2. *Unforgeability*: If all processes with identifier i are correct and none of them perform BROADCAST(m), then no correct process performs ACCEPT(m, i).

3. *Relay*: If some correct process performs ACCEPT(m, i) during superround r, then every correct process performs ACCEPT(m, i) by superround $\max(r + 1, T)$.

Proposition 6 *It is possible to implement authenticated broadcasts satisfying the correctness, unforgeability and relay properties in the basic partially synchronous model, provided $\ell > 3t$.*

PROOF SKETCH. The implementation is a straightforward generalization of the ones given in [10, 21] for systems with

```
Code for process with identifier $i \in \{1, \ldots, \ell\}$

1  $locks = \emptyset$
2  $ph = 0$                                                        /* phase number */
3  $proper = \{v\}$                    /* $v$ is the value proposed by the process */
4  Note: in each round, $proper$ is updated as described on page
5  **while** true
6      /* beginning of superround 1 of phase */
7      let $V$ be the set of values $v \in proper$ such that there is no pair $(w, *) \in locks$ for any $w \neq v$
8      Broadcast($\langle$propose $V, ph\rangle$)                     /* superround 1 */
9      /* beginning of superround 2 of phase */
10     **if** $i = (ph \bmod \ell) + 1$ and there is some value $v_{lock}$ such that the process has performed Accept($\langle$propose $V_j, ph\rangle, j$)
11             from $\ell - t$ different identifiers $j$ with $v_{lock} \in V_j$
12         **then** send $\langle$lock $v_{lock}, ph\rangle$ to all processes     /* round 1 of superround 2 */
13     /* beginning of superround 3 of phase */
14     **if** there is some value $v$ for which the process received $\langle$lock $v, ph\rangle$ from identifier $(ph \bmod \ell) + 1$ and
15             has performed Accept($\langle$propose $V_j, ph\rangle, j$) for $\ell - t$ different identifiers $j$ with $v \in V_j$
16         **then** choose one such $v$ and perform Broadcast($\langle$vote $v, ph\rangle$)     /* superround 3 */
17     /* beginning of superround 4 of phase */
18     **if** for some $v$, the process has performed Accept($\langle$vote $v, ph\rangle, j$) from $\ell - t$ different identifiers $j$
19         **then**  add $(v, ph)$ to locks and remove any other pair $(v, *)$ from $locks$
20                   send $\langle$ack $v, ph\rangle$ to all processes     /* round 1 of superround 4 */
21     **if** $i = (ph \bmod \ell) + 1$ and the process has received $\langle$ack $v_{lock}, ph\rangle$ from $\ell - t$ different identifiers in this round
22         **then** decide $v_{lock}$ (but continue running the algorithm)
23     **if** the process has already decided some value $v$
24         **then** send $\langle$decide $v\rangle$ to all processes     /* round 2 of superround 4 */
25     **if** for some $v$, the process has received $\langle$decide $v\rangle$ from $t + 1$ different identifiers $j$ in this round
26         **then** decide $v$ (but continue running the algorithm)
27     **for each** $(v_1, ph_1) \in locks$
28         **if** for some $v_2 \neq v_1$ and $ph_2 > ph_1$, the process has performed Accept($\langle$vote $v_2, ph_2\rangle, j$) for $\ell - t$
29                 different identifiers $j$
30             **then** remove $(v_1, ph_1)$ from $locks$
31     $ph = ph + 1$
```

Figure 5: Byzantine agreement algorithm for the partially synchronous model.

unique identifiers. To perform Broadcast(m) in superround r, a process sends a message $\langle$init $m\rangle$ in the first round of superround r. Any process that receives this message from identifier i sends $\langle$echo $m, r, i\rangle$ in the following round, which is the second round of superround r, and in all subsequent rounds. In each round after superround r, any process that has so far received $\langle$echo $m, r, i\rangle$ from $\ell - 2t$ distinct identifiers sends a message $\langle$echo $m, r, i\rangle$. If, at any time, a process has received the message $\langle$echo $m, r, i\rangle$ from $\ell - t$ distinct identifiers, the process performs Accept(m, i). ☐

We now describe the Byzantine agreement protocol. Each process keeps track of a set of *proper* values, which are values that can be output without violating validity. Initially, only the process's own input value is in this set. Each process appends its *proper* set to each message it sends. If a process receives *proper* sets containing v in messages from $t + 1$ different identifiers, it adds v to its own *proper* set. Also, if a process has received *proper* sets from $2t + 1$ different identifiers and no value appears in $t + 1$ of them, it adds all possible input values to its own *proper* set. (This can be done because $t + 1$ of the *proper* sets are from correct processes, so there are at least two different inputs to correct processes.)

The Byzantine agreement algorithm is shown in Figure 5. Whenever a correct process sends a message, it sends it to all processes. The execution of the algorithm is broken into phases, each of which lasts four superrounds. Processes assigned the identifier $(ph \bmod \ell)+1$ are the *leaders* of phase *ph*. In each phase, each process first performs a Broadcast of a proposal containing the set of values it would be willing to decide (line 8). These are the values in its *proper* set, unless it has already locked a value, as described below, in which case it can only send its locked value. Each phase leader chooses a value that appears in proposals the leader has accepted from $\ell - t$ different identifiers (if such a value exists) and sends out a request for processes to lock that value (line 12) during superround 2 of the phase. Then, in superround 3 of the phase, all processes vote on which lock message to support, using a Broadcast (line 16). In the third superround of the phase, each process that performed Accept for votes for a particular value v from $\ell - t$ different identifiers sends $\langle$ack $v\rangle$ back to the leaders (line 20) and locks the value v (by adding the value to its *locks* set, along with the phase number associated with the lock). A leader that receives $\ell - t$ ack messages for the value it wanted locked can decide that value (line 22). Finally, each process that has decided sends a message to others (line 23); if any process receives such a message with the same decision value from $t + 1$ identifiers, it can also decide that value (line 26). At the end of a phase, a process releases old locks (line 30) if it has accepted enough votes for a later lock request.

To cope with homonyms, our algorithm differs from the original algorithm of [10] in the following three ways. (1) The new algorithm uses a set of processes with $\ell - t$ different identifiers as a quorum (e.g., for vote messages). The key property of these quorums is that any two such sets must both contain a process that is correct and does not share its identifier with any other process, as shown in Lemma 7, below. (2) The vote messages are needed to ensure agreement in the case where several leaders ask processes to lock different values, something which could not occur in the original algorithm of [10], since each phase in that algorithm has a unique leader. (3) The decide messages are used to ensure that a correct process that shares its identifier with a Byzantine process can eventually decide. (This is similar to the mechanism used in Section 3.2.) We begin by proving the property of quorums used by the algorithm.

Lemma 7 *Assume $\ell > \frac{n+3t}{2}$. If A and B are sets of identifiers and $|A| \geq \ell - t$ and $|B| \geq \ell - t$, then $A \cap B$ contains an identifier that belongs to only one correct process and no Byzantine processes.*

Proof. At most $n-\ell$ identifiers belong to more than one process. At most t identifiers belong to Byzantine processes. Thus, any set that has more than $n - \ell + t$ identifiers must contain an identifier that belongs to only one correct process and no Byzantine processes. Since $2\ell - 3t > n$, we have $|A \cap B| = |A| + |B| - |A \cup B| \geq |A| + |B| - \ell \geq (\ell - t) + (\ell - t) - \ell = 2\ell - 3t - \ell + t > n - \ell + t$. □

In the original algorithm of [10], each phase has a unique leader. In our algorithm, there may be several leaders. The new voting superround ensures this cannot cause problems, as shown in the following lemmas.

Lemma 8 *If the messages $\langle \text{ack } v, ph\rangle$ and $\langle \text{ack } v', ph\rangle$ are sent by correct processes, then $v = v'$.*

Proof. Suppose a correct process p sends $\langle \text{ack } v, ph\rangle$ and a correct process p' sends $\langle \text{ack } v', ph\rangle$. (We may have $p = p'$.) According to line 18, there is a set A of $\ell - t$ identifiers j for which p performs Accept($\langle \text{vote } v, ph\rangle, j$). Similarly, there is a set B of $\ell - t$ identifiers j for which p' performs Accept($\langle \text{vote } v', ph\rangle, j$). By Lemma 7, $A \cap B$ contains an identifier j that belongs to only one correct process and no Byzantine processes. By unforgeability, the correct process with identifier j performed Broadcast($\langle \text{vote } v, ph\rangle$) and Broadcast($\langle \text{vote } v', ph\rangle$). Thus, $v = v'$. □

Lemma 9 *If two correct processes decide on line 22 in the same phase, then they decide the same value.*

Proof. Suppose two correct processes p and p' decide values v and v', respectively, during some phase ph. Then, process p received $\langle \text{ack } v, ph\rangle$ from $\ell - t > t$ different identifiers, so some correct process must have sent $\langle \text{ack } v, ph\rangle$. Similarly, some correct process must have sent $\langle \text{ack } v', ph\rangle$. By Lemma 8, $v = v'$. □

The remainder of the proof of correctness of the algorithm is similar to the proof for the original algorithm of [10]. It is given in [8], using the following lemmas.

Lemma 10 *Suppose there is a value v and a phase ph such that processes with $\ell - t$ different identifiers sent an $\langle \text{ack } v, ph\rangle$ message in phase ph. Then, at all times after phase ph, each correct process that sent $\langle \text{ack } v, ph\rangle$ has a pair (v, ph') with $ph' \geq ph$ in its locks set.*

Lemma 11 *At the end of any phase ph_3 that occurs after T, if (v_1, ph_1) is in the locks variable of a correct process p_1 and (v_2, ph_2) is in the locks variable of a correct process p_2, then $v_1 = v_2$.*

Lemma 12 *Let p be a correct process. Let ph be a phase such that $(ph \bmod \ell) + 1$ is the identifier of p and phase $ph - 1$ occurs after T. Then, p will send a lock message in superround 2 of phase ph.*

Combining Proposition 4 and 5 yields the following theorem (for numerate or innumerate processes).

Theorem 13 *Partially synchronous Byzantine agreement is solvable if and only if $\ell > \frac{n+3t}{2}$.*

5. RESTRICTED BYZANTINE PROCESSES

We now consider the effect of restricting the Byzantine processes so that each Byzantine process can send at most one message to each recipient in each round. We prove that this restriction reduces the number of identifiers needed to reach agreement if processes are numerate but does not help if processes are innumerate.

5.1 Numerate Processes

First, we consider the model where processes can count copies of identical messages. We prove the following two theorems for this model.

Theorem 14 *Synchronous Byzantine agreement is solvable with numerate processes against restricted Byzantine processes if and only if $\ell > t$.*

Theorem 15 *Partially synchronous Byzantine agreement is solvable with numerate processes against restricted Byzantine processes if and only if $\ell > t$.*

Both of these theorems follow from Proposition 16 and 18, below.

Proposition 16 *Synchronous Byzantine agreement is unsolvable with numerate processes against restricted Byzantine processes if $\ell \leq t$.*

Proof sketch. To derive a contradiction, assume that there exists an algorithm $\mathcal{A}$ that solves Byzantine agreement with $\ell \leq t$. In the argument below, we consider only executions of $\mathcal{A}$ with some fixed set of ℓ Byzantine processes, chosen so that each of the ℓ identifiers is held by one Byzantine process.

We consider configurations of the the algorithm $\mathcal{A}$ at the end of a synchronous round. Such a configuration can be completely specified by the state of each process. A configuration C is *0-valent* if, starting from C, the only possible decision value that correct processes can have is 0; it is *1-valent* if, starting from C, the only possible decision value that correct processes can have is 1. C is *univalent* if it is either 0-valent or 1-valent; C is *multivalent* if it is not univalent.

The following lemma encapsulates a Byzantine agent's ability to influence the decision value.

Lemma 17 *Let C and C' be two configurations of $\mathcal{A}$ such that the state of only one correct process is different in C and C'. Then, there exist executions α and α' that start from C and C', respectively, which both produce the same output value.*

PROOF. Let p be the correct process whose state is different in C and C' and let i be the identifier assigned to p. Let s and s' be the state of p in C and C', respectively. Let b be a Byzantine process that has identifier i.

Let α be the execution from C in which b starts in state s' and follows p's algorithm, and all other Byzantine processes send no messages. Let α' be the execution from C' in which b starts in state s and follows p's algorithm, and all other Byzantine processes send no messages. No correct process other than p can distinguish between α and α', since p and b send the same messages in α as b and p send in α'. Thus, each correct process other than p must output the same decision in α and α'. □

The remainder of the proof of Proposition 16 is a standard valency argument (see [8]). We sketch it here. By validity, the initial configuration where all inputs are 0 is 0-valent. We can obtain a sequence of initial configurations by changing the correct process's inputs to 1, one at a time. By validity, the final configuration in this sequence is 1-valent. By Lemma 17 successive configurations in this sequence are capable of leading to the same output. It follows that some initial configuration in this sequence is multivalent.

A similar argument can be used to show that every multivalent configuration must have a multivalent successor configuration, again using Lemma 17. Hence, we can construct an infinite execution of multivalent configurations in which no process ever decides, violating termination. This contradiction establishes Proposition 16. □

Proposition 18 *Partially synchronous Byzantine agreement is solvable with numerate processes against restricted Byzantine processes if $\ell > t$.*

The algorithm used to prove this proposition is similar to the one presented in Section 4.2. Details may be found in [8]. Like the algorithm in Section 4.2, it uses an authenticated broadcast primitive, but here ACCEPT actions have an extra parameter indicating the multiplicity of the accepted message. More precisely, this multiplicity is greater than the number of correct processes that sent the message and does not exceed the number of correct processes by more than the actual number of Byzantine processes in the execution. Furthermore, all correct processes agree eventually on the multiplicity of each message.

This authenticated broadcast with multiplicity is used to ensure the agreement property. As $\ell > t$, at least one identifier is assigned only to correct processes. This property is used to ensure the termination property of the agreement algorithm.

5.2 Innumerate Processes

Theorem 19 *Synchronous Byzantine agreement is solvable with innumerate processes against restricted Byzantine processes if and only if $\ell > 3t$.*

PROOF SKETCH. (See [8] for a detailed proof.) By Proposition 2, there is an algorithm if $\ell > 3t$, even against (unrestricted) Byzantine processes, so the same algorithm would work against restricted Byzantine processes. To prove that $\ell > 3t$ is necessary, we use a simulation. If it were possible to solve the problem when $\ell \leq 3t$, this algorithm would work, in particular, when $n - \ell + 1$ of the processes are all assigned the same identifier and input, and all receive exactly the same messages from the Byzantine agents. In this situation, the $n - \ell + 1$ processes would behave as clones, taking exactly the same sequence of steps. This would imply that the same algorithm would solve Byzantine agreement when $n = \ell \leq 3t$, which is known to be impossible. □

Theorem 20 *Partially synchronous Byzantine agreement is solvable with innumerate processes against restricted Byzantine processes if and only if $\ell > \frac{n+3t}{2}$.*

PROOF. By Proposition 5, there is an algorithm if $\ell > \frac{n+3t}{2}$, even against (unrestricted) Byzantine processes, so the same algorithm would work against restricted Byzantine processes. The impossibility result can be proved in exactly the same way as Proposition 4. In that proof, only the Byzantine process denoted $\mathcal{B}_1$ must send multiple messages to a single recipient in execution γ. Consider the messages $\mathcal{B}_1$ must send to $\mathcal{A}_t(0)$ in γ. It must send the same messages as the entire stack of processes running $\mathcal{A}_1$ send to $\mathcal{A}_t(0)$ in α. However, all processes in that stack behave identically in α, so $\mathcal{B}_1$ must simply send $n - \ell + 1$ copies of a message to $\mathcal{A}_t(0)$. Since we are now considering a model where $\mathcal{A}_t(0)$ is innumerate, $\mathcal{B}$ can simply send one copy of the message to $\mathcal{A}_t(0)$ instead. (A symmetric argument applies to the messages sent by $\mathcal{B}_1$ to each other process in γ.) □

6. CONCLUDING REMARKS

Since the pioneering work of [1], the question of what can be computed in a totally anonymous distributed systems has been extensively studied. Some results depended on properties of the communication graph (e.g., [4, 23]). Some work considered shared memory for the "wake up" problem [13], others considered consensus [3]. The power of anonymous broadcast systems, in comparison with anonymous shared-memory systems has also been studied [2]. None of these considered process failures. Anonymous processes with crash failures have been considered more recently [5, 6, 9, 12, 17, 20]. In [16], Byzantine agreement was studied in a model with a restricted kind of anonymity: processes have no identifiers, but each process has a separate channel to every other process and a process can detect through which channel an incoming message is delivered. It was shown that Byzantine agreement can be solved in this model when $n > 3t$.

This paper is the first to study a distributed system model with homonyms, *i.e.*, with a limited number of identifiers. The model unifies both classical (non-anonymous) and anonymous models and is interesting from both a theoretical and a practical viewpoint. We completely characterized the solvability of Byzantine agreement in this model, precisely quantifying the impact of the adversary, with some surprising results. We focused however on agreement and many other problems would be interesting to consider. We also focused on computability and complexity is yet to be explored.

Acknowledgments.
We are grateful to Christian Cachin for his useful comments on our model with homonyms. A small subset of our results was presented in a preliminary form in [7]. Eric Ruppert received funding from the Natural Sciences and Engineering Research Council of Canada.

7. REFERENCES

[1] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proc. 12th ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980.

[2] James Aspnes, Faith Ellen Fich, and Eric Ruppert. Relationships between broadcast and shared memory in reliable anonymous distributed systems. *Distributed Computing*, 18(3):209–219, February 2006.

[3] Hagit Attiya, Alla Gorbach, and Shlomo Moran. Computing in totally anonymous asynchronous shared memory systems. *Information and Computation*, 173(2):162–183, 2002.

[4] Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In *Proc. 15th International Conference on Distributed Computing*, volume 2180 of *LNCS*, pages 33–47. Springer, 2001.

[5] François Bonnet and Michel Raynal. The price of anonymity: Optimal consensus despite asynchrony, crash and anonymity. In *Proc. 23rd International Symposium on Distributed Computing*, volume 5805 of *LNCS*, pages 341–355. Springer, 2009.

[6] Harry Buhrman, Alessandro Panconesi, Riccardo Silvestri, and Paul M. B. Vitányi. On the importance of having an identity or, is consensus really universal? *Distributed Computing*, 18(3):167–176, February 2006.

[7] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Anne-Marie Kermarrec. Brief announcement: Byzantine agreement with homonyms. In *Proc. 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 74–75, 2010.

[8] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec, Eric Ruppert, and Hung Tran-The. Byzantine agreement with homonyms. Technical Report hal-00580133, CNRS, France, 2011.

[9] Carole Delporte-Gallet, Hugues Fauconnier, and Andreas Tielmann. Fault-tolerant consensus in unknown and anonymous networks. In *Proc. 29th IEEE International Conference on Distributed Computing Systems*, pages 368–375. IEEE Computer Society, 2009.

[10] Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[11] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, January 1986.

[12] Rachid Guerraoui and Eric Ruppert. Anonymous and fault-tolerant shared-memory computing. *Distributed Computing*, 20(3):165–177, October 2007.

[13] Prasad Jayanti and Sam Toueg. Wakeup under read/write atomicity. In Jan van Leeuwen and Nicola Santoro, editors, *Proc. 4th International Workshop on Distributed Algorithms*, volume 486 of *LNCS*, pages 277–288. Springer, 1990.

[14] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[15] Michael Okun. Agreement among unacquainted Byzantine generals. In *Proc. 19th International Conference on Distributed Computing*, volume 3724 of *LNCS*, pages 499–500. Springer, 2005.

[16] Michael Okun and Amnon Barak. Efficient algorithms for anonymous Byzantine agreement. *Theory of Computing Systems*, 42(2):222–238, February 2008.

[17] Michael Okun, Amnon Barak, and Eli Gafni. Renaming in synchronous message passing systems with Byzantine failures. *Distributed Computing*, 20(6):403–413, April 2008.

[18] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[19] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, volume 2218 of *LNCS*, pages 329–350, 2001.

[20] Eric Ruppert. The anonymous consensus hierarchy and naming problems. In *Proc. Principles of Distributed Systems, 11th International Conference*, volume 4878 of *LNCS*, pages 386–400. Springer, 2007.

[21] T. K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[22] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, pages 149–160, 2001.

[23] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part I-characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.