Distrib. Comput. (2013) 26:321–340
DOI 10.1007/s00446-013-0190-3

Byzantine agreement with homonyms

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

Received: 26 August 2011 / Accepted: 22 April 2013 / Published online: 26 May 2013

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 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, assuming processes can count the number of messages with the same identifier they receive in a round.

C. Delporte-Gallet (✉) · H. Fauconnier · H. Tran-The
University Paris Diderot, Paris, France
e-mail: caroledelporte@gmail.com

R. Guerraoui
Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

A.-M. Kermarrec
INRIA Rennes-Bretagne Atlantique, Rennes, France

E. Ruppert
York University, Toronto, Canada

Keywords Consensus · Message passing · Process identifiers · Byzantine failures

1 Introduction

We consider a distributed system with Byzantine failures 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*. If a process p receives a message from a process q with identifier i, then p knows that the message was sent by some process with identifier i, but p does not know whether the message was sent by q or another process q' having the same identifier i. This is true even if q is Byzantine: a Byzantine process cannot change its own identifier.

This model generalizes the classical scheme where processes have distinct identifiers (i.e., $\ell = n$), and the less classical scheme where processes are anonymous (i.e., $\ell = 1$). Studying systems with homonyms provides a better understanding of the importance of identifiers in distributed computing. There are two additional motivations for the new model. In systems such as Pastry or Chord [21,24], assuming 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, users of a system may wish to preserve their privacy by remaining anonymous.

Unfortunately, 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 [17], even if the system is synchronous and only one process can be faulty. With a limited number of identifiers, more problems become solvable, and some level of anonymity can be preserved by hiding, to some extent, the association between users and identifiers. For example, users of a distributed protocol might use only their domain names as identifiers. Others will see that some user within the domain is participating, but will not know exactly which one. (In our model, we assume that message recipients can verify the domain from which a message came, so that even Byzantine processes cannot modify their identifier). If several users within the same domain participate in the protocol, they will behave as homonyms.

In this paper we study how many distinct identifiers are needed to reach *agreement* in a system of n processes, up to t of which can be Byzantine. If $n \leq 3t$, even synchronous Byzantine agreement is known to be unsolvable for $\ell = n$ [16,20], so it is also unsolvable in systems with homonyms. Thus, we need only consider systems where $n > 3t$. 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 $\ell > \frac{n+3t}{2}$. (The bound $\frac{n+3t}{2}$ is strictly greater than $3t$ because $n > 3t$). We show that this bound is also tight by giving a new partially synchronous Byzantine agreement algorithm. This bound is somewhat surprising because the number of required identifiers ℓ depends on n as well as t. Counter-intuitively, increasing the number of correct processes can render agreement impossible. For example, if $t = 1$ and $\ell = 4$, agreement is solvable for 4 processes but not for 5. Another difference from the classical situation (where $\ell = n$) is that the condition that makes Byzantine agreement solvable is different for the synchronous and partially synchronous models.

To strengthen our results, we show that (a) both the synchronous and partially synchronous lower bounds hold even if correct processes are *numerate*, i.e., can count the number of processes that send identical messages in a round and (b) the matching algorithms are correct even if processes are *innumerate*. Since a process knows the identifier of the sender of each message it receives, it is trivial for the process to count copies of the messages it receives in a system with unique identifiers ($\ell = n$). However, using identifiers to count copies 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 gets no advantage from 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.

Our 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 Sect. 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 identifier from the set $\{1, \ldots, \ell\}$. 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, so

Table 1 Necessary and sufficient conditions for solving Byzantine agreement in a system of n processes using ℓ identifiers and tolerating t Byzantine failures

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

In all cases, n must be greater than $3t$

they are 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 are given the same algorithm to execute (to avoid hardcoding an identifier into a process's algorithm). Processes with different identifiers may be given different algorithms. 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 $0 < t < n$. As mentioned in the introduction, we assume throughout the paper that $n > 3t$, since it is already known that Byzantine agreement is impossible when $n \leq 3t$, even in synchronous systems with unique ids [16,20]. A Byzantine process may choose to send arbitrary messages (or no message) to each other process. However, we assume a Byzantine process cannot modify its own identifier. Thus, a process will know the true identifier of the sender of each message it receives, even if the sender is Byzantine. 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 m was sent by some process with identifier $m.id$.

Since processes with the same identifier are supposed to be indistinguishable in our model, a correct process cannot send different messages to two processes with the same identifier during a single round. However, Byzantine processes are not constrained in this way: they can send different messages to each process. This is because Byzantine processes are intended to model arbitrarily bad failures in the system, which might, for example, corrupt messages going to individual processes.

In the *synchronous model*, computation proceeds in rounds. In each round, each process can send messages to all other processes and then receive all messages that were sent to it during that round. It is sometimes convenient to assume that correct processes perform broadcasts; there is no loss of generality in assuming this, because if a process wishes to send different messages to processes with different identifiers, it can include the intended recipients' identifiers in the message itself.

For the *partially synchronous model* we use the definition of Dwork, Lynch and Stockmeyer [10]: computation proceeds in rounds, as in the synchronous model. However, in each execution, a finite number of messages might not be delivered to all of their intended recipients. There is no bound on the number of messages that can be dropped. As argued in [10], this basic partially synchronous model is equivalent to other models with partially synchronous communication. More specifically, the model in which message delivery times are eventually bounded by a known constant and the model in which message delivery times are always bounded by an unknown constant can both simulate the basic partially synchronous model. Conversely, each of these models can be simulated by the basic partially synchronous model. Thus, our characterization of the values of n, ℓ and t for which Byzantine agreement can be solved applies to these other models with partially synchronous communication 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 round. In general, we consider unrestricted Byzantine processes unless the restriction is explicitly mentioned. We also distinguish the case where processes are innumerate from the case where they are numerate. We say that a process is *innumerate* if the messages it receives in a round form a set of messages: the process cannot count the number of copies of identical messages it receives in the round. We say that a process is *numerate* if the messages it receives in a round form a multiset of messages: the process can count the number of copies of identical messages it receives in the round. (As we shall show, the numerate model is more powerful than the innumerate model against restricted Byzantine processes).

The goal of an agreement algorithm is for a set of processes proposing values to decide on exactly one of these values. We consider the classical *Byzantine agreement* problem [11,20], defined by the following three properties.

1. *Validity:* If all correct processes propose the same value v, then no value different from v can be decided by any correct process.
2. *Agreement:* No two correct processes decide different values.
3. *Termination:* Eventually, each correct process decides some value.

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

In the synchronous model, processes executing our Byzantine agreement algorithm terminate after producing an output. In the partially synchronous model, a process must continue participating in a Byzantine agreement protocol, even after it has decided. If processes did stop after deciding, it would be possible for the adversary to block all incoming and outgoing messages from one correct process until after a decision has been made by the other processes. When that one correct process does begin receiving messages, it would not be able to find out what decision value was chosen if all other processes have terminated.

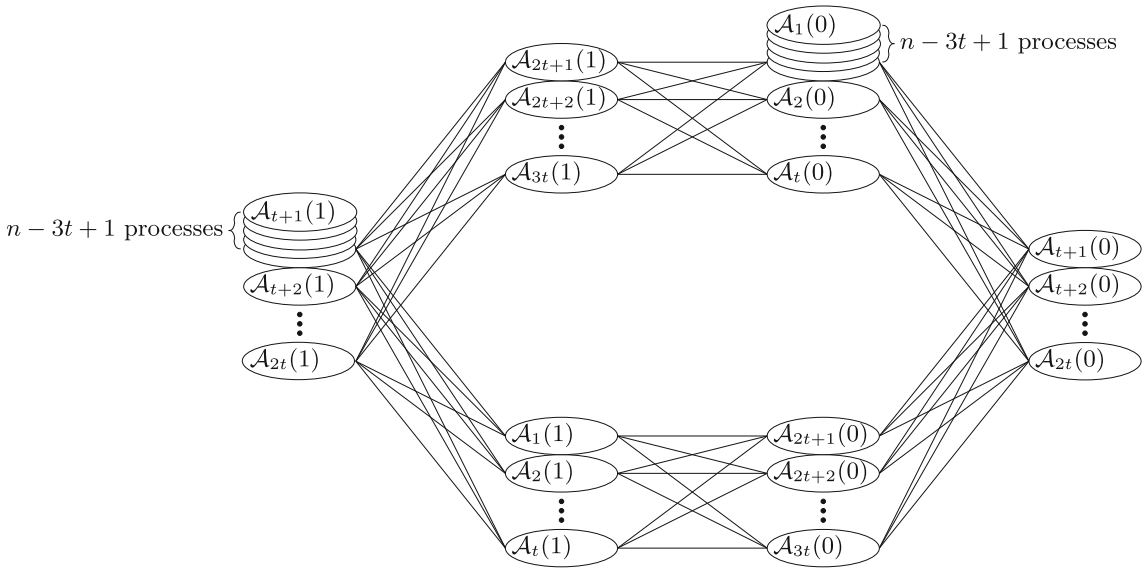

Fig. 1 The system used in the proof of Proposition 1

3 Synchronous model

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

3.1 Impossibility

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

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

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

Imagine setting up a system as shown in Fig. 1. Every process correctly executes the algorithm $\mathcal{A}_i(v)$ 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 $\mathcal{A}_i(v)$, as shown. The algorithm will *not* necessarily solve Byzantine agreement in this system, since the algorithm is designed for a system of n processes. However, we shall derive the desired contradiction by considering how processes must behave in this system.

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, so that a single Byzantine process with identifier 1 in the n-process system can send the messages sent by all $n-3t+1$ processes running $\mathcal{A}_1(0)$ in Fig. 1). 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 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 construction of the agreement algorithm is generic. We begin with any synchronous Byzantine agreement algorithm for ℓ processes with unique identifiers that terminates in a bounded number of rounds (such algorithms exist when $\ell = n > 3t$, e.g., [13,16]). We transform any such algorithm 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

```
Code for process p_i with input v
1   s = init(i, v)
2   for r = 1 to maxrounds
3       send ⟨M(s)⟩ to all processes
4       R = the set of messages received in this round
5       s = δ(s, R)
6   end for
7   output decide(s)
```

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

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 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 $\mathcal{A}$ (Fig. 2) for a system with ℓ processes $\{p_1, \ldots, p_\ell\}$ that uses at most $maxrounds$ rounds. Without loss of generality, we assume that each process terminates in $\mathcal{A}$ after running for exactly $maxrounds$ rounds, since processes that terminate early could instead send no messages in the final rounds and terminate at the required time. $\mathcal{A}$ can be specified by:

- a set of local process states,
- a function $init(i, v)$ that gives the initial state of process p_i when p_i has input value v,
- a function $M(s)$ that determines the message to send when a process is in state s,
- a transition function $\delta(s, R)$ that determines the new state to which the process moves from state s after receiving a set of messages R, and
- a decision function $decide(s)$ which is the output value for a process whose final state is s.

In our new algorithm $\mathcal{T}(\mathcal{A})$, shown in Fig. 3, two rounds simulate each round of $\mathcal{A}$. We call these two rounds a *phase*. Each phase consists of a *selecting round*, and a *running round*. In the selecting round (line 4 to 6) of a phase r, the processes within each group try to agree on a state for phase r. We shall show that if all processes in a group are correct, then, in each round, the selected state will be the same for the processes in this group. In running rounds (line 8 to 13), each process simulates one step of algorithm $\mathcal{A}$ using the state chosen in the preceding selecting round and the messages received in the running round. After $maxrounds$ complete phases, the algorithm performs one more selecting round. Then, during a final *deciding round* processes broadcast the output value corresponding to their simulated states, and each process chooses a decision value that was sent to it by processes from more than $2t$ different groups. This additional round is needed to ensure that processes that

```
Code for processes with identifier i with input v
1   s = init(i, v)
2   r = 1
3   loop
4       send ⟨s⟩ to all processes                          /* selecting round: groups agree on their state */
5       R = the set of messages received in this round
6       s = choose({x.val : x ∈ R and x.id = i})           /* choose deterministically chooses one element of this set */
7       exit when r > maxrounds
8       send ⟨M(s)⟩ to all processes                       /* running round: simulate one round of original algorithm */
9       R = the set of messages received in this round
10      for all j ∈ {1, ..., ℓ}                            /* eliminate messages from known Byzantine groups */
11          if there is more than one different message from identifier j in R then remove all of them from R
12      end for
13      s = δ(s, R)
14      r = r + 1
15  end loop
16  send ⟨decide(s)⟩ to all processes                      /* deciding round: replaces decision line of original algorithm */
17  R = set of messages received in this round
18  output a value x such that |{i : ∃m ∈ R such that m.id = i and m.val = x}| > 2t
```

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

share an identifier with a Byzantine process can decide correctly.

Let α_H be an execution of $\mathcal{T}(\mathcal{A})$. Let $G(i)$ be the group consisting of processes with identifier i. We say that the group $G(i)$ is *correct* if all processes in $G(i)$ are correct in α_H. At most t of the ℓ groups are not correct. We first observe that in the selecting round of each phase r of α_H, all processes in a correct group $G(i)$ select the same state s_i^r. This is because each process in $G(i)$ sends the same message to all processes, and therefore receives the same set of messages with identifier i during the selecting round. Every process in the group then makes the same choice for the new value of s at line 6. (For example, the *choose* function could return the minimum element of the set, if we have a total order on possible messages). Let $winner_i^r$ be a process in $G(i)$ that sent a message containing s_i^r in the selecting round of phase r.

We construct an execution α of $\mathcal{A}$ that has the following properties for every correct group $G(i)$.

1. The input to p_i in α is the input to some process of $G(i)$ in α_H.
2. Process p_i is correct in α and for $1 \leq r \leq maxrounds + 1$, p_i's state variable s has value s_i^r at the beginning of round r in α.

We construct the execution α inductively. In the first selecting round of α_H, the processes in group $G(i)$ select a state s_i^1 contained in a message sent by the process $winner_i^1 \in G(i)$ during the first selection round. Thus $s_i^1 = init(i, v)$, where v is the input to process $winner_i^1$. Let v be the input to process p_i in α. This establishes property 1 for α. Then, according to algorithm $\mathcal{A}$, p_i will be in state s_i^1 at the beginning of round 1 in α, so property 2 is satisfied for round 1.

Assume that property 2 holds for some $r \geq 1$. We construct round r of α so that property 2 holds for $r+1$. We now describe the messages sent by each process p_i in round r of α. For each correct group $G(j)$, we let p_j send the message specified by $\mathcal{A}$ to all other processes. By the hypothesis, p_j is in state s_j^r at the beginning of round r of α, so this message will be $M(s_j^r)$. For each incorrect group $G(j)$, we let p_j behave in the following Byzantine manner. For each correct group $G(i)$, p_j sends to process p_i the message that $winner_i^{r+1}$ has in set R from a process with identifier j at the end of the running round of phase r of α_H (if any). (There is at most one such message after $winner_i^{r+1}$ has executed line 11).

We show that, for all correct groups $G(i)$, $winner_i^{r+1}$ receives the same set of messages at the end of the running round of phase r of α_H as p_i receives in round r of α. (Below, we denote this common set by R_i^r). If $G(j)$ is correct, all processes in $G(j)$ send $M(s_j^r)$ to $winner_i^{r+1}$ in the running round of phase r. Since processes are innumerate, $winner_i^{r+1}$ will only receive a single copy of $M(s_j^r)$ from processes with identifier j, just as p_i does in round r of α. If $G(j)$ is not correct, then by definition, $winner_i^{r+1}$ has the same message labelled with identifier j at the end of the running round of phase r of α_H as p_i receives in round r of α (if any).

Now consider any correct group $G(i)$. In the selection round of phase $r+1$, $winner_i^{r+1}$ sends $\delta(s_i^r, R_i^r)$ and all processes in group $G(i)$ choose this as their new state s_i^{r+1}. At the end of round r in α, p_i updates its state to $\delta(s_i^r, R_i^r)$. This guarantees property 2 holds for $r+1$.

This completes the inductive construction of execution α satisfying property 1 and property 2.

Since $\mathcal{A}$ is a synchronous Byzantine agreement algorithm that tolerates t Byzantine failures, all correct processes decide some value x in α. It follows from property 2 that in α_H, for all correct groups $G(i)$, $decide(s_i^{maxrounds+1}) = x$. As $\ell > 3t$, at least $2t+1$ groups $G(i)$ are correct and all processes in these groups send x in the deciding round. Thus, each correct process in α_H decides x, even if it is in a group with a Byzantine process. Thus, the agreement property is satisfied. For validity, if all correct processes in α_H have the same input value v then all correct processes in α also have input value v, by property 1, and all correct processes in α and α_H must output v. □

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

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

4 Partially synchronous model

Here we prove that having $\ell > \frac{3t+n}{2}$ (and $n \geq 3t$) is necessary and sufficient for solving Byzantine agreement in a partially synchronous system, regardless of whether the processes are numerate or innumerate. At most $n - \ell$ identifiers belong to more than one process, so the number of identifiers that are assigned only to a single process is at least $\ell - (n - \ell) = 2\ell - n$. Thus, our condition that $\ell > \frac{3t+n}{2}$ (or, equivalently, $2\ell - n > 3t$) means that at least $3t + 1$ of the identifiers must each be assigned to a single process. We shall see in Sect. 4.2 that having this many non-homonym processes will be crucial in proving the correctness of the algorithm that we design.

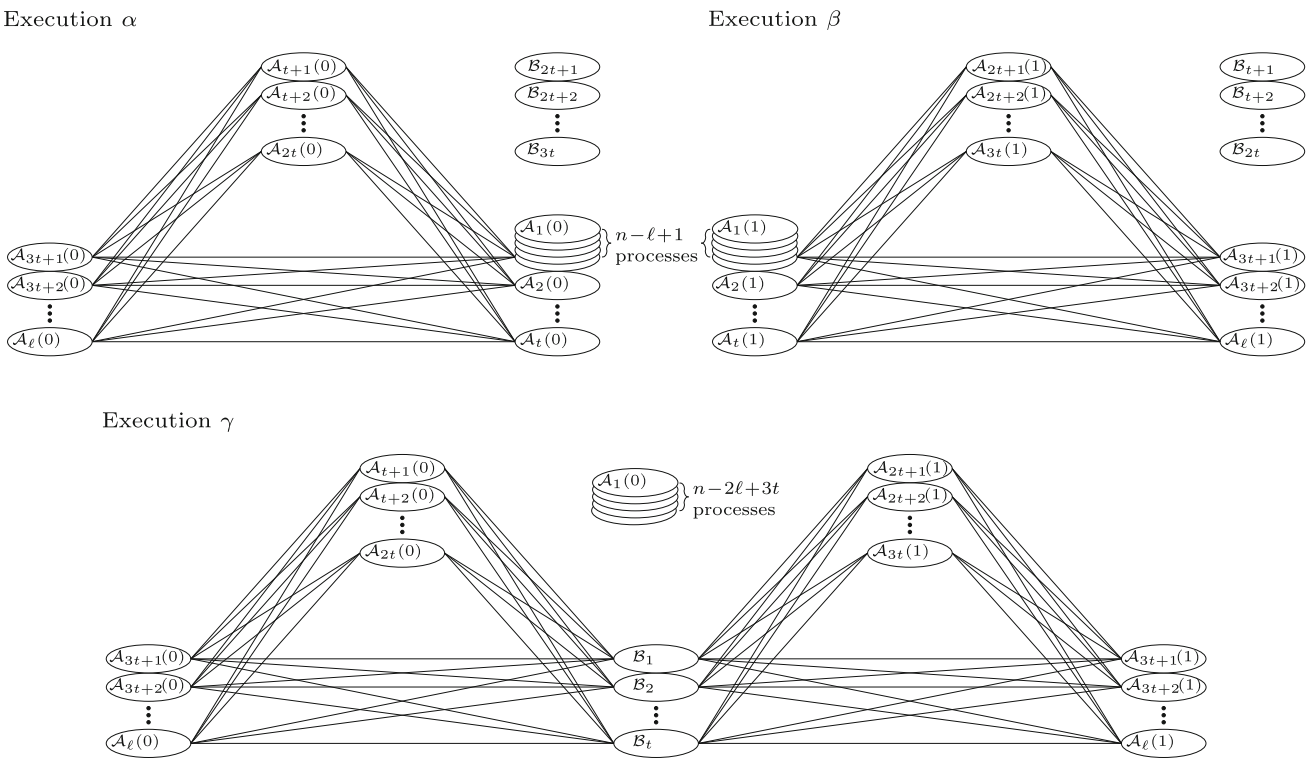

Fig. 4 The system used in the proof of Proposition 4

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 correct 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 Byzantine agreement is impossible when $\ell > 3t$ and $\ell \leq \frac{n+3t}{2}$. To derive a contradiction, assume a Byzantine agreement algorithm $\mathcal{A}$ does exist for such a system. In our proof, we construct three executions of this algorithm, α, β and γ.

In α, process identifiers are assigned as shown in the upper left portion of Fig. 4. In this diagram, a process labelled $\mathcal{A}_i(v)$ has identifier i and input v 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 Fig. 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 and input values as shown in the bottom half of Fig. 4. The identifiers $3t+1, 3t+2, \ldots, \ell$ are each assigned to two correct processes, and the identifier 1 is assigned to $n-2\ell+3t$ correct processes and to one Byzantine process. (Here, we use the assumption that $\ell \leq \frac{n+3t}{2}$, so that $n-2\ell+3t \geq 0$). The t Byzantine processes send no messages to the correct processes with identifier 1. The t Byzantine processes $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_t$ send to each other 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 running $\mathcal{A}_{t+1}(0), \ldots, \mathcal{A}_{2t}(0)$ and $\mathcal{A}_{3t+1}(0), \ldots, \mathcal{A}_\ell(0)$ cannot distinguish γ from α for the first r rounds, so they must decide 0 by round r. Similarly, 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. □

4.2 Algorithm

In this section, we describe an algorithm that solves Byzantine agreement in the partially synchronous model when $\ell > \frac{n+3t}{2}$ to prove the following proposition.

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

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

We think of an execution as being divided into superrounds, where each superround consists of two consecutive rounds. In the partially synchronous model, only a finite number of messages are not delivered. 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 [23]. 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 correct process with identifier i performs Broadcast(m) in superround $r \geq T$, then every correct process performs Accept(m, i) during superround r.
2. *Unforgeability*: If all processes with identifier i are correct and none of them perform Broadcast(m), then no correct process performs Accept(m, i).
3. *Relay*: If some correct process performs Accept(m, i) during superround r, then every correct process performs Accept(m, i) by superround $\max(r + 1, T)$.

In the following descriptions of algorithms, when we say processes with k different identifiers have performed some action or that messages with k different identifiers have been received, we mean *at least* k different identifiers.

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 The implementation is a straightforward generalization of the ones given in [10,23] 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 $\langle \text{echo } m, i \rangle$ 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 $\langle \text{echo } m, i \rangle$ from $\ell - 2t$ distinct identifiers sends a message $\langle \text{echo } m, i \rangle$. If, at any time, a process has received the message $\langle \text{echo } m, i \rangle$ from $\ell - t$ distinct identifiers, the process performs Accept(m, i).

Correctness: If a correct process with identifier i performs Broadcast(m) in some superround $r \geq T$, then all correct processes send $\langle \text{echo } m, i \rangle$ 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 $\langle \text{echo } m, i \rangle$ is if it has previously received $\langle \text{echo } m, i \rangle$ 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 $\langle \text{echo } m, i \rangle$ message. So, no process can receive $\langle \text{echo } m, i \rangle$ 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. Then, p has received $\langle \text{echo } m, i \rangle$ 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 $\langle \text{echo } m, i \rangle$ in every round after superround r. Thus, in superround $\max(r+1, T)$ every correct process sends $\langle \text{echo } m, i \rangle$ and all of these messages are delivered, so every correct process performs Accept(m, i). □

We now describe the Byzantine agreement protocol, shown in Fig. 5, that we use to prove Proposition 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. Recall that each superround consists of two rounds. (In fact, in the fourth superround of

Code for process with identifier i and input v_{in}

```
1  ph = 0                                                                          /* phase number */
2  lock = ⊥                                                                        /* locked value (initially none) */
3  lockphase = 0                                                                   /* phase when lock was last set to a non-⊥ value */
4  proper = {v_in}                                                                 /* values known to satisfy validity */
5  loop
6      /* beginning of superround 1 of phase */
7      if lock = ⊥ then BROADCAST(⟨propose proper, ph⟩)
8      else if lock ∈ proper then BROADCAST(⟨propose {lock}, ph⟩)
9      /* beginning of superround 2 of phase */
10     send ⟨proper⟩ to all other processes                                        /* round 1 of superround 2 */
11     if messages received from t + 1 different identifiers in this round contain some value v then add v to proper
12     if there is any set of 2t + 1 messages received in this round from different identifiers such that there is no value
13         that appears in at least t + 1 of them then add all possible input values to proper
14     if i = (ph mod ℓ) + 1 and there is some value v such that the process has performed ACCEPT(⟨propose V_j, ph⟩, j)
15             for ℓ − t different identifiers j, where V_j is a set that contains v
16         then choose one such v and send ⟨lock v, ph⟩ to all processes            /* round 2 of superround 2 */
17     /* beginning of superround 3 of phase */
18     if there is some value v for which the process received ⟨lock v, ph⟩ from identifier (ph mod ℓ) + 1 and
19             has performed ACCEPT(⟨propose V_j, ph⟩, j) for ℓ − t different identifiers j where V_j is a set that contains v
20         then choose one such v and perform BROADCAST(⟨vote v, ph⟩)               /* superround 3 */
21     /* beginning of superround 4 of phase */
22     if for some v, the process has performed ACCEPT(⟨vote v, ph⟩, j) for ℓ − t different identifiers j
23         then   choose one such v and set lock = v and lockphase = ph
24                send ⟨ack v, ph⟩ to all processes                                /* round 1 of superround 4 */
25     if for some v the process has received ⟨ack v, ph⟩ from ℓ − t different identifiers in this round and
26             has performed ACCEPT(⟨propose V_j, ph⟩, j) for ℓ − t different identifiers j where V_j is a set that contains v
27         then decide v (but continue running the algorithm)
28     /* after end of superround 4 of phase */
29     if for some v ≠ lock and ph′ > lockphase, the process has performed ACCEPT(⟨vote v, ph′⟩, j) for ℓ − t different identifiers j
30         then lock = ⊥
31     ph = ph + 1
32 end loop
```

Fig. 5 Byzantine agreement algorithm for the partially synchronous model

each phase, the agreement algorithm only uses the first round, but we still consider allocate two rounds to the superround for consistency with the authenticated broadcast mechanism, which is running in the background). Processes assigned the identifier $(ph \bmod \ell) + 1$ are called the *leaders* of phase ph. The algorithm uses the authenticated broadcast primitive of Proposition 6, which is possible because $\ell > \frac{n+3t}{2} > 3t$.

We first describe how each process keeps track of a set of *proper* values, which are values that it knows can be output without violating validity. Initially, only the process's own input value is in this set. This set is updated in the second superround of each phase. Each process sends its *proper* set to all others. If a process receives *proper* sets containing v in messages from $t + 1$ different identifiers, it adds v to its own *proper* set at line 11: at least one correct process must already have v in its *proper* set, so it is safe to add it. Also, if a process has received *proper* sets from $2t + 1$ different identifiers and no value appears in $t + 1$ of them, the process adds all possible input values to its own *proper* set at line 13. 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.

Now, we describe the main sequence of events that takes place during each phase. In superround 1 of each phase, each process performs a BROADCAST of a proposal containing the set of values it would be willing to decide, if any (line 7 and 8). The process never proposes a value that is not in its *proper* set. Moreover, if the process has already locked a value, as described below, it does not propose any other value. In superround 2 of the phase, each phase leader chooses a value that appears in proposals that 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 16). Then, in superround 3 of the phase, all processes vote on which lock message to support, using a BROADCAST (line 20). In superround 4 of the phase, if a process ACCEPTS votes for a particular value v that comes from $\ell - t$ different identifiers, the process locks that value v (line 23) and sends an $\langle$ack $v\rangle$ message (line 24). Then, if a process receives $\ell - t$ ack messages for a value (and that value has been proposed by processes with $\ell - t$ different identifiers in this phase), then the process can decide that value (line 27). Finally, a process releases an old lock (line 30) if it has accepted enough votes for a later lock request by the end of the fourth superround of the phase.

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

We begin by proving the property of quorums used by the algorithm.

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

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

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

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

Proof Suppose a correct process p sends $\langle \text{ack } v, ph\rangle$ and a correct process p' sends $\langle \text{ack } v', ph\rangle$. According to line 22, there is a set A of $\ell - t$ identifiers j for which p performs Accept($\langle \text{vote } v, ph\rangle, j$). Similarly, there is a set B of $\ell - t$ identifiers j for which p' performs Accept($\langle \text{vote } v', ph\rangle, j$). By Lemma 7, $A \cap B$ contains an identifier j that belongs to only one correct process and no Byzantine processes. By unforgeability, the correct process with identifier j must have performed Broadcast($\langle \text{vote } v, ph\rangle$) and Broadcast($\langle \text{vote } v', ph\rangle$). Thus, $v = v'$. □

Lemma 9 *If two correct processes decide during the same phase, then they decide the same value.*

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

The remainder of the proof of correctness of the algorithm is similar to the proof for the original algorithm of [10]. The following lemma is used to ensure agreement between values decided on line 27 in *different* phases.

Lemma 10 *Suppose there is a value v and a phase ph such that processes with $\ell - t$ different identifiers send an $\langle \text{ack } v, ph\rangle$ message in phase ph. Then, at all times after phase ph, each correct process that sent $\langle \text{ack } v, ph\rangle$ has $lock = v$.*

Proof To derive a contradiction, suppose the claim is false. Let A be the set of $\ell - t$ identifiers of the processes that send an $\langle \text{ack } v, ph\rangle$ message in phase ph. Consider the first time the claim is violated: some correct process p that sent an $\langle \text{ack } v, ph\rangle$ message changes its $lock$ to a value different from v at line 23 or 30. In either case, there is some $v' \neq v$ and $ph' > ph$ such that p 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 q be the correct process with this identifier j. Since q's identifier is in A, q sent $\langle \text{ack } v, ph\rangle$ in phase ph. Since q's identifier is in B, it follows from unforgeability that q performed a Broadcast($\langle \text{propose } V_j, ph'\rangle$) with $v' \in V_j$. According to line 7 and 8, this is possible only if q's $lock$ variable did not have the value v at the beginning of phase ph'. This contradicts our assumption that each correct process that sent an $\langle \text{ack } v, ph\rangle$ message in phase ph (including q) keeps the value v in its $lock$ variable from the time it executes line 24 of phase ph until it executes line 23 of phase ph'. □

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

Lemma 11 *At the end of any phase ph_3 that occurs after superround T, any two correct processes that have non-$\perp$ lock values have the same lock value.*

Proof Let p_1 and p_2 be any two correct processes. Suppose the values of their $lock$ variables are $v_1 \neq \perp$ and $v_2 \neq \perp$ at the end of phase ph_3. We shall prove that $v_1 = v_2$. Let ph_1

and ph_2 be the values of the *lockphase* variable of processes p_1 and p_2 at the end of phase ph_3. Then, $ph_1 \leq ph_3$ 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 $ph_1 < ph_2$.

Before process p_2 set its *lock* variable to v_2 in phase ph_2, it performed ACCEPT($\langle$vote v_2, $ph_2\rangle$, j) for $\ell - t$ different identifiers j by the end of the third superround of phase ph_2. By the relay property of the authenticated broadcasts, p_1 will accept all of these messages by the end of the fourth superround of phase $ph_3 \geq ph_2$. Thus, if p_1's *lock* is not v_2 at line 29 of phase ph_3, p_1 would set its *lock* value to $\perp$ at line 30 of phase ph_3. However, we assumed that p_1's *lock* value at the end of phase ph_3 is $v_1 \neq \perp$, so 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 non-$\perp$ value will be appear in the *lock* variables of correct processes at the end of phase $ph - 1$. We consider two cases.

Case 1: the *lock* variable of some correct process q is non-$\perp$ at the end of phase $ph - 1$. Let v and ph_v be the values of q's *lock* and *lockphase* variables at the end of phase $ph - 1$. Then, ph_v is smaller than ph and q performed ACCEPT($\langle$vote $v, ph_v\rangle$, j) for $\ell - t > t$ different identifiers j, including some identifier that does not belong to any Byzantine process. Thus, some correct process s performed BROADCAST($\langle$vote $v, ph_v\rangle$). So, s performed ACCEPT($\langle$propose $V_j, ph_v\rangle$, j) for $\ell - t \geq 2t + 1$ different identifiers j with $v \in V_j$. At least $t + 1$ of those identifiers do not belong to any Byzantine process. Therefore, correct processes with $t + 1$ 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 (and hence at the beginning of phase $ph - 1$, since *proper* sets can only grow). In superround 2 of phase $ph - 1$ these processes will send *proper* sets containing v. All of these messages will be delivered, since $ph - 1$ is after superround T. Thus, by the end of phase $ph - 1$, v will be in the *proper* set of every correct process. It follows from Lemma 11 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 a lock message during superround 2 of phase ph.

Case 2: the *lock* variable of every correct process is $\perp$ at the end of phase $ph - 1$. If there are $t + 1$ correct processes with the same input value, line 11 of phase $ph - 1$ will ensure that value is in the *proper* set of all correct processes. Otherwise, consider a set of $2t + 1$ messages received by some correct process q in phase $ph - 1$ from correct processes with $2t + 1$ different identifiers: no value will appear in $t + 1$ of them, so q will add all possible input values to its *proper* set at line 13. Either way, there exists a value that 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 a lock message during superround 2 of phase ph. □

We are now ready to prove Proposition 5 by showing the algorithm in Fig. 5 solves Byzantine agreement.

Proof We prove each of the three correctness properties of the algorithm in Fig. 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 $V, ph\rangle$) message only if $V = \{v_0\}$. It follows from unforgeability that a correct process can perform an ACCEPT($\langle$propose $V, ph\rangle$, j) only if $V = \{v_0\}$ or a Byzantine process has identifier j. Thus, according to the test on line 26, no correct process can decide a value different from v_0 since $\ell - t > t$.

Agreement: If no correct processes ever decide, agreement is trivially satisfied, so we consider executions where at least one correct process decides.

Let phase ph_1 be 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_1 be a correct process that decides v_1 during phase ph_1. Then p_1 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_2 decides a value v_2 in some phase $ph_2 > ph_1$. We shall prove that $v_2 = v_1$. Before deciding v_2, process p_2 must have performed ACCEPT($\langle$propose $V_k, ph_2\rangle, k$) with $v_2 \in V_k$ for $\ell - t$ different identifiers k to satisfy the test on line 26. 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, the *lock* variable of q equals v_1 at the beginning of phase ph_2. Thus, the process with identifier k does not perform BROADCAST($\langle$propose $V_k, ph_2\rangle$) unless $V_k = \{v_1\}$. By unforgeability, no correct process can perform ACCEPT($\langle$propose $V_k, ph_2\rangle, k$) unless $V_k = \{v_1\}$. Since $k \in B$, process p did perform ACCEPT($\langle$propose $V_k, ph_2\rangle, k$) and $v_2 \in V_k$. Thus, $v_2 = v_1$. This completes the proof of the agreement property.

Termination: There are at least $2t + 1$ correct processes that do not share their identifier with any other process. Let p be one such process and let ph be a phase such that $(ph \mod \ell) + 1$ 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 $(ph \mod \ell) + 1$ in this phase. According to the test in line 14, p must have performed ACCEPT($\langle$propose V_j, $ph\rangle$, j) for $\ell - t > t$ 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 ACCEPTactions 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 4 of phase ph. Each correct process receives all of these messages and decides v. □

Combining Proposition 4 and 5, and the classical result that Byzantine agreement is impossible when $n \leq 3t$ even if $\ell = n$, yields the following theorem (for numerate or innumerate processes).

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

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* $\ell > t$ *and* $n > 3t$.

Theorem 15 *Partially synchronous Byzantine agreement is solvable with numerate processes against restricted Byzantine processes if and only if* $\ell > t$ *and* $n > 3t$.

Both of these theorems follow from Proposition 16 and 18, below. The impossibility result of Proposition 16 is proved using a valency argument, the proof technique introduced by Fischer, Lynch and Paterson [12].

Proposition 16 *Synchronous Byzantine agreement is unsolvable with numerate processes against restricted Byzantine processes if* $\ell \leq t$ *or* $n \leq 3t$.

Proof It is a classical result that synchronous consensus is impossible if $n \leq 3t$, even if each process has a unique identifier [20]. We show that it is impossible when $\ell \leq t$. 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 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 v-*valent* if, starting from C, the only possible decision value that correct processes can have is v. C is *univalent* if it is v-valent for some v. 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. 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 $\mathcal{A}$).

We first show that there is a multivalent initial configuration. 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.

Next, we show that every multivalent configuration of $\mathcal{A}$ has a multivalent successor configuration. Suppose this claim is false to derive a contradiction. Then, there exists a multivalent configuration C_θ of $\mathcal{A}$ such that every successor configuration of C_θ is univalent. Thus, 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 that is reached if, in round $\theta+1$, the Byzantine processes send the same messages to the first j correct processes as they do to make the system reach $C'_{\theta+1}$, and send the same messages to the rest of the processes as they to make the system reach $C_{\theta+1}$. 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.

Thus, 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 contradiction completes the proof of Proposition 16. □

Next, 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 $\ell > t$ and $n > 3t$.*

The algorithm used to prove this proposition is similar to the one presented in Sect. 4.2. In Sect. 5.1.1, we first introduce a more powerful version of authenticated broadcasts, which can be implemented in systems with numerate processes against restricted Byzantine processes. Then, we use the broadcasts to give the Byzantine Agreement algorithm in Sect. 5.1.2.

5.1.1 Authenticated broadcasts with multiplicities

In this more powerful version of authenticated broadcasts, Accept actions have two extra parameters indicating the superround in which the message was broadcast and an estimate of the number of correct processes that performed the broadcast in that round. More precisely, this estimate is greater than or equal to the number of correct processes that broadcasted the message and does not exceed the number of correct broadcasters by more than the actual number of Byzantine processes in the execution. Furthermore, all correct processes eventually agree on the multiplicity of each message.

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(m), where m is a message, and Accept(m, i, r, α) where α is a positive integer. In the Accept action, α is an estimate of the number of processes with identifier i that broadcasted m in superround r.

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 and are not known by the processes). The following properties define the correctness of the authenticated broadcasts. We use $*$ as a wildcard, indicating that one field of an action or message can take on any value.

1. *Correctness:* If $\alpha > 0$ correct processes with identifier i perform Broadcast(m) in some superround $r \geq T$ then every correct process performs Accept(m, i, r, α') with $\alpha' \geq \alpha$ during superround r.
2. *Relay:* If a correct process performs Accept(m, i, r, α) in superround $r' \geq r$ then every correct process performs Accept(m, i, r, α') with $\alpha' \geq \alpha$ in superround $\max(r', T) + 1$.
3. *Unforgeability:* If the number of correct processes with identifier i that perform Broadcast(m) in superround r is α, and some correct process performs Accept(m, i, r, α') then $\alpha' \leq \alpha + f_i$.
4. *Unicity:* for each m, i and r, each correct process performs at most one Accept$(m, i, r, *)$ action per superround.

We give an implementation of this version of authenticated broadcast in Fig. 6. In the algorithm, we call a message sent in round R *valid* if

- it contains at most one tuple whose first element is init, if R is even (and none if R is odd),
- for each m, h and r, it contains at most one tuple (echo, $m, h, r, *$), and
- in each tuple (echo, m, h, r, α), $r \leq \lfloor \frac{R-1}{2} \rfloor$ and $\alpha > 0$.

All messages sent by correct processes are valid. Before proving that the algorithm has the required properties, we give a brief overview of the intuition behind the argument.

To initiate a Broadcast(m) in superround r, a process with identifier i sends an (init, m) message in round $2r$ (line 4). Each process maintains an estimate $a[m, h, r]$ of how many processes with identifier h performed broadcasts of message m in superround r. The algorithm maintains an invariant that the estimates exceed the true number by at most f_h, in order to guarantee unforgeability. In round $2r$, this estimate is simply the number of copies of $(init, m)$ messages the process received from processes with identifier h (line 12).

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

```
1  a[m, h, r] = 0 for all h, m and r
2  for R = 0 to ∞
3      M = ∅
4      if R is even and the process wishes to BROADCAST(m) in this superround then M = M ∪ {(init, m)}
5      for all h ∈ {1, 2, ..., ℓ}, m ∈ possible messages, and k ∈ {0, ..., ⌊(R−1)/2⌋}
6          if a[m, h, k] ≠ 0 then M = M ∪ {(echo, m, h, k, a[m, h, k])}
7      end for
8      send ⟨M⟩ to all processes
9      Let V be the multiset of valid messages received in the round
10     for all h ∈ {1, 2, ..., ℓ} and m ∈ possible messages
11         if R is even then
12             a[m, h, R/2] = number of occurrences of (init, m) in messages of V received from processes with identifier h
13         for all k ∈ {1, ..., ⌊(R−1)/2⌋}
14             Let W be the multiset of tuples (echo, m, h, k, ∗) that occur in V
15             if |W| ≥ n − 2t then
16                 Let α1 be the maximum α such that W has at least n − 2t occurrences of (echo, m, h, k, α′) with α′ ≥ α
17                 a[m, h, k] = max(a[m, h, k], α1)
18             if R is odd and |W| ≥ n − t then
19                 Let α2 be the maximum α such that W has at least n − t occurrences of (echo, m, h, k, α′) with α′ ≥ α
20                 ACCEPT(m, h, k, α2)
21         end for
22     end for
23 end for
```

Fig. 6 Authenticated broadcast primitive for numerate processes and restricted Byzantine processes

In each round, each process sends the non-zero estimates in its array a to the other processes (see line 6). Then, each process updates its estimates based on these messages at line 17. The new estimate is the maximum multiplicity α_1 that is smaller than or equal to the estimates that were received from $n - 2t$ processes in this round. Since $n - 2t > t$, one of these $n - 2t$ estimates comes from a correct process, and therefore exceeds the true number by at most f_h, so updating the estimate in this way preserves the invariant mentioned above.

Finally, processes perform ACCEPT actions at line 20. By choosing the maximum multiplicity α_2 that is less than or equal to $n - t$ processes' estimates, we ensure that the multiplicity chosen for the accept action is less than or equal to some correct process's estimate. Together with the invariant mentioned above, this will be shown to guarantee unforgeability. The different thresholds ($n-2t$ and $n-t$) used in the definitions of α_1 and α_2 are used to ensure the relay property. Suppose some process performs ACCEPT(m, h, k, α_2). Then $n - t$ processes sent estimates greater than or equal to α_2, and at least $n - 2t$ of them must be correct. Those $n - 2t$ correct processes will eventually deliver messages to all correct processes, causing all correct processes to increase their estimates to at least α_2. Once that has occurred, all correct processes will perform ACCEPT(m, h, k, α) with $\alpha \geq \alpha_2$.

We now provide the formal proof that the implementation in Fig. 6 satisfies the specification of authenticated broadcast when $n > 3t$ and $\ell > t$.

Lemma 19 *If a correct process performs* ACCEPT$(*, *, *, \alpha)$ *then* $\alpha > 0$.

Proof To perform ACCEPT$(*, *, *, \alpha)$ in line 20, correct processes consider only valid messages (echo, $*, *, *, \beta$) with $\beta > 0$. □

The following lemma shows that a correct process's estimate $a[m, h, k]$ of the number of processes with identifier h that performed BROADCAST(m) in superround k is not too large. This will be useful in proving the unforgeability property.

Lemma 20 *Let α be the number of correct processes with identifier i perform* BROADCAST(m) *in superround r. If a correct process q sends (echo,m, i, r, a_q) in round $R \geq 2r+1$ then $a_q \leq \alpha + f_i$.*

Proof We prove this lemma by induction.

Base case ($R = 2r + 1$): A process with identifier i sends a message containing (init, m) in round $2r$ only if it performs a BROADCAST(m) in superround r or it is Byzantine. Thus, each correct process receives at most $\alpha + f_i$ valid messages containing (init, m) from processes with identifier i in round $2r$. Therefore, at the end of round $2r$, each correct process has $a[m, i, r,] \leq \alpha + f_i$. So, if a correct process q sends (echo, m, i, r, a_q) in round $2r + 1$ then $a_q \leq \alpha + f_i$.

Induction step: Let $R > 2r + 1$. Assume the lemma is true for round $R - 1$. Let q be a correct process. In line 17 of

round $R-1$, process q either did not change $a[m,i,r]$ or set it to α_1. If $a[m,i,r]$ did not change, then q sends the same tuple (echo, m,i,r,a_q) that it sent in the previous round, and the claim follows from the induction hypothesis. Otherwise, suppose q changed $a[m,i,r]$ to α_1 in round $R-1$. Then, q must have received at least $n-2t > t+1$ messages containing tuples of the form (echo, m,h,k,α') 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. □

Let $\Pi_R(m,i,r)$ be the set of correct processes that send a message containing (echo, $m,i,r,*$) in round R. The following lemma will be useful in proving the correctness property of our authenticated broadcasts.

Lemma 21 *Let α be the number of correct processes with identifier i that perform* Broadcast(m) *in superround $r \geq T$. In round $R \geq 2r+1$, we have:*

1. *if $\alpha > 0$ then every correct process is in $\Pi_R(m,i,r)$,*
2. *$\Pi_{R-1}(m,i,r) \subseteq \Pi_R(m,i,r)$, and*
3. *for every q in $\Pi_R(m,i,r)$, if q sends (echo, m,i,r,a_q) in round R then $a_q \geq \alpha$.*

Proof We prove this lemma by induction.

Base case ($R = 2r+1$): Each of the α correct processes with identifier i that perform Broadcast(m) in superround r sends (init, m) in round $2r$. Since $r \geq T$, each correct process receives all of these messages and sets $a[m,i,r] \geq \alpha$. If $\alpha > 0$, then each correct process sends (echo, $m,i,r,a[m,i,r]$) in round $2r+1$, so (1) and (3) are satisfied. If $\alpha = 0$, (1) is trivially satisfied and (3) follows from Lemma 19. From the algorithm, a correct process never sends (echo, $m,i,r,*$) in round $2r$, so $\Pi_{2r}(m,i,r) = \emptyset$. Thus, we have (2).

Induction step: Let $R > 2r+1$. Assume properties (1), (2) and (3) are true for round $R-1$. Property (2) for round R follows from the fact that $a[m,i,r]$ never decreases. Property (1) for round R follows from (1) and (2) in the induction hypothesis. To prove (3), consider any process q in $\Pi_R(m,i,r)$. If $\alpha = 0$, (3) follows from Lemma 19. If $\alpha > 0$, then by the induction hypothesis, q is in $\Pi_{R-1}(m,i,r)$ and q sent (echo, m,i,r,a'_q) with $a'_q \geq \alpha$ in round $R-1$. Since q's value of $a[m,i,r]$ can only increase, q sends a message (echo, m,i,r,a_q) in round R with $a_q \geq a'_q \geq \alpha$. □

Theorem 22 *The algorithm in Fig. 6 ensures the correctness, relay, unforgeability and unicity properties when $n > 3t$ and $\ell > t$.*

Proof We prove each of the four properties in turn.

Correctness: Suppose $\alpha > 0$ correct processes with identifier i perform Broadcast(m) in superround $r \geq T$. Each of these processes sends (init, m) in round $2r$. By Lemma 21, every correct process q sends (echo, m,i,r,α_q) in round $2r+1$, with $\alpha_q \geq \alpha$. All of these messages are delivered. Thus, every correct process sets α_2 to a value greater than or equal to α on line 19 and then performs Accept(m,i,r,α_2) at the end of superround r.

Relay: Assume some correct process p performs Accept(m,i,r,α) in superround $r' \geq r$. 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 (echo, m,i,r,α') 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 $a[m,i,r]$ variable can only increase, each of these $n-2t$ correct senders also sends a tuple of the form (echo, m,i,r,α') with $\alpha' \geq \alpha$ in the second round of superround $\max(r',T)$. All of these messages are delivered. Thus, for each correct process, the value of $a[m,i,r]$ is at least α after the process executes line 17 during the second round of superround $\max(r',T)$. Then, in superround $\max(r',T)+1$, each of the $n-t$ correct processes sends a tuple of the form (echo, m,i,r,α') with $\alpha' \geq \alpha$. All of these messages are delivered. Thus, each correct process performs Accept(m,i,r,α') with $\alpha' \geq \alpha$ in superround $\max(r',T)+1$.

Unforgeability: Assume that some correct process q performs Accept(m,i,r,α') in superround r'. Then it received at least $n-t$ messages containing tuples of the form (echo, m,i,r,α'') with $\alpha'' \geq \alpha'$. Because $n-t \geq t+1$, one of those messages came from a correct process. By Lemma 20, the α'' in that message is less than or equal to $\alpha + f_i$, so $\alpha' \leq \alpha + f_i$.

Unicity: This follows directly from the code. □

5.1.2 Byzantine agreement algorithm

Our Byzantine agreement algorithm uses the authenticated broadcasts described in Sect. 5.1.1. During superround r', a process p may perform Accept(m,i,r,α_i). For each identifier i, α_i is p's estimate of the number of processes with identifier i that performed Broadcast(m) in superround r. We say that the number of *witnesses* that p has for (m,r) in superround r' is the sum, over all i, of the α_i's that appear in all Accept(m,i,r,α_i) actions that p performs during superround r'. It follows from the properties of authenticated broadcast that the number of witnesses will eventually be at least as large as the actual number of correct processes that performed Broadcast(m) in superround r and exceed that number by at most t.

In Sect. 4.2, we gave a partially synchronous Byzantine agreement algorithm for the weakest model we consider: the model had innumerate processes and unrestricted Byzantine processes. The algorithm worked if $\ell > \frac{n+3t}{2}$ (and $n > 3t$). Here, we give a similar partially synchronous algorithm for

Code for process with identifier i and input v_{in}

```
1  ph = 0                                                                  /* phase number */
2  lock = ⊥                                                                /* locked value (initially none) */
3  lockphase = 0                                                           /* phase when lock was last set to a non-⊥ value */
4  proper = {v_in}                                                         /* values known to satisfy validity */
5  loop
6      /* beginning of superround 1 of phase */
7      if lock = ⊥ then V = proper else V = proper ∩ {lock}
8      for each v ∈ V do BROADCAST(propose v)                              /* superround 1 */
9      /* beginning of superround 2 of phase */
10     send ⟨proper⟩ to all other processes                                /* round 1 of superround 2 */
11     if messages received from t + 1 different processes in this round contain some value v then add v to proper
12     if there is any set of 2t + 1 messages received in this round from different processes such that there is no value
13         that appears in at least t + 1 of them then add all possible input values to proper
14     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)
15         then choose one such v and send ⟨lock v, ph⟩ to all processes    /* round 2 of superround 2 */
16     /* beginning of superround 3 of phase */
17     if there is some value v for which the process received ⟨lock v, ph⟩ from a process with identifier ph mod ℓ + 1
18             and there are at least n − t witnesses for (propose v, 4ph)
19         then deterministically choose one such value v and perform BROADCAST(vote v)
20     /* beginning of superround 4 of phase */
21     if for some v, there are at least n − t witnesses for (vote v, 4ph + 2)
22         then   choose one such v and set lock = v and lockphase = ph
23                send ⟨ack v, ph⟩ to all processes                        /* round 1 of superround 4 */
24     if for some v, there are at least n − t witnesses for (propose v, 4ph) and the process received n − t messages ⟨ack v, ph⟩
25         then decide v (but continue running the algorithm)
26     if for some v′ ≠ lock and ph′ > lockphase, there are n − t witnesses for (vote v′, 4ph′ + 2)
27         then lock = ⊥
28     ph = ph + 1
29 end loop
```

Fig. 7 Partially synchronous Byzantine agreement algorithm for numerate processes and restricted Byzantine processes

the stronger model where processes are numerate and the Byzantine processes are restricted. (See Fig. 7). This new version of the algorithm works under a weaker condition on the number of identifiers: $\ell > t$. The structure of the algorithm in this section is identical to the algorithm of Sect. 4.2. Here, we point out the important differences.

The algorithms of Sect. 4.2 and this section both use quorums. The main difference between them is how the quorums are defined. Often, a quorum is defined as a set of at least Q processes, for some threshold Q. In Sect. 4.2, the ability of unrestricted Byzantine processes to send multiple messages meant that we could not define quorums in this way; instead, a quorum was defined as a set of at least $\ell - t$ *identifiers*. The condition that $\ell > \frac{n+3t}{2}$ ensured that any two quorums shared at least one identifier belonging to a single correct process and no Byzantine processes, which was crucial to ensure agreement. Now that we are considering numerate processes and restricted Byzantine processes, it makes sense to define a quorum in a more traditional way, by saying a quorum consists of $n - t$ processes. Two such quorums are guaranteed to share at least one correct process since $n > 3t$.

As in Sect. 4.2, the *proper* variable stores a set of values that can be output without violating validity and is updated similarly, except that processes count the number of incoming messages containing a value, rather than the number of different identifiers those messages came from. The condition that $n > 3t$ ensures that a process could decide on a value in its *proper* set without violating validity.

Both algorithms use rotating leaders: processes with identifier i are leaders of phase ph if $ph \mod \ell = i$. The condition that $\ell > t$ is used in the algorithm of this section to guarantee that at least one identifier is assigned only to correct processes, which is needed to prove termination. Thus, the agreement and validity properties of our algorithm depend on the condition $n > 3t$, while termination is guaranteed by the condition $\ell > t$.

At line 19, processes deterministically choose one of the values they receive in lock requests to vote for. This means that the value chosen is uniquely determined by the set of lock requests received. (For example, if the values come from a totally ordered set, a process could choose the minimum value among the lock requests received).

We now prove the correctness of the algorithm in Fig. 7. Consider any execution of the algorithm. Let f_i be the number of Byzantine processes with identifier i and let $f = \sum_{i=1}^{\ell} f_i$

be the total number of Byzantine processes in the execution. The following sequence of lemmas are analogous to the ones proved in Sect. 4.2.

Lemma 23 *If some correct process p has $n-t$ witnesses for (m, r) in some superround $r' \geq r$, then at least $n-t-f$ correct processes performed* Broadcast(m) *in superround r.*

Proof For each identifier i, let α_i be the number of correct processes with identifier i that perform Broadcast(m) in superround r. By unforgeability, if p performs Accept (m, i, r, α_i') 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 24 *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) *in superround r and* Broadcast(m') *in superround r'.*

Proof By Lemma 23, there is a set A of at least $n-t-f$ correct processes that performed Broadcast(m) in superround r and there is a set B of at least $n-t-f$ correct processes that performed Broadcast(m') in superround 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 25 *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 $\langle$ack $v, ph\rangle$ and a correct process p' sends $\langle$ack $v', ph\rangle$. According to line 21, process p has $n-t$ witnesses for (vote $v, 4ph+2$). Similarly, p' has $n-t$ witnesses for (vote $v', 4ph+2$). By Lemma 24, there is at least one correct process that performed both Broadcast(vote v) and Broadcast(vote v') in superround 3 of phase ph, so $v = v'$. □

Lemma 26 *If two correct processes decide 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 $\langle$ack $v, ph\rangle$, 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 25, $v = v'$. □

Lemma 27 *Suppose there is a value v and a phase ph such that $n-t$ processes send 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 $lock = v$.*

Proof Let A be the set of correct processes that sent $\langle$ack $v, ph\rangle$ in phase ph. By the hypothesis $|A| \geq n-2t$. To derive a contradiction, suppose the claim is false. Consider the first time the claim is violated: some correct process p that sent an $\langle$ack $v, ph\rangle$ message changes its *lock* to a value different from v on line 22 or 27. In either case, there is some $v' \neq v$ and $ph' > ph$ such that p has $n-t$ witnesses for (vote $v', 4ph'+2$). By Lemma 23, some correct process performed Broadcast(vote v') in superround $4ph'+2$. That process had $n-t$ witnesses for (propose $v', 4ph'$) in superround 2 of phase ph'. By Lemma 23 there is a set B of $n-t-f$ correct processes that perform Broadcast(propose v') in superround $4ph'$. 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 q that sends $\langle$ack $v, ph\rangle$ in phase ph and performs Broadcast(propose v') in phase ph'. According to line 21, this is possible only if q's *lock* variable is not equal to v at the beginning of phase ph'. This contradicts our assumption that each correct process that sent an $\langle$ack $v, ph\rangle$ message in phase ph keeps the value v in its *lock* variable from the time it executes line 17 of phase ph until it executes line 22 of phase ph'. □

Lemma 28 *At the end of any phase ph_3 that occurs after T, any two correct processes that have non-$\perp$ lock values have the same lock value.*

Proof Let p_1 and p_2 be correct processes. Suppose the *lock* values of p_1 and p_2 at the end of phase ph_3 are $v_1 \neq \perp$ and $v_2 \neq \perp$. Let ph_1 and ph_2 be the *lockphase* values of p_1 and p_2 at the end of phase ph_3. 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 25. So for the rest of the proof assume, without loss of generality, that $ph_1 < ph_2$. Before process p_2 changes its *lock* to v_2 in phase ph_2, it has $n-t$ witnesses for (vote $v_2, 4ph_2+2$). By the relay property of the authenticated broadcast, p_1 will accept all of these messages by the end of phase ph_3 and set its *lock* to $\perp(v_1, ph_1)$, if $v_1 \neq v_2$. Thus, v_1 must be equal to v_2. □

Lemma 29 *Let p be a correct process. Let ph be a phase such that $ph \mod \ell + 1$ 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 28, at most one non-$\perp$ value will appear in the *lock* variables of correct processes at the end of phase $ph-1$. We consider two cases.

Case 1: the *lock* variable of some correct process q is non-$\perp$ at the end of phase $ph-1$. Let v and ph_v be the value of q's *lock* and *lockphase* variables at the end of

phase $ph-1$. Then, $ph_v < ph$ and q has $n-t$ witnesses for (vote v, $4ph_v+2$) in superround 3 of phase ph_v. Thus, some correct process performed BROADCAST(vote v) in superround $4ph_v+2$. That process must have $n-t \geq 2t+1$ witnesses for (propose v, $4ph_v$). By Lemma 23, at least $t+1$ different correct processes performed BROADCAST(propose v) in superround $4ph_v$, which means v is in the *proper* set 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(propose v) in superround $4ph$, and process p will be able to find a value that it can send in superround 2 of phase ph.

Case 2: the *lock* variable of every correct process is $\perp$ 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 be able to find a value that it can send in superround 2 of phase ph. □

We can now prove Proposition 18 by showing the algorithm in Fig. 7 solves Byzantine agreement for numerate processes against restricted Byzantine processes when $\ell > t$ and $n > 3t$.

Proof We prove each of the three properties that Byzantine agreement algorithms must satisfy.

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 BROADCAST(propose v) only if $v = v_0$. Thus, according to the test on line 17, a correct process can perform BROADCAST(vote v) only if $v = v_0$. Then, according to the test on line 21, 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.

Agreement Let phase ph_1 be the first phase during which some correct process p decides. By Lemma 26, 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 process 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 23, there is a set B of $n-t-f$ correct processes that perform BROADCAST(propose v_2) in superround $4ph_2$. Thus, there is some correct process $h \in A \cap B$.

Process h sent an $\langle \text{ack } v_1, ph_1 \rangle$ in phase ph_1. By Lemma 27, h's *lock* variable is v_1 at the beginning of phase ph_2. Thus, h performs only BROADCAST(propose v_1) in superround 1 of phase ph_2. But $h \in B$, so it performs BROADCAST(propose v_2) in superround $4ph_2$. Thus, $v_1 = v_2$

Termination As there are $\ell > t$ identifiers, there is at least one identifier, say k, such that all processes with this identifier are correct. Let ph be a phase such that $ph \mod \ell + 1 = k$ and phase $ph-1$ occurs after T. By Lemma 29, 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 14, 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 at line 19. Let v be the value chosen by all correct processes. All correct processes then perform BROADCAST(vote v) in superround $4ph+2$. Thus, every correct process has $n-t$ witnesses for (vote v, $4ph+2$) in superround 3 and according to the test on line 21, sends $\langle \text{ack } v, ph \rangle$ in round 1 of superround 4 of phase ph. Every correct process receives all these messages and has $n-t$ witnesses for (propose v, $4ph$) in superround 3, and thus decides v. □

If this Byzantine agreement algorithm is run in a synchronous environment, the termination argument guarantees that all correct processes will output a value by the end of the first phase in which all phase leaders are correct. Thus, the algorithm can always terminate after $t+1$ phases, so it satisfies the stronger termination condition that we require for synchronous algorithms.

5.2 Innumerate processes

We first consider the case of innumerate processes when there is full synchrony.

Theorem 30 *Synchronous Byzantine agreement is solvable with innumerate processes against restricted Byzantine processes if and only if* $\ell > 3t$.

Proof The synchronous algorithm given in Sect. 3.2 obviously still works if the Byzantine processes are restricted.

To derive a contradiction, assume that for some ℓ and t with $\ell \leq 3t$, there is an algorithm $\mathcal{A}$ that solves Byzantine agreement in a synchronous system H of n processes, ℓ identifiers and up to t Byzantine processes. Let $\mathcal{A}_i$ denote the code executed by the processes with identifier i.

Consider the classical synchronous system (where each process has its own identifier) S, with ℓ processes and at most t Byzantine processes. Let $\{q_1, q_2, \ldots, q_\ell\}$ be these processes. Let q_i run algorithm $\mathcal{A}_i$. We shall prove that this will solve Byzantine agreement in S. This contradicts the

classical impossibility result [10,16], since the number of processes is $\ell \leq 3t$.

Let α_S be any execution of the algorithm in S. We prove that the properties of Byzantine agreement are satisfied for α_S. Let $Input(i)$ denote the input of process q_i. If all ℓ processes are Byzantine in α_S, then the properties of Byzantine agreement are vacuously satisfied. Assume that α_S has $b < \ell$ Byzantine processes. Without loss of generality, assume the Byzantine processes are $q_1, \ldots, q_b$.

We consider an execution α_H of the algorithm in H where $(n - \ell + 1)$ processes have the identifier ℓ, and the other processes have identifiers $1, \ldots, \ell - 1$ (one process per identifier). In the execution α_H,

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 ℓ are correct and have input $Input(\ell)$.

The processes with identifier ℓ 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 ℓ 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. This completes the proof. □

Next, we show that the condition for solving Byzantine agreement is more restrictive when there is only partial synchrony.

Theorem 31 *Partially synchronous Byzantine agreement is solvable with innumerate processes against restricted Byzantine processes if and only if* $\ell > \frac{n+3t}{2}$ *and* $n > 3t$.

Proof By Proposition 5, there is an algorithm if $\ell > \frac{n+3t}{2}$ and $n > 3t$, 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 the correct process running $\mathcal{A}_{t+1}(0)$ in γ. It must send the same messages as the entire stack of processes running $\mathcal{A}_1$ send to the process running $\mathcal{A}_{t+1}(0)$ 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 the process running $\mathcal{A}_{t+1}(0)$. Since we are now considering a model where processes are innumerate, $\mathcal{B}_1$ can simply send one copy of the message to the process running $\mathcal{A}_{t+1}(0)$ 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., [4,25]). Some work considered shared memory for the "wake up" problem [15], others considered consensus [3]. The power of anonymous broadcast systems, in comparison with anonymous shared-memory systems has also been studied [2]. None of these considered process failures. Anonymous processes with crash failures have been considered more recently [5,6,9,14,19,22]. In [18], 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 $n > 3t$.

A kind of homonym is considered in group signatures [7], which define a signature scheme where a receiver can determine that the message was sent by a member of a group, but cannot determine which process within the group sent it. Thus, processes can choose to behave like homonyms to preserve some privacy. These signature schemes ensure stronger additional properties. For example, the signatures provide authentication that prevents Byzantine processes from forging relayed messages, and messages can be "opened" to reveal the actual sender using a secret key.

The model of homonyms considered here generalizes both the classical (non-anonymous) and the anonymous model. We completely characterized the solvability of Byzantine agreement in this model, precisely quantifying the impact of the adversary, with some surprising results. Many challenging questions remain open. How do homonyms affect the solvability of problems other than Byzantine agreement? For problems that remain solvable when homonyms are introduced into the model, how does the existence of homonyms affect the complexity of solving the problem?

Acknowledgments We are grateful to Christian Cachin for his useful comments on our model with homonyms. We thank the anonymous reviewers for all of their very helpful comments. An abbreviated version of this paper appeared in [8]. Eric Ruppert received funding from the Natural Sciences and Engineering Research Council of Canada. This work is supported by the ERC Starting Grant GOSSPLE number 204742 and the ANR VERSO SHAMAN.

References

1. Angluin, D: Local and global properties in networks of processors (extended abstract). In: Proceedings of the 12th ACM Symposium on Theory of Computing, pp. 82–93. ACM (1980)
2. Aspnes, J., Fich, F.E., Ruppert, E.: Relationships between broadcast and shared memory in reliable anonymous distributed systems. Distrib. Comput. **18**(3), 209–219 (2006)
3. Attiya, H., Gorbach, A., Moran, S.: Computing in totally anonymous asynchronous shared memory systems. Inf. Comput. **173**(2), 162–183 (2002)
4. Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: Proceedings of the 15th International Conference on Distributed Computing, vol. 2180 of LNCS, pp. 33–47. Springer (2001)
5. Bonnet, F., Raynal, M.: The price of anonymity: optimal consensus despite asynchrony, crash, and anonymity. ACM Trans. Auton. Adapt. Syst. **6**(4), 23:1–23:28 (2011)
6. Buhrman, H., Panconesi, A., Silvestri, R., Vitányi, P.M.B.: On the importance of having an identity or, is consensus really universal? Distrib. Comput. **18**(3), 167–176 (2006)
7. Chaum, D., van Heyst, E.: Group signatures. In: Advances in Cryptology: EUROCRYPT '91, vol. 547 of LNCS, pp. 257–265, (1991)
8. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Kermarrec, A.-M., Ruppert, E., Tran-The, H.: Byzantine agreement with homonyms. In: Proceedings of the 30th ACM Symposium on Principles of Distributed Computing, pp. 21–30 (2011)
9. Delporte-Gallet, C., Fauconnier, H., Tielmann, A.: Fault-tolerant consensus in unknown and anonymous networks. In: Proceedings of the 29th IEEE International Conference on Distributed Computing Systems, pp. 368–375. IEEE Computer Society (2009)
10. Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
11. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. Distrib. Comput. **1**(1), 26–39 (1986)
12. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
13. Garay, J.A., Yoram, M.: Fully polynomial byzantine agreement for $n>3t$ processors in $t+1$ rounds. SIAM J. Comput. **27**(1), 247–290 (1998)
14. Guerraoui, R., Ruppert, E.: Anonymous and fault-tolerant shared-memory computing. Distrib. Comput. **20**(3), 165–177 (2007)
15. Jayanti, P., Toueg, S.: Wakeup under read/write atomicity. In: van Leeuwen J., Santoro N., (eds.). Proceedings of the 4th International Workshop on Distributed Algorithms, vol. 486 of LNCS, pp. 277–288. Springer (1990)
16. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
17. Okun, M.: Agreement among unacquainted Byzantine generals. In: Proceedings of the 19th International Conference on Distributed Computing, vol. 3724 of LNCS, pp. 499–500. Springer (2005)
18. Okun, M., Barak, A.: Efficient algorithms for anonymous Byzantine agreement. Theory Comput. Syst. **42**(2), 222–238 (2008)
19. Okun, M., Barak, A., Gafni, E.: Renaming in synchronous message passing systems with Byzantine failures. Distrib. Comput. **20**(6), 403–413 (2008)
20. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
21. Rowstron, A.I.T., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware, vol. 2218 of LNCS, pp. 329–350 (2001)
22. Ruppert, E.: The anonymous consensus hierarchy and naming problems. In: Proceedings of the Principles of Distributed Systems, 11th International Conference, vol. 4878 of LNCS, pp. 386–400. Springer (2007)
23. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distrib. Comput. **2**(2), 80–94 (1987)
24. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. **11**(1), 17–32 (2003)
25. Yamashita, M., Kameda, T.: Computing on anonymous networks: part I-characterizing the solvable cases. IEEE Trans. Parallel Distrib. Syst. **7**(1), 69–89 (1996)