HAL
archives-ouvertes.fr

Byzantine Agreement with Homonyms

Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec, Eric Ruppert, Hung Tran-The

▶ **To cite this version:**

Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec, Eric Ruppert, et al.. Byzantine Agreement with Homonyms. PODC - 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Jun 2011, San Jose, United States. pp.21-30, 2011, <10.1145/1993806.1993810>. <hal-00580133>

HAL Id: hal-00580133
https://hal.archives-ouvertes.fr/hal-00580133

Submitted on 4 Apr 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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.

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 gives us 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 [18,21] 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 [14], even with a single faulty process.) With a limited number of identifiers, more problems become solvable, and one can still preserve some level of anonymity 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. Thus, 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 consider systems where $n > 3t$: this assumption is known to be a requirement for solving Byzantine agreement, even when $\ell = n$ [13, 17], 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 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 agree-

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

	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 $3t$.

ment 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.

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 $\ell = n$ [13, 17], 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 [9]: 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 [9], this basic partially synchronous model is equivalent to other kinds of partially synchronous models. 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 types of partially synchronous systems too.

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

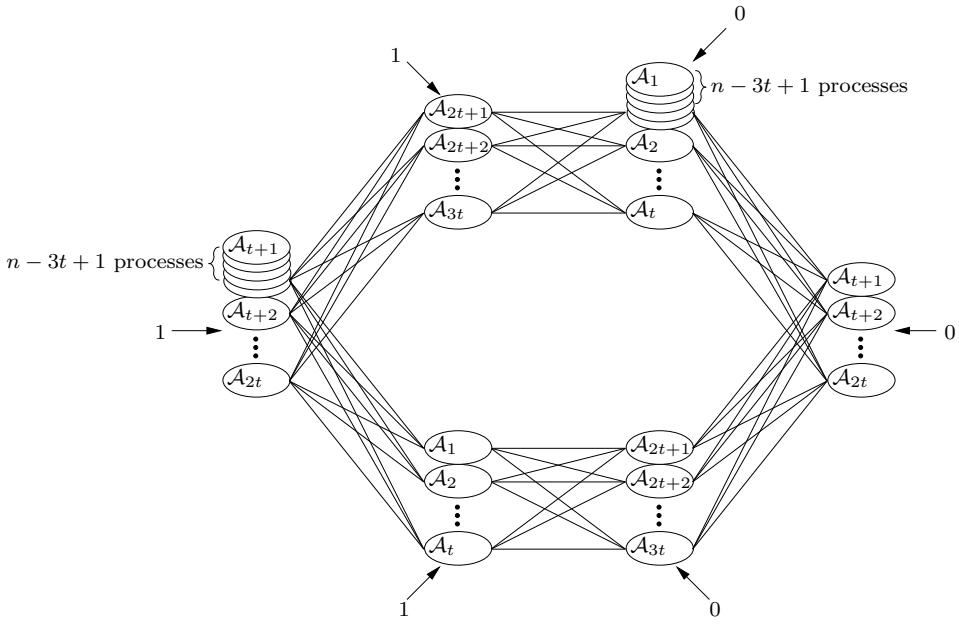

Figure 1: System used in proof of Proposition 1

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 [10, 17], 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 differently. (3) *Termination:* Eventually, every 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. THE SYNCHRONOUS CASE

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 [10].

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., [13]), 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 in this group. In deciding rounds (line 6 to 9), if there is a value decided by $t+1$ processes with different identifiers

Code for process p_i

Variable:

1 $s = init(i, v)$ /* v is the value proposed by p_i */

Main code:

2 **for all** r from 1 to ∞
3 **if** $decide(s) \neq \perp$ **then** decide the value $decide(s)$
4 **send**$(M(s, r))$ to all processes
5 **receive**(R) /* receive messages sent this round */
6 $s = \delta(s, r, R)$

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

Code for processes with identifier i

Variable:

1 $s = init(i, v)$ /* v is the value proposed by the process */

Main code:

2 **for all** r from 1 to ∞
3 **send**(s) to all processes /* get groups to agree on their state */
4 **receive**(R) /* receive the messages of the round */
5 s = deterministic choice of some element $x.val$ such that $x \in R$ and $x.id = i$
6 **send**$(decide(s))$ to all processes /* deciding round replaces decision line of original algorithm */
7 **receive**(R) /* receive the messages of the round */
8 **if** there is a $v \neq \perp$ such that $|\{d \in R : d.val = v\}| \geq t + 1$
9 **then** decide such a v
10 **send**$(M(s, r))$ to all processes /* almost identical to original algorithm */
11 **receive**(R) /* receive the messages of the round */
12 **for all** j in $\mathcal{L}$ /* eliminate messages from known Byzantine groups */
13 **if** there is more than one different message from identifier j in R
14 **then** remove all of them from R
15 $s = \delta(s, r, R)$

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

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

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. THE PARTIALLY SYNCHRONOUS CASE

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 $\mathcal{A}$ exists for such a system. 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. □

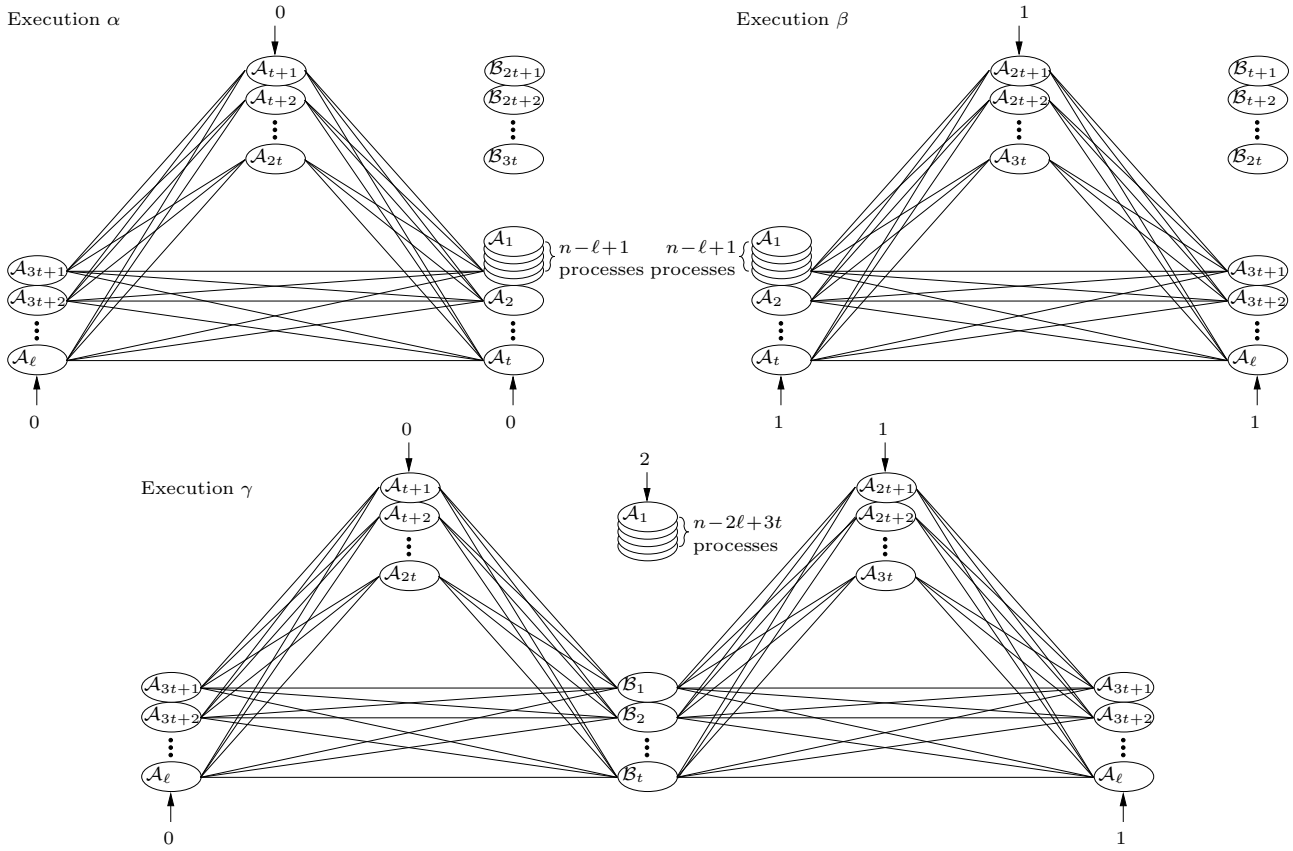

Figure 4: System used in proof of Proposition 4

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 [9] 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 [9]. 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 [20]. 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 performs 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 [9, 20] for systems with 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 mod ℓ) + 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 [9] 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 [9], 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 [9], 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 $\langle$ack $v, ph\rangle$ and $\langle$ack $v', ph\rangle$ are sent by correct processes, then $v = v'$.*

PROOF. Suppose a correct process p sends $\langle$ack $v, ph\rangle$ and a correct process p' sends $\langle$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($\langle$vote $v, ph\rangle, j$). Similarly, there is a set B of $\ell - t$ identifiers j for which p' performs ACCEPT($\langle$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($\langle$vote $v, ph\rangle$) and BROADCAST($\langle$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 $\langle$ack $v, ph\rangle$ from $\ell - t > t$ different identifiers, so some correct process must have sent $\langle$ack $v, ph\rangle$. Similarly, some correct process must have sent $\langle$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 [9]. It is given in Appendix A.1, 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*

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

```
1  locks = ∅
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 ∈ proper such that there is no pair (w, ∗) ∈ locks for any w ≠ v
8      BROADCAST(⟨propose V, ph⟩)                                      /* superround 1 */
9      /* beginning of superround 2 of phase */
10     if i = (ph mod ℓ) + 1 and there is some value v_lock such that the process has performed ACCEPT(⟨propose V_j, ph⟩, j)
11             from ℓ − t different identifiers j with v_lock ∈ V_j
12         then send ⟨lock v_lock, ph⟩ 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 ⟨lock v, ph⟩ from identifier (ph mod ℓ) + 1 and
15             has performed ACCEPT(⟨propose V_j, ph⟩, j) for ℓ − t different identifiers j with v ∈ V_j
16         then choose one such v and perform BROADCAST(⟨vote v, ph⟩)  /* superround 3 */
17     /* beginning of superround 4 of phase */
18     if for some v, the process has performed ACCEPT(⟨vote v, ph⟩, j) from ℓ − t different identifiers j
19         then   add (v, ph) to locks and remove any other pair (v, ∗) from locks
20                send ⟨ack v, ph⟩ to all processes                    /* round 1 of superround 4 */
21     if i = (ph mod ℓ) + 1 and the process has received ⟨ack v_lock, ph⟩ from ℓ − 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 ⟨decide v⟩ to all processes                        /* round 2 of superround 4 */
25     if for some v, the process has received ⟨decide v⟩ from t + 1 different identifiers j in this round
26         then decide v (but continue running the algorithm)
27     for each (v1, ph1) ∈ locks
28         if for some v2 ≠ v1 and ph2 > ph1, the process has performed ACCEPT(⟨vote v2, ph2⟩, j) for ℓ − t
29                 different identifiers j
30             then remove (v1, ph1) from locks
31     ph = ph + 1
```

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

$\langle ack\ v, ph\rangle$ message in phase ph. Then, at all times after phase ph, each correct process that sent $\langle ack\ v, ph\rangle$ 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 Appendix A.2). 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 Appendix A.3. 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 Appendix A.4 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, 22]). Some work considered shared memory for the "wake up" problem [12], 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 [5,6,8,11,16,19]. In [15], 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].

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, 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.

[9] 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.

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

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

[12] 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.

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

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

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

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

APPENDIX

A. APPENDIX

A.1 Detailed Proof of Proposition 5

The proof of the proposition below is included for the sake of completeness, but it is very similar to earlier proofs of [9,20].

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

PROOF. The implementation of an authenticated broadcast primitive is a straightforward generalization of the ones given in [9, 20] for systems with unique identifiers. To perform BROADCAST(m) in superround r, a process sends a message $\langle \text{init } m\rangle$ in the first round of superround r. Any process that receives this message from identifier i sends $\langle \text{echo } m, r, i\rangle$ 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 $\langle \text{echo } m, r, i\rangle$ from $\ell - 2t$ distinct identifiers sends a message $\langle \text{echo } m, r, i\rangle$. If, at any time, a process has received the message $\langle \text{echo } m, r, i\rangle$ from $\ell - t$ distinct identifiers, the process performs ACCEPT(m, i).

Correctness: If a process with identifier i performs BROADCAST(m) in some superround $r \geq T$, then all correct processes send $\langle \text{echo } m, r, i\rangle$ messages in the second round of superround r. All of these messages will be delivered and they will come from at least $\ell - t$ different identifiers, so all processes will perform ACCEPT(m, i) in the second round of superround r.

Unforgeability: Suppose all processes with identifier i are correct and none perform BROADCAST(m). The only reason a correct process will send a message $\langle \text{echo } m, r, i\rangle$ (for some r) is if it has previously received $\langle \text{echo } m, r, i\rangle$ messages from $\ell - 2t > t$ identifiers, one of which must have been sent by a correct process. Thus, no correct process can send the first $\langle \text{echo } m, r, i\rangle$ message. So, no process can receive $\langle \text{echo } m, r, i\rangle$ from $\ell - t > t$ identifiers. It follows that no correct process performs ACCEPT(m, i).

Relay: Suppose some correct process P performs ACCEPT(m, i) during superround r. For some r', P has received $\langle \text{echo } m, r', i\rangle$ messages from $\ell - t$ different identifiers. At least $\ell - 2t$ of those messages were sent by correct processes. Each of those $\ell - 2t$ processes continue to send $\langle \text{echo } m, r', i\rangle$ in every round after superround r. Thus, in superround $\max(r+1, T)$ every correct process sends $\langle \text{echo } m, r', i\rangle$ and all of these messages are delivered, so every correct process performs ACCEPT(m, i). □

We now complete the proof that the algorithm given in Figure 5 correctly solves Byzantine agreement in the basic partially synchronous model when $\ell > \frac{n+3t}{2}$. The following lemma is used to ensure agreement between values decided on line 22 in *different* phases.

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

PROOF. To derive a contradiction, suppose the claim is false. Let A be the set of $\ell - t$ identifiers of the processes that sent an $\langle \text{ack } v, ph\rangle$ message in phase ph. Consider the first time the claim is violated: some correct process q that sent an $\langle \text{ack } v, ph\rangle$ message removes its lock on v (on line 30). This means that there is some $v' \neq v$ and $ph' > ph$ such that q has performed ACCEPT($\langle \text{vote } v', ph'\rangle, j$) for $\ell - t > t$ different identifiers j, at least one of which must belong only to correct processes. By unforgeability, some correct process performed BROADCAST($\langle \text{vote } v', ph'\rangle$). That process must have performed ACCEPT($\langle \text{propose } V_j, ph'\rangle, j$) for $\ell - t$ different identifiers j with $v' \in V_j$. Let B be this set of identifiers.

By Lemma 7, some identifier $j \in A \cap B$ belongs to only one correct process and no Byzantine processes. Let r be the correct process with this identifier j. Since r's identifier is in A, r sent $\langle \text{ack } v, ph\rangle$ in phase ph. Since r's identifier is in B, it follows from unforgeability that r performed a BROADCAST($\langle \text{propose } V_j, ph'\rangle$) with $v' \in V_j$. According to line 7, this is possible only if $(v, *)$ is not in r's *locks* set at the beginning of phase ph'. This contradicts our assumption that all correct processes that sent an $\langle \text{ack } v, ph\rangle$ message in phase ph (including r) keep the value in v in their *locks* set from the time they execute line 20 of phase ph until they execute line 30 of phase ph'. □

The following lemmas are useful for proving termination. Recall that all messages sent after T are guaranteed to be delivered.

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 $v_1 = v_2$.*

PROOF. Since, at the end of phase ph_3, correct processes have locks associated with phases ph_1 and ph_2, we must have $ph_1 \leq ph_3$ and $ph_2 \leq ph_3$. If $ph_1 = ph_2$, then $v_1 = v_2$ by Lemma 8. So for the rest of the proof assume, without loss of generality, that $ph_2 > ph_1$. Before process p_2 added (v_2, ph_2) to its *locks* set in phase ph_2, it performed ACCEPT($\langle \text{vote } v_2, ph_2\rangle, j$) for $\ell - t$ different identifiers j. By the relay property of the authenticated broadcasts, p_1 will accept all of these messages by the end of phase ph_3 and remove (v_1, ph_1) from its *locks* set, if $v_1 \neq v_2$. Thus, v_1 must be equal to 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.*

PROOF. By Lemma 11, at most one value will be appear in the *locks* variables of correct processes at the end of phase $ph - 1$. We consider two cases.

Case 1: the *locks* set of some correct process q is non-empty at the end of phase $ph - 1$. Let (v, ph_v) be the (unique) entry in the *locks* set of q, where ph_v must be smaller than

ph. Then q performed ACCEPT($\langle$vote $v, ph_v\rangle, j$ for $\ell - t > t$ different identifiers, including some identifier j that does not belong to any Byzantine process. Thus, some correct process q performed BROADCAST($\langle$vote $v, ph_v\rangle$). So, q performed ACCEPT($\langle$propose $V_j, ph_v\rangle, j$) from $\ell - t \geq 2t + 1$ different identifiers j with $v \in V_j$. At least $t + 1$ of those identifiers do not belong to any Byzantine process. So correct processes with at least $t + 1$ different identifiers performed BROADCAST($\langle$propose $V_j, ph_v\rangle$), which means v is in the *proper* set of correct processes with at least $t+1$ different identifiers at the beginning of phase ph_v. Thus, by the end of phase $ph - 1$, v will be in the *proper* set of every correct process. It follows that, in phase ph, every correct process will BROADCAST($\langle$propose $V, ph\rangle$) with $v \in V$, and process p will be able to find a value that it can send in superround 2 of phase ph.

Case 2: the *locks* set of every correct process is empty at the end of phase $ph - 1$. If there are $t + 1$ correct processes with the same input value, that value will be in the *proper* set of all correct processes by the beginning of phase ph. Otherwise, all input values will be in the *proper* set of all correct processes at the beginning of phase ph. Either way, some value will appear in the propose message that is broadcast by every correct process in phase ph, so p will be able to find a value that it can send in superround 2 of phase ph. □

We are now ready to prove the correctness of the algorithm.

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

PROOF. We prove each of the three correctness properties of the algorithm in Figure 5 in turn.

Validity: Suppose all correct processes have the same input value, v_0. Then no correct process ever adds any other value to its *proper* set. So, a correct process can perform a BROADCAST($\langle$propose $V, *\rangle$) message only if $V \subseteq \{v_0\}$. It follows from unforgeability that a correct process can perform an ACCEPT($\langle$propose $V, *\rangle, j$) only if $V \subseteq \{v_0\}$ or a Byzantine process has identifier j. Thus, according to the test on line 15, a correct process can perform BROADCAST($\langle$vote $v, *\rangle$) only if $v = v_0$. Then, according to the test on line 18, a correct process can send a message $\langle$ack $v, *\rangle$ only if $v = v_0$. Then, according to the test on line 21, a correct process can decide v only if $v = v_0$. Thus, no correct process decides a value different from v_0 on line 22. A process can decide a value different from v_0 on line 26 only if at least one correct process has already decided that value on line 22, so no correct process will decide a value different from v_0 on line 26.

Agreement: If no correct processes ever decide, agreement is trivially satisfied. A correct process decides a value v on line 26 only if some correct process has previously decided v. Thus, for agreement, it suffices to prove that all values decided by correct processes on line 22 are identical. So, for the remainder of the proof of agreement, we only consider processes that decide on line 22.

Suppose phase ph_1 is the first phase during which some correct process decides. By Lemma 9 there is a unique value v_1 that correct processes decide during phase ph_1. Let p be a correct process that decides v_1 during phase ph_1. Then p received $\langle$ack $v_1, ph_1\rangle$ messages from $\ell - t$ different identifiers. Let A be this set of $\ell - t$ identifiers.

Suppose some correct process p decides a value v in a phase $ph > ph_1$. We shall prove that $v = v_1$. Process p must have performed ACCEPT($\langle$propose $V_k, ph\rangle, k$) from $\ell - t$ different identifiers k with $v \in V_k$. Let B be this set of $\ell - t$ identifiers. By Lemma 7, some identifier $k \in A \cap B$ belongs to only one correct process and no Byzantine processes. Let q be the correct process with this identifier k. Since $k \in A$, q sent an $\langle$ack $v_1, ph_1\rangle$ message in phase ph_1. By Lemma 10, $(v_1, *)$ is in the *locks* set of q at the beginning of phase ph. Thus, no process with identifier k performs BROADCAST($\langle$propose $V_k, ph\rangle$) unless $V_k \subseteq \{v_1\}$. By unforgeability, no correct process can perform ACCEPT($\langle$propose $V_k, ph\rangle, k$) unless $V_k \subseteq \{v_1\}$. Since $k \in B$, process p did perform ACCEPT($\langle$propose $V_k, ph\rangle, k$) and $v \in V_k$. Thus, $v = v_1$. This completes the proof of the agreement property.

Termination: First, we show that if p is any correct process that does not share its identifier with any other process, then p terminates. Let ph be a phase such that $(ph \bmod \ell) + 1$ is p's identifier and phase $ph - 1$ occurs after T. By Lemma 12, there is some value v such that p sends a $\langle$lock $v, ph\rangle$ message in superround 2 of phase ph. Every correct process receives this message, and no other lock messages are received from a process with identifier $(ph \bmod \ell) + 1$ in this phase. According to the test in line 10, p must have performed ACCEPT($\langle$propose $V_j, ph\rangle, j$) for $\ell - t$ different identifiers j with $v \in V_j$ during superround 1 of phase ph. By the relay property, all correct processes must have performed these ACCEPT actions by the end of superround 2 of phase ph. Thus, every correct process performs BROADCAST($\langle$vote $v, ph\rangle$) during superround 3 of phase ph and all correct processes accept this broadcast. Thus, all correct processes send $\langle$ack $v, ph\rangle$ in round 1 of superround 3 of phase ph. Process p receives all of these messages and decides v.

There are at least $2t + 1$ correct processes that do not share their identifier with any other process (since $n \leq 2\ell - 3t$). By the argument above, these will all decide. By the agreement property, they will all decide on the same value v. Eventually, all of these $2t+1$ processes will send $\langle$decide $v\rangle$ messages in superround 4 of each phase, and all correct processes will receive these messages and decide on line 26. □

A.2 Detailed Proof of Proposition 16

As described in Section 5.1, suppose there is an algorithm $\mathcal{A}$ that solves Byzantine agreement with $\ell \leq t$. We consider only executions of $\mathcal{A}$ with some fixed set of ℓ Byzantine processes, chosen so that each of the ℓ identities is held by one Byzantine process.

We use C_k to denote a configuration at end of round k. From C_k, the system can reach different possible configurations C_{k+1}. In an execution of algorithm $\mathcal{A}$, the configuration C_{k+1} is completely determined by (1) C_k and (2) the messages sent by the Byzantine processes to the correct

processes in round $k+1$. (The messages sent by correct processes are determined by C_k and $\mathcal{A}$.)

Lemma 21 *There is a multivalent initial configuration.*

PROOF. For $0 \leq j \leq n-\ell$, let C_0^j be the initial configuration where the first j correct processes have input 1 and the rest of the correct processes have input 0. By validity, C_0^0 is 0-valent and $C_0^{n-\ell}$ is 1-valent. Choose j so that C_0^j is 0-valent and C_0^{j+1} is not 0-valent. Only one correct process is in a different state in these two initial configurations, so there is an execution from C_0^{j+1} that decides 0, by Lemma 17. Thus, C_0^{j+1} is multivalent. □

Lemma 22 *Every multivalent configuration of $\mathcal{A}$ has a multivalent successor configuration.*

PROOF. Suppose the claim is false. Then there exists a multivalent configuration C_θ of $\mathcal{A}$ such that every successor configuration of C_θ is univalent. Then, some successor configuration $C_{\theta+1}$ is v-valent and some successor configuration $C'_{\theta+1}$ is v'-valent, where $v \neq v'$. For $0 \leq j \leq n-\ell$, let $C^j_{\theta+1}$ be the successor of C_θ where, in round $\theta+1$, the Byzantine processes send the same messages to the first j correct processes as they do in $C'_{\theta+1}$, and send the same messages to the rest of the processes as they do in $C_{\theta+1}$. Then, $C^0_{\theta+1} = C_{\theta+1}$ is v-valent and $C^{n-\ell}_{\theta+1} = C'_{\theta+1}$ is v'-valent. Choose j so that $C^j_{\theta+1}$ is v-valent and $C^{j+1}_{\theta+1}$ is not v-valent. Only one correct process is in a different state in these two configurations, so by Lemma 17, some execution from $C^{j+1}_{\theta+1}$ decides v. Thus, $C^{j+1}_{\theta+1}$ is multivalent, contradicting the assumption. □

This lemma implies that, starting from the multivalent initial configuration of $\mathcal{A}$, we can construct an infinite execution consisting only of multivalent configurations. No correct process can ever decide in this execution, which violates the termination condition of consensus, so algorithm $\mathcal{A}$ cannot exist. This completes the proof of Proposition 16.

A.3 Proof of Proposition 18

In this section we give an algorithm to prove the following proposition.

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

The approach used to design the algorithm is similar to the one used in Section 4.2, but various thresholds must be adjusted to take advantage of the restriction on the Byzantine processes, and to make use of the processes' ability to count copies of identical messages. In combination, these two factors allow us to weaken the condition on number of identifiers from $\ell > \frac{n+3t}{2}$ (which is required in Section 4.2) to $\ell > t$. The safety of our algorithm depends on the condition $n > 3t$, while liveness is guaranteed by the condition $\ell > t$. The algorithm uses a more powerful version of the authenticated broadcast primitive [20].

A.3.1 Authenticated Broadcast

The computation proceeds in superrounds. Superround r is composed of the two rounds $2r$ and $2r+1$. Our authenticated broadcast is defined by two primitives: BROADCAST(i, m, r), where i is the identifier of the broadcaster, m is the message and r is the superround number, and ACCEPT(i, α, m, r) where α is an integer (α is an estimate of the number of processes with identifier i that have broadcast m in superround r).

The authenticated broadcast primitive is specified as follows. Consider any execution that uses authenticated broadcasts. Let T be the first superround such that all messages sent during or after superround T are delivered. Let f_i be the number of Byzantine processes with identifier i. The f_i values are used only in the specification of the authenticated broadcast; they are not known by the processes.

- Correctness: If α correct processes with identifier i perform BROADCAST(i, m, r) in superround $r \geq T$ then every correct process performs ACCEPT(i, α', m, r) with $\alpha' \geq \alpha$ during superround r.
- Relay: If a correct process performs ACCEPT(i, α, m, r) in superround $r' \geq r$ then every correct process performs ACCEPT(i, α', m, r) with $\alpha' \geq \alpha$ in superround $\max(r', T) + 1$.
- Unforgeability: If α correct processes with identifier i perform BROADCAST(i, m, r) in superround r and some correct process performs ACCEPT(i, α', m, r) in superround r' then $r \leq r'$ and $0 \leq \alpha' \leq \alpha + f_i$.
- Unicity: for each i, m and r, each correct process performs at most one ACCEPT$(i, *, m, r)$ per superround.

We give an implementation of authenticated broadcast in Figure 6. In the algorithm, we call a message sent by some process with identifier i at round R *valid* if

- it contains at most one tuple (init, i, m, r) and $2r = R$ in that tuple, and
- for each j, m and r, it contains at most one tuple $(\text{echo}, j, m, *, r)$ and $R \geq 2r$ in that tuple.

All messages sent by correct processes are valid.

In the following, we consider a run where, for each i, f_i is the number of Byzantine processes with identifier i. Let $\Pi_R(h, m, k)$ be the set of correct processes that send a message containing $(\text{echo}, h, *, m, k)$ in round R.

Lemma 23 *Assume that α correct processes with identifier i perform* BROADCAST(i, m, r) *in round $2r$. Let $R \geq 2r+1$ be a round. For every q in $\Pi_R(i, m, r)$, if q sends $(\text{echo}, i, a_q, m, r)$ in round R then $0 \leq a_q \leq \alpha + f_i$.*

PROOF. We prove this lemma by induction.

- $R = 2r+1$: Each of the α correct processes with identifier i who perform BROADCAST(i, m, r) in superround

Code for process with identifier $i \in \{1, 2, \ldots, \ell\}$

Variable:

1 $a[h, m, r] = 0$ for all h, m and r

Main code:

2 IN ROUND R
3 $\mathcal{M} = \emptyset$
4 For all $h \in \{1, 2, \ldots, \ell\}$
5 For all $m \in$ possible messages
6 For all $k \in \{1, \ldots, R/2\}$
7 **if** $a[h, m, k] \neq 0$ **then** $\mathcal{M} = \mathcal{M} \cup \{(\text{echo}, h, a[h, m, k], m, k)\}$
8 **if** $R = 2r$ **then**
9 To perform BROADCAST(i, m, r) : $\mathcal{M} = \mathcal{M} \cup \{(\text{init}, i, m, r)\}$
10 send $(\mathcal{M}, R)$ to all processes
11 For all $h \in \{1, 2, \ldots, \ell\}$
12 For all $m \in$ possible messages
13 **if** $(R = 2r)$ and the set of valid received messages contains $\alpha \geq 1$ messages (init, h, m, r) **then**
14 $a[h, m, r] = \alpha$
15 For all $k \in \{1, ..., R/2\}$
16 **if** at least $n - 2t$ valid messages containing $(\text{echo}, h, *, m, k)$ were received this round **then**
17 $\alpha_1 = \max\{\alpha :$ at least $n - 2t$ messages of the contained $(\text{echo}, h, \alpha', m, k)$ with $\alpha' \geq \alpha\}$
18 $a[h, m, k] = \max(\alpha_1, a[h, m, k])$
19 **if** R is odd and at least $n - t$ valid messages containing $(\text{echo}, h, *, m, k)$ were received this round **then**
20 $\alpha_2 = \max\{\alpha :$ at least $n - t$ of the messages contained $(\text{echo}, h, \alpha', m, k)$ with $\alpha' \geq \alpha\}$
21 ACCEPT(h, α_2, m, k)

Figure 6: An authenticated broadcast primitive for n partially synchronous, numerate processes and ℓ identifiers in a system with restricted Byzantine processes.

r sends (init, i, m, r) in round $2r$. Each correct process receives at most $\alpha + f_i$ valid messages containing (init, i, m, r) in round $2r$. Thus, if a process q in $\Pi_{2r+1}(i, m, r)$ sends $(\text{echo}, i, a_q, m, r)$ in round $2r + 1$ then $0 \leq a_q \leq \alpha + f_i$.

- Let $R > 2r + 1$. Assume the lemma is true for round $R - 1$. Let q be a correct process in $\Pi_R(i, m, r)$. In line 20 of round $R - 1$, process q either did not change $a[i, m, r]$ or set it to α_1. If $a[i, m, r]$ did not change, then q sends the same tuple $(\text{echo}, i, a_q, m, r)$ that it sent in the previous round, and the claim follows from the induction hypothesis. Otherwise, suppose q changed $a[i, m, r]$ to α_1 in round $R - 1$. Then, q must have received at least $n - 2t > t + 1$ messages containing tuples of the form $(\text{echo}, h, \alpha', m, k)$ with $\alpha' \geq \alpha_1$ in the previous round. At least one of those messages was from a correct process, which had $\alpha' \leq \alpha + f_i$ by the induction hypothesis. Thus, $\alpha_1 \leq \alpha + f_i$ and the claim follows.

□

Lemma 24 *Assume that α correct processes with identifier i perform* BROADCAST(i, m, r) *in round $2r \geq T$. Let $R \geq 2r + 1$. Then,*

(1) if $\alpha > 0$ then $\Pi_R(i, m, r)$ is the set of correct processes,

(2) $\Pi_{R-1}(i, m, r) \subseteq \Pi_R(i, m, r)$, and

(3) for every q in $\Pi_R(i, m, r)$, q sends $(\text{echo}, i, a_q, m, k)$ in round R with $a_q \geq \alpha$ in round R.

PROOF. We prove this lemma by induction.

- $R = 2r + 1$: Each of the α correct processes with identifier i who perform BROADCAST(i, m, r) in superround r sends (init, i, m, r) in round $2r$. Since $2r \geq T$, every correct process receives at least α messages containing (init, i, m, r) in round $2r$. If $\alpha > 0$, every correct process q sends $(\text{echo}, i, a_q, m, r)$ in round $2r + 1$ with $a_q \geq \alpha$. Thus, we have (1) and (3). From the algorithm, a correct process never sends $(\text{echo}, i, *, m, r)$ in round $2r$, so $\Pi_{2r}(i, m, r) = \emptyset$. Thus, we have (2).
- Let $R > 2r + 1$. Assume the properties (1) , (2) and (3) are true for round $R - 1$. Property (2) follows from the fact that $a[h, m, k]$ never decreases. Claim (1) for round R follows from (2) and claim (1) of the induction hypothesis. To prove claim (3), consider any process q in $\Pi_R(h, m, r)$. If $\alpha = 0$, claim (3) is trivially true, so assume $\alpha > 0$. In line 20 of round $R - 1$, process q either did not change $a[i, m, r]$ or set it to α_1. If $a[i, m, r]$ did not change, then q sends the same tuple $(\text{echo}, i, a_q, m, r)$ that it sent in the previous round, and the claim follows from the induction hypothesis. Otherwise, suppose q changed $a[i, m, r]$ to α_1 in round $R - 1$. By claim (1) and (3) of the induction hypothesis each of the $n - t$ correct processes sends $(\text{echo}, i, \alpha', m, r)$ with $\alpha' \geq \alpha$ in round $R - 1$. Since $R > T$, all of these messages are received by q, so $\alpha_1 \geq \alpha$. Thus, q sends $(\text{echo}, i, \alpha_1, m, r)$ with $\alpha_1 \geq \alpha$ in round R.

□

Proposition 25 (Unicity) *For each i, m and r, each cor-*

rect process performs at most one ACCEPT$(i, *, m, r)$ *per superround.*

PROOF. This follows directly from the code. □

Proposition 26 (Correctness) *If α correct processes with identifier i performs* BROADCAST(i, m, r) *in superround $r \geq T$ then every correct process performs accept(i, α', m, r) with $\alpha' \geq \alpha$ in superround r.*

PROOF. Each of the α correct processes with identifier i who perform BROADCAST(i, m, r) in superround $r \geq T$ sends (init, i, m, r) in round $2r$. By Lemma 24, every correct process q sends $(\text{echo}, i, \alpha_q, m, r)$ in round $2r+1$, with $\alpha_q \geq \alpha$. All of these messages are delivered. Thus, every correct process will set α_2 to a value greater than or equal to α on line 20 and then perform ACCEPT(i, α_2, m, r) at the end of superround r. □

Proposition 27 (Relay) *If a correct process performs* ACCEPT (i, α, m, r) *in superround $r' \geq r$ then every correct process performs* ACCEPT(i, α', m, r) *with $\alpha' \geq \alpha$ in superround* $\max(r', T) + 1$.

PROOF. Assume some correct process p performs ACCEPT (i, α, m, r) in superround r'. Then it must do so in round $2r' + 1$ (since a correct process accepts only in the second round of the superround). Process p must have received at least $n - t$ messages containing tuples of the form $(\text{echo}, i, \alpha', m, r)$ with $\alpha' \geq \alpha$ in this round. Among the $n-t$ senders of these messages, at least $n - 2t$ are correct. Since the value stored in each sender's $a[i, m, r]$ variable can only increase, each of these $n - 2t$ correct senders also sends a tuple of the form $(\text{echo}, i, \alpha', m, r)$ with $\alpha' \geq \alpha$ in round $\max(r', T)$. All of these messages are delivered. Thus, for each correct process, the value of $a[i, m, r]$ is at least α after the process executes line 18 in superround $\max(r', T)$. Then, in superround $\max(r', T) + 1$, each of the $n - t$ correct processes sends a tuple of the form $(\text{echo}, i, \alpha', m, r)$ with $\alpha' \geq \alpha$. All of these messages are delivered. Thus, each correct process performs *accept*(i, α', m, r) with $\alpha' \geq \alpha$ in superround $\max(r', T) + 1$. □

Proposition 28 (Unforgeability) *If α correct processes with identifier i perform* BROADCAST(i, m, r) *in superround r and some correct process performs* ACCEPT(i, α', m, r) *in superround r' then $r \leq r'$ and $0 \leq \alpha' \leq \alpha + f_i$.*

PROOF. Assume that some correct process q performs ACCEPT(i, α', m, r) in superround r'. Then it received at least $n-t$ messages containing tuples of the form $(\text{echo}, h, \alpha'', m, k)$ with $\alpha'' \geq \alpha'$. Because $n - t \geq t+1$, one of those messages came from a correct process. By Lemma 23, the α'' in that message is less than or equal to $\alpha + f_i$, so $\alpha' \leq \alpha + f_i$ and $\alpha' \geq 0$ follows directly from the code. □

Thus, we have the following theorem.

Theorem 29 *The algorithm in Figure 6 ensures the unicity, correctness, unforgeability and relay properties.*

A.3.2 Consensus algorithm

A partially synchronous algorithm for Byzantine agreement when $n > 3t$ and $\ell > t$ is shown in Figure 7. It uses the authenticated broadcast primitive described in the previous subsection and follows the same general pattern as the algorithm of Dwork, Lynch and Stockmeyer [9]. Each iteration of the main loop is called a *phase*, which takes four superrounds.

Each process has a *proper* variable, which stores a set of values that can be output without violating validity. Initially, only the process's own value is in this set. In each round, each process updates its *proper* variable as follows. Each process appends its *proper* set to each message it sends. If a process receives *proper* sets containing v in $t+1$ messages in the same round, it adds v to its own *proper* set. Also, if a process has received *proper* sets in $2t+1$ messages during the round and no value appears in $t+1$ of them, the process adds all possible input values to its own *proper* set.

Consider a process p executing the algorithm. There are several times when p needs to have an estimate of the number of processes that performed a broadcast of a particular message m in an earlier superround $r \leq r'$. During superround r', p performs a number of ACCEPT(i, α_i, m, r). For each identifier i, α_i is p's estimate of the number of processes with identifier i that performed BROADCAST(i, m, r). We say that the number of *witnesses* that p has in superround r' for (m, r) is the sum, over all i, of the α_i's that appear in all ACCEPT(i, α_i, m, r) actions that p performs during superround r'. It follows from the properties of authenticated broadcast that this estimate will eventually be at least as large as the actual number of correct processes that performed BROADCAST$(*, m, r)$ and exceed that number by at most t.

For the remainder of this section, we consider an execution in which f_i processes with identifier i are Byzantine processes, for each identifier i. Let $f = \sum_{i=1}^{\ell} f_i$ be the total number of Byzantine processes in the execution.

Lemma 30 *If some correct process p has $n-t$ witnesses for (m, r) in some superround $r' \geq r$, then there are at least $n - t - f$ correct processes that performed* BROADCAST$(*, m, r)$ *in round r.*

PROOF. For each identifier i, let α_i be the number of correct processes with identifier i that perform BROADCAST(i, m, r) in round r. By unforgeability, if p performs ACCEPT(i, α_i', m, r) in superround r', then $\alpha_i' \leq \alpha_i + f_i$. Thus, the number of witnesses that p has for (m, r) in superround r' is at most $\sum_{i=1}^{\ell}(\alpha_i + f_i) = (\sum_{i=1}^{\ell} \alpha_i) + f$. So if p has $n-t$ witnesses for (m, r) in superround r', then $\sum_{i=1}^{\ell} \alpha_i \geq n - t - f$, as required. □

Lemma 31 *If some correct process has $n - t$ witnesses for (m, r) and some correct process has $n - t$ witnesses for (m', r'), then some correct process performed both* BROADCAST $(*, m, r)$ *and* BROADCAST$(*, m', r')$.

Code for process with identifier $i \in \{1, 2, ..., \ell\}$

```
1   locks = ∅ ;
2   ph = 0;
3   proper = {v} /* v is the input value for the process (see page  for how this variable is updated each round) */
4   while true
5       /* beginning of superround 1 of phase */
6       Let V be the set of values v ∈ proper such that there is no pair (w, ∗) ∈ locks for any w ≠ v
7       for each v ∈ V do BROADCAST(i, propose v, 4ph)
8       /* beginning of superround 2 of phase */
9       if i = ph mod ℓ + 1 and there is some value v such that there are at least n − t witnesses for (propose v, 4ph)
10          then send ⟨lock, v, ph⟩ to all processes /* round 1 of superround 2 */
11      /* beginning of superround 3 of phase*/
12      if there is some value v for which the process received ⟨lock, v, ph⟩ from processes with identifier ph mod ℓ + 1
13              and there are at least n − t witnesses for (propose v, 4ph)
14          then choose one such value v and perform BROADCAST(i, vote v, 4ph + 2)
15      /* beginning of superround 4 of phase */
16      if for some v, there are at least n − t witnesses for (vote v, 4ph + 2)
17          then
18              add (v, ph) to locks and remove any other pair (v, ∗) from locks
19              send ⟨ack, v, ph⟩ to all processes /* round 1 of superround 4 */
20      if for some v:
21              There are at least n − t witnesses for (propose v, 4ph) and
22              received n − t messages ⟨ack, v, ph⟩ in this round
23          then decide v (but continue running the algorithm)
24      for each (v1, ph1) ∈ locks
25          if for some v2 ≠ v1 and ph2 > ph1, there are n − t witnesses for (vote v2, 4ph2 + 2)
26              then remove (v1, ph1) from locks
27      ph = ph + 1
```

Figure 7: Partially Synchronous Byzantine agreement algorithm with n processes and ℓ identifiers.

PROOF. By Lemma 30, there is a set A of at least $n - t - f$ correct processes that performed BROADCAST$(*, m, r)$ and there is a set B of at least $n - t - f$ correct processes that performed BROADCAST$(*, m', r')$. Since there are $n - f$ correct processes, $|A \cap B| = |A| + |B| - |A \cup B| \geq (n - t - f) + (n - t - f) - (n - f) = n - 2t - f \geq n - 3t > 0$, so there is at least one process in $A \cap B$. □

Lemma 32 *If the messages $\langle ack, v, ph\rangle$ and $\langle ack, v', ph\rangle$ are both 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 $\langle \text{ack}, v', ph\rangle$. According to line 16, process p has $n - t$ witnesses for (vote $v, 4ph + 2$) in superround 3 of phase ph. Similarly, p' has $n-t$ witnesses for (vote $v, 4ph+2$). By Lemma 31, there is at least one correct process that performed BROADCAST$(*,$ vote $v, 4ph + 2)$ and performed BROADCAST$(*,$ vote $v', 4ph + 2)$ in superround 3 of phase ph, so $v = v'$. □

Lemma 33 *If two correct processes decide on line 23 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 $n-t$ copies of $\langle \text{ack}, v, ph\rangle$, 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 22, $v = v'$. □

Lemma 34 *At the end of each phase, the locks set of a correct process contains at most one pair.*

PROOF. Let p be a correct process. We first prove that in each phase ph, p can add at most one pair to its *locks* set. For each pair (v, ph) added in phase ph, p has $n-t$ witnesses for (vote $*, 4ph + 2$). By Lemma 31 this condition can be true for at most one value v. When p adds this unique pair (v, ph) to *locks*, it removes all other pairs $(v, *)$. Then in line 24 to 26, p will remove all other pairs (v', ph') from the *locks* set. □

Lemma 35 *Suppose that some correct process receives $n-t$ $\langle ack, v, ph\rangle$ messages. Then at all times after phase ph, each correct process that sent $\langle ack, v, ph\rangle$ in phase ph has a pair (v, ph') with $ph' \geq ph$ in its locks set.*

PROOF. To derive a contradiction, suppose the claim is false. Let A be the set of $n - 2t$ correct processes that sent $\langle \text{ack}, v, ph\rangle$ in phase ph. Consider the first time the claim is violated: some correct process q that sent an $\langle \text{ack}, v, ph\rangle$ message removes its lock on v (on line 26). This means that there is some $v' \neq v$ and $ph' > ph$ such that q has $n - t$ witnesses for (vote $v', 4ph' + 2$). By Lemma 30, some correct process performs BROADCAST$(*,$ vote $v', 4ph'+2)$. That process has $n - t$ witnesses for (propose $v', 4ph'$) in superround 2 of phase ph'. By Lemma 30 there is a set B of $n-t-f$ correct processes that perform BROADCAST$(*,$ propose $v', 4ph')$. Since there are $n-f$ correct processes, $|A \cap B| = |A| + |B| -$

$|A \cup B| \geq (n-2t)+(n-t-f)-(n-f) = n-3t > 0$. Thus, there is at least one correct process r that sends $\langle \text{ack}, v, ph\rangle$ in phase ph and performs BROADCAST$(*, \text{propose } v', 4ph')$ in phase ph'. According to line 6, this is possible only if $(v, *)$ is not in r's *locks* set at the beginning of phase ph'. This contradicts our assumption that all correct processes that sent an $\langle \text{ack}, v, ph\rangle$ message in phase ph keep the value v in their *locks* set from the time they execute line 12 of phase ph until they execute line 26 of phase ph'.

□

Lemma 36 *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 $v_1 = v_2$.*

PROOF. Since, at the end of phase ph_3, correct processes have locks associated with phases ph_1 and ph_2, we must have $ph_1 \leq ph_3$ and $ph_2 \leq ph_3$. If $ph_1 = ph_2$ then $v_1 = v_2$ follows from Lemma 32 (because just after p_i adds (v_i, ph_i) to its *locks* set, it sends $\langle \text{ack}, v_i, ph_i\rangle$). So for the rest of the proof assume, without loss of generality, that $ph_2 > ph_1$. Before process p_2 added (v_2, ph_2) to its *locks* set in phase ph_2, it has $n-t$ witnesses for (vote $v_2, 4ph_2+2$). By the relay property of the authenticated broadcasts, p_1 will accept all of these messages by the end of phase ph_3 and remove (v_1, ph_1) from its locks set, if $v_1 \neq v_2$. Thus, v_1 must be equal to v_2. □

Lemma 37 *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.*

PROOF. By Lemma 36, at most one value will appear in the *locks* variables of correct processes at the end of phase $ph-1$. We consider two cases.

Case 1: the *locks* set of some correct process q is non-empty at the end of phase $ph-1$. Let (v, ph_v) be the entry in the *locks* set of q, where ph_v must be smaller than ph. Then q has $n-t$ witnesses for (vote $v, 4ph_v+2$) in superround 3 of phase ph_v. Thus, some correct process performed BROADCAST$(*, \text{vote } v, 4ph_v+2)$. That process must have $n-t \geq 2t+1$ witnesses for (propose $v, 4ph_v$). By Lemma 30, at least $t+1$ different correct processes performed BROADCAST$(*, \text{propose } v, 4ph_v)$, which means v is in the *proper* set of of at least $t+1$ correct processes at the beginning of phase ph_v. Thus, by the end of phase $ph-1$, v will be in the *proper* set of every correct process. It follows that, in phase ph, every correct process will perform BROADCAST$(*, \text{propose } v, 4ph)$, and process p will be able to find a value that it can send in superround 2 of phase ph.

Case 2: the *locks* set of every correct process is empty at the end of phase $ph-1$. If there are $t+1$ correct processes with the same input value, that value will be in the *proper* set of all correct processes by the beginning of phase ph. Otherwise, every value will be in the *proper* set of all correct processes by the beginning of phase ph. Either way, some value will appear in the propose message that is broadcast by correct processes in phase ph, so p will able to find a value that it can send in superround 2 of phase ph. □

Proposition 38 (Validity) *If all correct processes propose v then no correct process decides a value different from v.*

PROOF. Suppose all correct processes have the same input value v_0. Then no correct process ever adds any other value to its *proper* set. So, a correct process can perform BROADCAST$(*, \text{propose } v, *)$ only if $v = v_0$. Thus, according to the test on line 12, a correct process can perform BROADCAST$(*, \text{vote } v, *)$ only if $v = v_0$. Then, according to the test on line 16, a correct process can send a message $\langle \text{ack}, v, *\rangle$ only if $v = v_0$. Thus, no correct process decides a value different from v_0 on line 23. □

Proposition 39 (Agreement) *If two correct processes decide v and v' then $v = v'$.*

PROOF. If no correct processes ever decide, agreement is trivially satisfied.

Suppose phase ph_1 is the first phase during which some correct process p decides. By Lemma 33, there is a unique value v_1 such that correct processes decide during phase ph_1. From the code, process p has received $n-t$ $\langle \text{ack}, v_1, ph_1\rangle$ messages. Let A be a set of $n-2t$ correct processes that sent $\langle \text{ack}, v_1, ph_1\rangle$.

Suppose some correct q decides a value v_2 in a phase $ph_2 > ph_1$. We shall prove that $v_1 = v_2$. Process q has $n-t$ witnesses for (propose $v_2, 4ph_2$). By Lemma 30, there is a set B of $n-t-f$ correct processes that perform BROADCAST$(*, \text{propose } v_2, 4ph_2)$. Thus, there is some correct process $h \in A \cap B$ that has sent an $\langle \text{ack}, v_1, ph_1\rangle$ in phase ph_1.

By Lemma 35 and 34, $(v_1, *)$ is in the lock set of h at the beginning of phase ph_2 and this is the only pair in the lock set. Thus, it performs only BROADCAST$(*, \text{propose } v_1, 4ph_2)$ in superround $4ph$. But $h \in B$, so it performs BROADCAST$(*, \text{propose } v_2, 4ph_2)$. Thus, $v_1 = v_2$. □

Proposition 40 (Termination) *All correct processes decide.*

PROOF. As there are $\ell > t$ identifiers, there is at least one identifier such that all processes with this identifier are correct. Suppose that it is identifier k. Let ph be a phase such that $ph \bmod \ell + 1 = k$ and phase $ph-1$ occurs after T. By Lemma 37, each process p_j with identifier k sends a $\langle \text{lock}, v_j, ph\rangle$ message in superround 2 of phase ph. According to the test in line 9, p_j must have $n-t$ witnesses for (propose $v_j, 4ph$) during superround 1 of phase ph. By the relay property, all correct processes must have $n-t$ witnesses for (propose $v_j, 4ph$) at the end of superround 2 of phase ph.

Each correct process receives the same set of lock messages from all processes with identifier k and deterministically chooses one of them. Let v be the value chosen by all correct

processes. All correct processes then perform BROADCAST(∗, vote $v, 4ph+2$). Thus, according to the test in line 16, every process has $n-t$ witnesses for (vote $v, 4ph+2$) in superround 3. Thus, all correct processes send $\langle \text{ack}, v, ph \rangle$ in round 1 of superround 4 of phase ph. Every correct process receives all of these messages and has $n-t$ witnesses for (propose $v, 4ph$) in superround 3, and thus decides v. □

A.4 Detailed Proof of Theorem 19

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

PROOF. The synchronous algorithm given in Section 3.2 obviously still works if the Byzantine processes are restricted. To prove that $\ell > 3t$ is necessary, we use a simulation.

Assume that for $\ell < 3t$, there is an algorithm $\mathcal{A}$ that solves Byzantine agreement in a synchronous system H of n processes, ℓ identifiers and t Byzantine processes. Let $A(i)$ the code executed by the processes with identifier i.

Then $\mathcal{A}$ solves also Byzantine agreement in H if we restrict the power of the Byzantine processes: each Byzantine process must send the same message to all processes with the same identifier.

If $\ell \geq t$, the result is clear so we consider that $\ell > t$.

Consider the classical synchronous system (where each process has its own identifier) S, with ℓ processes and at most t Byzantine processes ($0 \leq t < \ell$). Let $\{q_0, q_1, ..., q_{\ell-1}\}$ be these processes . We construct with the help of $\mathcal{A}$ an algorithm $\mathcal{A}'$ that solves Byzantine agreement in the synchronous model S with a number of processes less than $3t$, contradicting the result of [9, 13].

In the distributed algorithm $\mathcal{A}'$ for S, the code of the process q_i is $A(i)$.

We now prove that each execution of the algorithm $\mathcal{A}'$ in S satisfies the specification of Byzantine agreement. Let α_S be an execution of $\mathcal{A}'$, where the input of process q_i is $Input(i)$ and that contains at most t Byzantine processes. Assume that α_S has b Byzantine processes, for example $q_1, \ldots, q_b$, for some b with $0 \leq b \leq t$.

We consider a system H where $(n-\ell+1)$ processes have the identifier 0, and the other processes have identifiers $1, \ldots, \ell-1$ (one process per identifier). We consider an execution α_H of th is system where (1) the processes with identifier i ($1 \leq i \leq b$) are Byzantine, and they send the same messages to the process with identifier j in round r as the Byzantine process q_i sends to q_j in round r of α_S, (2) the process with identifier i ($b+1 \leq i \leq \ell-1$) is correct and has as input $Input(i)$, and (3) all processes with identifier 0 are correct and have input $Input(0)$.

The processes with identifier 0 all have the same input and receive the same messages, so they send the same message $m(r)$ in round r and have the same state at the end of each round. The other processes receive from processes with identifier 0 only the message $m(r)$ in round r.

The process q_i in α_S and a process with identifier i in α_H have the same state at the beginning of each round. As α_H satisfies the specification of Byzantine agreement, the execution α_S satisfies the specification of the Byzantine agreement. We have obtained a distributed algorithm $\mathcal{A}'$ that solves Byzantine agreement in the synchronous model with the number of processes less than $3t$, contradicting the result of [9, 13]. □