Uniform Consensus with Homonyms and Omission Failures*

Carole Delporte-Gallet, Hugues Fauconnier, and Hung Tran-The

LIAFA-Université Paris-Diderot, France
{cd,hf,hung}@liafa.univ-paris-diderot.fr

Abstract. In synchronous message passing models in which some processes may be homonyms, i.e. may share the same id, we consider the consensus problem. Many results have already been proved concerning Byzantine failures in models with homonyms [10], we complete here the picture with crash and omission failures.

Let n be the number of processes, t the number of processes that may be faulty ($t < n$) and l ($1 \leq l \leq n$) the number of identifiers. We prove that for crash failures and send-omission failures, uniform consensus is solvable even if $l = 1$, that is with fully anonymous processes for any number of faulty processes.

Concerning omission failures, when the processes are *numerate*, i.e. are able to count the number of copies of identical messages they received in each round, uniform consensus is solvable even for fully anonymous processes for $n > 2t$. If processes are not numerate, uniform consensus is solvable if and only if $l > 2t$.

All the proposed protocols are optimal both in the number of communication steps needed, and in the number of processes that can be faulty.

All these results show, (1) that identifiers are not useful for crash and send-omission failures or when processes are numerate, (2) for general omission or for Byzantine failures the number of different ids becomes significant.

1 Introduction

Generally distributed algorithms assume either that all processes have distinct identifiers and more rarely that they are anonymous. These two models are two extremes of the same model called homonyms in [10]: n processes use l different identifiers. Hence the case $l = 1$ corresponds to a fully anonymous model and the case $l = n$ to model in which all processes have different identifiers.

Anonymous models have a very restricted computational power and many impossibility results have been proved with anonymous models (e.g. leader election, Byzantine consensus[2,3,5,8,18,22] ...) and it is interesting to determine how identifiers are needed and useful. Beyond this theoretical interest, models with homonyms have also a practical interest. In large systems, it is not so easy

* Supported by ANR VERSO SHAMAN.

D. Frey et al. (Eds.): ICDCN 2013, LNCS 7730, pp. 161–175, 2013.

to ensure that all processes have unique and unforgeable identifiers. Moreover, in some cases, users of a system may wish to preserve their privacy and for this appear in the system not as an individual but rather as member of some group [9].

As agreement protocols are major tools for fault-tolerance (e.g. state machine approach [24]), to evaluate the power of identifiers we consider the classical consensus problem. In [10,12,13] many results have been proved concerning the Byzantine consensus problem in models with homonyms. In this paper we complete this picture considering less severe process failures.

With homonyms, l ($1 \leq l \leq n$) distinct identifiers are assigned to each process. Several processes may be homonyms and share the same identifier. When a process p receives a message from a process q with identifier i, then p knows that the message has been sent by some process with identifier i but does not know wether it was sent by q or another process having the same identifier i.

We restrict ourselves to synchronous message passing models. In the models we consider, computation proceeds in rounds. In each round each process sends messages to all other processes and then receives all messages that were sent to it during the round.

When processes with the same identifier send the same message, it can be assumed or not that the processes receiving this message know how many times this message was sent. Then, as in [10], we consider two variants of the model, in the first one processes are *innumerate*: each process receives only a *set* of messages without multiplicity, whereas in the second one processes are *numerate*: each process receives a *multiset* of messages.

Only benign process failures are considered here. More precisely we study: *process crashes* (a process stops its code and if the crash occurs in a round some messages from this process may be not received by some processes), *send omission* (a process may crash or omit to send messages to some processes), *receive omission* (a process may crash or omit to receive some messages) and *general omission* (a process may crash or commit send or receive omissions).

Results: Concerning innumerate processes we prove that uniform consensus[1] is solvable even with $l = 1$ (anonymous processes) for crash failures and send-omission failures. For general omission and innumerate processes, uniform consensus is solvable if and only if $l > 2t$. Hence there is no algorithm for fully anonymous processes, but the number of identifier needed depends only on the number of tolerated faulty processes. Moreover the solution we propose is in $t+1$ rounds for any t such that $t < n$, and is then optimal [1,16,21] concerning both the communication complexity in number of rounds and the resiliency. Hence, working with anonymous processes gets no penalty with crash or send-omission failures and in this sense identifiers are not useful.

Numerate processes are more powerful, even if $l = 1$ (anonymous processes) uniform consensus is solvable for general omission failures if and only if $n >$

[1] In the uniform version of the consensus, if a faulty process decides, the decided value has to satisfy the same properties as for correct processes.

Table 1. Necessary and sufficient conditions on the number identifiers for solving consensus in a system of n processes with at most t faulty processes

	Send-omission	General-omision	Byzantine	Restricted Byzantine
Innumerate processes	$l \geq 1$	$l > 2t$	$l > 3t$	$l > 3t$
Numerate processes	$l \geq 1$	$l \geq 1$	$l > 3t$	$n > 3t \wedge l > t$

$2t$ [23], that is with the same bound as for processes having different identifiers. Hence the ability of counting the number of messages may be a way to avoid identifiers.

Adding results from [10], Table 1 summarizes the necessary and sufficient conditions on the number of identifiers to solve the consensus. In the restricted Byzantine case, a Byzantine process is only able to send one message to each process in each round.

From this table, we may observe that (1) numerate processes enable to avoid the use of identifiers for omission failures, (2) for innumerate anonymous processes consensus is solvable only for send-omisison failures, and (3) identifiers are really needed for Byzantine failures.

2 Model and Definitions

We consider a synchronous message passing system of $n \geq 2$ processes. Each process has an identifier from the set $\mathcal{L} = \{1, \cdots l\}$. l is the number of different identifiers. We assume that $n \geq l$ and each identifier is assigned to at least one process. If $n > l$ some processes share the same identifiers and are homonyms, when $l = 1$ all processes have the same identifier and the processes are anonymous, when $l = n$ all processes have different identifiers. Processes with the same identifier execute the same code. For convenience, we sometimes refer to individual processes by names, but these names cannot be used by processes themselves in the algorithms. The total number of processes, n, is known by the processes.

Processes communicate by messages. Processes may only send messages to all processes with the same identifier. When a process with identifier id sends a message to the processes of identifier id', all processes with identifier id' receive m. When a process receives a message m it knows the identifier i of the sender of the message but it does not know which process with identifier i sent this message.

We consider a synchronous round based model for which in each round, each process sends the same set of messages to all other processes and then receives all messages that were sent to it during that round.

Process failures. We assume that communication is reliable in the sense that every message sent by correct processes is received by all correct processes in the round it was sent. We restrict ourselves to the following benign failures of processes:

- *Crash failure:* A faulty process stops its execution prematurely. After it has crashed, a process does nothing.
- *Send Omission failure:* A faulty process crashes or omits sending messages it was supposed to send to other processes.
- *General Omission failure:* A faulty process crashes or omits sending and/or receiving messages it was supposed to send to/from other processes.

A send (receive) omission failure actually models a failure of the output (input) buffer of a process. A buffer overflow is a typical example of such a failure. Note that when a process crashes in a round or commits a send omission, some processes may receive the message of this process for this round and some other not.

In the following t denotes an upper bound on the number of faulty processes.

Innumerate and numerate processes. As processes are anonymous, among the messages received in the round, a process may only distinguish between messages having different values, but it may be assumed or not that a process is able count the number of identical messages received in the round. [10]

More precisely, if more than one process sends the same message m in a round, the model ensures either that each process only knows that m has been sent or that each process knows not only that m has been sent but also how many time m has been sent in the round. In the first case, the processes are said *innumerate* whereas in the second case they are said *numerate*. When a process is innumerate, it cannot count the number of copies of identical messages it receives in the round and in this case each process p receives a *set* of messages. When a process is numerate, the messages it receives in round r is a *multiset* of messages.

The uniform consensus problem. The goal of a consensus algorithm is, for a set of processes proposing values, to decide on exactly one of these values. We consider the uniform consensus problem as defined in [21] by the following three properties:

1. *Termination*: Every correct process eventually decides.
2. *Uniform Validity*: If a (correct or not) process decides v, then v was proposed by some process.
3. *Uniform Agreement*: No two (correct or not) processes decide different values.

3 Consensus with Send-Omission Failures

In this section we prove that uniform consensus is solvable with send-omission failures for all t less than n even if processes are innumerate.

The crash-tolerant uniform consensus protocols in models in which processes have distinct identifiers described in [4,20,19] are based on a "flood set" strategy. Each process p maintains a local variable that contains its current estimates of

the decision value. Initially, the local variable is set to the input proposed by p. Then, during each round, each non-crashed process first broadcasts its current estimate, and then updates it to the smallest values among the estimates it has received. After $t+1$ rounds, as there is at least one round without any crash, all processes will have the same estimate. These algorithms do not use identifiers for processes and solve directly the uniform consensus problem when all processes are anonymous ($l = 1$) in presence of any number of crashes.

With omission failures, faulty processes may commit omissions in any round, and it is possible that there is failures in all rounds. To circumvent this difficulty, in [21,23,25], each process keeps track (explicitly or implicitly) of the set of processes it considers to be correct. A process does not accept messages from processes outside of this set. In [17], the current estimate of each process is updated to the current estimate of the leader selected in each round. All these ways use the fact that each process identifies the sender.

We present here a protocol that solves uniform consensus despite up to $t < n$ processes that commit send-omission failures even if all processes are fully anonymous. The underlying principles of this algorithm are inspired by [23,25]. Roughly speaking, the algorithm ensures that if some process changes its estimate in round r, then another process has changed its estimate in the previous round. After the first round, when a process changes its estimate to some value, this value may only come from a faulty process, if some process changes its estimate in round k, then at least $k-1$ processes are faulty.

The protocol for a process p is described in Figure 1. Each process p maintains local variables *new* and *old*: *old* is the estimate of the previous round and *new* the current estimate of the round. Initially, *new* is set to v, the initial value of p.

Note that after the first round, *new* is different from *old* if and only if the process has changed its estimate. Moreover a process changes its estimate only for a smaller value, then if $new < old$ that means that the process has changed its estimate. During each round r, each process first broadcasts its current value of variables *new* and *old* and then updates them as follows: the variable *old* is set to the value of variable *new* at round $r-1$. Variable *new* may change only if the process receives some pairs (v, o) with $v < o$. From the previous remark, a process changes its variable *new* in round r only if it sees that a process has changed its value of variable *new* in the previous round. If *new* is modified, it is updated by the min of the previous value of *new* and the min of all v received from processes having changed their estimate in the previous round.

Finally, at round $t+1$, each process decides on the maximum of values received in round $t+1$.

To prove the correctness of the algorithm, we use the following notation: we say that a process "changes its estimate to some value v_0 in round r", if at the end of this round, $new = v_0 \wedge v_0 < old$. We say that a process "keeps its estimate in round r", if at the end of this round, $new = old$.

Let p_0 be the process having the minimum input value v_{min}. Let $new_p(r)$ be the value of variable *new* of the process p at the end of round r, $old_p(r)$ be the

```
Code for process p
Variable:
1  input = {v}                                   /* v is the value proposed value */
2  new = input
3  old = input
Main code:
4  ROUND 1
5     send new to all processes
6     old = new
7     new = Min{v|p has received v in this round}
8  ROUND r from 2 to t
9     send (new, old) to all processes
10    old = new
11    let G_p[r] = {v| p has received (v, o) in this round and v < o}
12    if G_p[r] ≠ ∅ and Min{v|v ∈ G_p[r]} < new then new = Min{v|v ∈ G_p[r]}
13 ROUND t + 1
14    send new to all processes
15    decide Max{v|p has received v in this round}
```

Fig. 1. Anonymous Consensus Protocol with at most t send omission processes

value of variable *old* of process p at the end of round r, $V[r]$ be the set of values of variable *new* of all processes (not crashed at the end of round r) at the end of the round r: $V[r] = \{new_p(r)|p \text{ is any process}\}$, and v_{max} be the largest value of set $V[t]$.

We begin with two simple facts:

Fact 1. *$old_p(r+1) = new_p(r)$ for every round $1 \leq r < t$ and every process p not crashed at the end of round $r+1$.*

Fact 2. *$new_p(r) \leq old_p(r)$ for every round $1 \leq r \leq t$ and every process p not crashed at the end of round r.*

Lemma 1. *$V[r+1] \subseteq V[r]$ for every round $1 \leq r < t$.*

Proof. Consider variable $new_p(r+1)$ for some process p not crashed at the end of round $r+1$. There are two cases:

- p changes its estimate in round $r+1$ to some value v_0 at Line 12. Then $new_p(r+1) = v_0$ where v_0 is the value of variable *new* of some process q that sent the pair (v_0, old) to p with $v_0 = new_q(r)$.
- p keeps its estimate in this round. Hence, $new_p(r+1) = old_p(r+1) = new_p(r)$.

Thus, $V[r+1] \subseteq V[r]$ for every round $1 \leq r < t$.

We directly get:

Lemma 2. *If p_0 is correct then $new_p(r) = v_{min}$ for every round $1 \leq r \leq t$ and every process p not crashed at the end of round r.*

Lemma 3. *If a correct process has input value v_0 or changes its estimate to some value v_0 in round $1 \leq r < t$ then $new_q(t) \leq v_0$ for every process q not crashed at the end of round t.*

Proof. If some correct process p has input v_0 then at the end of round 1, the variable *new* of all processes is less than or equal to v_0 (Line 7). By Lemma 1, we have $V[t] \subseteq V[1]$. Thus, $new_q(t) \leq v_0$ for every process q.

If p changes its estimate *new* to v_0 in round $1 \leq r < t$. Then $v_0 = new_p(r) < old_p(r)$. In the round $r+1 \leq t$, the correct process p sends the pair $(v_0, old_p(r))$ to all processes. At the end of the round $r+1$, all processes receive $(v_0, old_p(r))$. We consider any process q, after Line 12, $new'_q r + 1) \leq v_0$. By Lemma 1, we have $V[t] \subseteq V[r+1]$. Thus, $new_q(t) \leq v_0$ for every process q.

Lemma 4. *If $t > 1$ and $r \geq 2$, if some p changes its estimate to v_0 in round r then, there is a set of processes $\{q_1, \cdots, q_{r-1}\}$ such that for all i, $1 \leq i \leq r-1$, q_i changes its estimate to v_0 in the round i and $new_{q_i}(r-1) \leq v_0$.*

Proof. Since p changes its estimate to v_0 in round r, we must have $new_p(r) = Min\{v | v \in G_p[r]\}$ and there is at least one process q_{r-1} that sent (new, old) to p, with $new = v_0 < old$. Hence, at the end of round $r-1$, the process q_{r-1} must have $new_{q_r}(r-1) < old_{q_r}(r-1)$. That means that q_{r-1} has changed its estimate in round $r-1$.

By induction, we have a sequence of processes $(q_1, \cdots, q_{r-1})$ such that q_i changes its estimate in round i, $1 \leq i \leq r-1$.

Furthermore, if a process changes its estimate to v_0 in some round r_0 then after this round, its value of variable *new* is less than or equal to v_0 and no process can change its estimate to v_0 twice. Thus, all processes q_i are distinct and $new_{q_i}(r-1) \leq v_0$ for all i such that $1 \leq i \leq r-1$, proving the Lemma.

Lemma 5. *Either* (a) $new_q(t) = v_{max}$ *for every correct process q, or* (b) $V[t] = \{v_{min}, v_{max}\}$, $new_s(t) = v_{min}$ *for every faulty process s and some correct process changes its estimate to v_{min} in round t.*

Proof. If $t = 1$ then consider two cases:

- if p_0 is correct and then by Lemma 2, $new_q(t) = v_{min}$ for all correct processes q.
- if p_0 is faulty. At the end of round 1, either all correct processes have the same value of *estimate* or some correct process changes its estimate to v_{min}, proving the Lemma.

faux

If $t > 1$, we consider the set of values of variable *new* of correct processes at the end of round t. Since $n > t$, this set is not empty.

Assume that (a) is not satisfied then for some correct process p $new_p(t) < v_{max}$. Let v_0 be $new_p(t)$ $(v_0 < v_{max})$. Consider the following two cases:

- v_0 is the input value of p or p changes its estimate to v_0 in a round $1 \leq r < t$ then by Lemma 3, $new_q(t) \leq v_0$ for every process q, contradicting the hypothesis that $v_0 < v_{max}$.
- p changes its estimate to v_0 in round t. By Lemma 4, we have a set of processes $\{q_1, \cdots, q_{t-1}\}$ such that q_i changes its value of variable new to v_0 in the round i, $1 \leq i \leq t-1$. Moreover, no process among them is correct because if some process q_i is correct then the value of variable new at the end of the round t of every process is less than or equal to v_0, hence $v_{max} \leq v_0$, contradicting the hypothesis that $v_0 < v_{max}$. Moreover, p_0 is faulty because if p_0 were correct then by Lemma 2, $new_p(1) = v_{min}$ and p could not change its estimate in round t, contradicting the hypothesis. Therefore, we have a set of t faulty processes $\{q_1, ..., q_{t-1}\} \cup \{p_0\}$.

  Since all processes q_i that have ever changed their estimate to v_0 in some round $r \geq 1$, v_0 cannot be the input value of one of these processes. On the other hand, by Lemma 3, if v_0 is the input value of a correct process then $new_q(t) \leq v_0$ for every process q, by definition of v_{max}, $v_{max} = v_0$, contradicting the hypothesis that $v_0 < v_{max}$. Hence, v_0 may only be the input value of faulty process p_0. That means that $new_p(t) = v_0 = v_{min}$. Moreover, all faulty processes q_i that changed their estimate to v_{min} in some round $r \geq 1$ do not change their estimate more because v_{min} is minimum value, hence $new_s(t) = v_{min}$ for every faulty process $s \in \{q_1, ..., q_{t-1}\} \cup \{p_0\}$.

Proposition 1. ***Uniform agreement****: No two processes decide different values.*

Proof. By Lemma 5, we have either

- If $new_q(t) = v_{max}$ for every correct process q then, every process p not crashed at round $t+1$ receives v_{max} in round $t+1$ and decides v_{max} at Line 15.
- $V[t] = \{v_{min}, v_{max}\}$, $new_s(t) = v_{min}$ for every faulty process s. Hence, v_{max} must come from some correct process. At the beginning of round $t+1$, this process sends v_{max} to all and every process not crashed at round $t+1$ receives the value v_{max} and decides v_{max}.

The Termination and Uniform Validity are trivially satisfied, then with the previous proposition we deduce:

Theorem 1. *Uniform consensus is solvable in $t+1$ rounds with send-omission failures of any number of processes even if all processes are anonymous.*

This algorithm is optimal concerning the number of rounds $(t+1)$ and the number of tolerated faulty processes $(t < n)$.

4 Consensus with General-Omission Failures

In this section we give an algorithm solving uniform consensus with general-omission failures if processes are numerate (even if they are anonymous). In

a second subsection we prove there is no solution for uniform consensus with general-omission failures when processes are anonymous and innumerate. More precisely we prove that at least $l > 2t$ identifiers are needed. Recall that uniform consensus is solvable with processes having unique identifiers for general-omission failures only if there is a majority of correct processes, then we always assume in this section that $n > 2t$.

4.1 Numerate Processes

In this subsection, processes are anonymous and numerate.

An algorithm in Figure 2 solves the consensus in general omission model base on the same principles as Figure 1. Round $t + 1$ has to be adapted to general-omission failures for which it is not ensured that all correct processes have the same estimate in round $t + 1$.

```
Code for process p
Variable:
1  input = {v}                                          /* v is the proposed value */
2  new = input
3  old = input
Main code:
4  ROUND 1
5     send new to all processes
6     old = new
7     new = Min{v| v in the set of messages received }
8  ROUND r from 2 to t
9     send (new, old) to all processes
10    old = new
11    let G_p[r] = {v| p has received (v, v_1) in this round and v < v_1}
12    if G_p[r] ≠ ∅ and Min{v|v ∈ G_p[r]} < new then new = Min{v|v ∈ G_p[r]}
13 ROUND t + 1
14    send (new, old) to all processes
15       if for some v received n − t pairs (v, ∗) in this round
16       then  decides v
17          else if received at least n − t pairs
18                   and one of them is (x, y) such that x < y
19                   then
20                      let G_p = {v| p has received (v, ∗) in this round }
21                      if ∃(x, y) such that x < y and (x, y) received in this round
                              and x = Min{v|v ∈ G_p}
22                      then decide x
```

Fig. 2. Anonymous Consensus Protocol with at most t general-omission processes

We now present the steps of the proof that the protocol of Figure 2 satisfies the specification of uniform consensus. We use the same notations $p_0, v_{min}, V, new_p(r)$ and $old_p(r)$ as in the proof of Theorem 1. Let $C[r]$ be the set of values of variable *new* of all correct processes at the end of the round r: $C[r] = \{new_p(r) | p$ is a correct process$\}$, and c_{max} be the largest value of set $C[t]$.

Although here processes may commit receive omission the following lemmata may be proved in a very similar way as in the proof of Theorem 1.

Lemma 6. *$V[r+1] \subseteq V[r]$ for every round $1 \leq r < t$.*

Lemma 7. *If p_0 is correct then $new_p(r) = v_{min}$ for every correct process p and every round $1 \leq r \leq t$.*

Lemma 8. *If a correct process changes its estimate to some value v_0 in round $1 \leq r < t$ or has input v_0 then $new_q(t) \leq v_0$ for every correct process q.*

Lemma 9. *If $t > 1$ and $r \geq 2$, if some process p changes its estimate to v_0 in round r then, there is a set of processes $\{q_1, ..., q_{r-1}\}$ such that for all i, $1 \leq i \leq r-1$, q_i changes its estimate to v_0 in the round i.*

Lemma 10. *Either* (a) *$new_q(t) = c_{max}$ for every correct process q, or* (b) *$V[t] = \{v_{min}, c_{max}\}$, $new_s(t) = v_{min}$ for every faulty process s, some correct process changes its estimate to v_{min} in round t, and t processes are faulty.*

Proof. If $t = 1$ then consider two cases:

- if p_0 is correct then by Lemma 7, $new_q(1) = v_{min}$ for all correct processes.
- if p_0 is faulty. At the end of round 1, either all correct processes have the same value of *estimate* or some correct process changes its estimate to v_{min}, proving the Lemma.

If $t > 1$, we consider the set of values of variable *new* of correct processes at the end of round t. Since $n > 2t$, this set is not empty.

Assume that (a) is not satisfied then for some correct process p $new_p(t) < v_{max}$. Let v_0 be $new_p(t)$ ($v_0 < c_{max}$). Consider the following two cases:

- v_0 is the input value of p or p changes its estimate to v_0 in a round $1 \leq r < t$ then by Lemma 8, $new_q(t) \leq v_0$ for every correct process q. By definition of c_{max}, we must have $c_{max} = v_0$, contradicting the hypothesis that $v_0 < c_{max}$.
- p changes its estimate to v_0 in round t. By Lemma 9, we have a set of processes $\{q_1, \cdots, q_{t-1}\}$ such that q_i changes its value of variable *new* to v_0 in the round i, $1 \leq i \leq t-1$. Moreover, no process among them is correct because if any process q_i is correct then the value of variable *new* at the end of the round t of every process is less than or equal to v_0, hence $c_{max} \leq v_0$, contradicting the hypothesis that $v_0 < c_{max}$. On the other hand, p_0 is faulty because if p_0 were correct then by Lemma 7, $new_p(1) = v_{min}$ and p could not change its estimate in round t, contradicting the hypothesis. Therefore, we have a set of t faulty processes $\{q_1, ..., q_{t-1}\} \cup \{p_0\}$.

As all q_i changed their estimate to v_0 in some round $r \geq 1$, v_0 is not any input value of these processes. On the other hand, by Lemma 8, if v_0 is input value of a correct process then $new_q(t) \leq v_0$ for every process q. By definition of c_{max}, we must have $c_{max} = v_0$, contradicting the hypothesis that $v_0 < c_{max}$. Then, v_0 may only be the input value of faulty process p_0. That means that $new_p(t) = v_0 = v_{min}$. Moreover, all faulty processes q_i that have changed their estimate to v_{min} in some round $r \geq 1$ do not change their estimate after because v_{min} is minimum value. Hence $new_s(t) = v_{min}$ for every faulty process $s \in \{q_1, ..., q_{t-1}\} \cup \{p_0\}$.

Lemma 11. *Suppose that $new_q(t) = c_{max}$ for every correct process q. Thus, if a faulty process changes its estimate to v_0 in round t then $v_0 \geq c_{max}$.*

Proof. If p_0 is correct then at the end of round 1, we have $new_q(1) = v_{min}$ for every correct process q. q never changes its estimate because v_{min} is the minimum value. Thus, $c_{max} = v_{min}$. If a faulty process changes its estimate to v_0 in round t then obviously, $v_0 \geq v_{min}$.

Now, consider the case where p_0 is a faulty process. Suppose that a faulty process q changes its estimate to v_0 in round t. By Lemma 9, we have a set of processes $\{q_1, \cdots, q_{t-1}\}$ such that q_i changes its value of variable *new* to v_0 in the round i, $1 \leq i \leq t-1$. Moreover, the faulty process p_0 that never changes its estimate must be different from all processes in the set $\{q_1, \cdots, q_{t-1}\}$. If all these processes were faulty with $p0$ and q we would have $t+1$ faulty processes, contradicting the hypothesis that there are at most t faulty processes. Therefore, at least one of processes of set $\{q_1, \cdots, q_{t-1}\}$ is correct, say p. p changes its estimate to v_0 in round $1 \leq r < t$: $new_p(r) = v_0$. But we have always $new_p(t) \leq new_p(r)$ for every $1 \leq r < t$. Thus, $v_0 = new_p(r) \geq new_p(t) = c_{max}$.

Lemma 12. *If two processes decide at Line 16 then they decide the same value.*

Proof. Suppose that process p decides v_0 at Line 16 and process q decides v_1 at Line 16. Thus, p receives at least $n-t$ pairs $(v_0, *)$ and q receives at least $n-t$ pairs $(v_1, *)$ in round $t+1$. That means that at least $n-t$ processes sent $(v_0, *)$ and at least $n-t$ processes sent $(v_1, *)$. Since $(n-t)+(n-t) > n$, $v_0 = v_1$.

Theorem 2. *If processes are numerate, uniform consensus is solvable in $t+1$ rounds if $n > 2t$ even if all processes are anonymous.*

Proof. Termination: By Lemma 10, at the end of round t, either:

- the value of variable *new* of every correct process is the same, then all correct processes decide at Line 16.
- at least one correct process p changes its estimate to v_{min} and the value of variable *new* of every faulty process is v_{min}. Thus, every correct process q receives at least $n-t$ pairs and one of them is (v_{min}, y) such that $v_{min} < y$ from p. q decides v_{min} at Line 22.

Uniform Validity: if a process decides it decides some value in set $V[t]$. By Lemma 6, we have $V[t] \subseteq V[1]$. Moreover, $V[1]$ is a subset of the inputs proposed by

processes. Therefore, if a process p decides v, then v is input value of some process.

Uniform agreement: By Lemma 10, at the end of round t either:

- the value of variable *new* of every correct process is the same c_{max} then all correct processes decide this value at Line 16. Suppose that a faulty process q want to decide. If it decides at Line 16 then by Lemma 12 it must decide the same c_{max}. If it decides at Line 22 it has received at least $n - t$ pairs and one of them is (x, y) such that $x < y$ and $x = Min\{v|v \in G_q\}$. As q receives at least $n - t$ pairs, at least one pair comes from a correct process. then $x = Min\{v|v \in G_q\} \leq c_{max}$. If this pair (x, y) comes from a correct process then $x = c_{max}$. If the pair come from a faulty process. By Lemma 11, we have $x \geq c_{max}$. Thus, in all cases, we have always $x = c_{max}$. That means that q decides the same value as correct processes.
- $V[t] = \{v_{min}, c_{max}\}$, $new_s(t) = v_{min}$ for every faulty process s, some correct process p changes its estimate to v_{min} in round t, and t processes are faulty. As p is correct, all correct processes receive its value. Moreover a correct process receives at most $n - t - 1$ value different from v_{min} and cannot decide at Line 16 then it decides v_{min} at Line 22. If a faulty process q decide, by hypothesis it has $new_s(t) = v_{min}$.

4.2 Innumerate Processes

Proposition 2. *Uniform consensus is not solvable with innumerate processes if $l \leq 2t$ with general-omission failures.*

Proof. The proof is based on a classical partitioning argument. By contradiction, assume that there is a protocol that solves the uniform binary consensus problem with $l \leq 2t$.

Let a partition of the set of identifiers $\mathcal{L}$ into two sets $I = \{1, \ldots, l/2\}$ and $J = \{l/2 + 1, \ldots l\}$, such that $|I| \leq t$ and $|J| \leq t$. Consider the two following repartitions of identifiers. In repartition R, all identifiers in I are identifier of only one process, identifiers in $J - \{l\}$ are identifiers of only one process too and identifier l is the identifier of the remaining $n - l + 1$ processes. $R(I)$ and $R(J)$ denote respectively the set of processes with identifiers in I and the set of processes with identifiers in J for repartition R Repartition S is identical to R except that only one process has identifier l and the other processes having identifier l for R have now identifier 1. $S(I)$ and $S(J)$ denote respectively the set of processes with identifiers in I and the set of processes with identifiers in J for repartition S. Note that $R(I)$ and $S(J)$ contain at most t processes. Note also that as processes are innumerate if all processes with the same identifier send the same messages in R and S and have the same initial state, execution in R or S are indistinguishable for any process.

Consider the following executions:

Execution α. The repartition is R, all processes have 0 as initial value. Processes in $R(I)$ are crashed from the beginning. By validity the decision value is 0.

Execution α'. The repartition is R, the initial values are 0 for processes in $R(J)$ and 1 for processes in $R(I)$. Processes in $R(I)$ commit send and receive omission failures: processes in $R(J)$ do not receive any message from processes in $R(I)$ and processes in $R(I)$ do not receive any message from processes in $R(J)$. α' is indistinguishable from α for processes in $R(J)$ and the decision value is 0.

Execution β. The repartition is S, all processes have 1 as initial value. Processes in $S(J)$ are crashed from the beginning. By validity the decision value is 1.

Execution β'. The repartition is S, processes in $S(I)$ have 1 as initial value and processes in $S(J)$ have 0 as initial value. Processes in $S(J)$ commit send and receive omission and processes in $S(I)$ don't receive any message from $S(J)$ and processes in $S(J)$ don't receive any message from $S(I)$. β' is indistinguishable from β for processes in $S(I)$ and the decision value is 1.

Now consider any process p with identifier in I both for R and S. As processes are innumerate p receives in β' and α' exactly the same messages from identifiers in I, and receives no messages from identifiers in J, both execution are then indistinguishable for p. In β' it decides 1 then in α' it decides 1 too. But the decision value for α' is 0.

In the other hand, the consensus is solvable when $l > 2t$. The protocol is similar to the one presented in Figure 2 only Lines 13 - 22 (round $t + 1$) are replaced by:

```
15 ROUND t + 1
16   send (new, old) to all processes
17     if for some v received (v, ∗) from at least l − t identifiers in this round
18     then  decides v
19       else if received at least messages from l − t identifiers
20               and one of them is (x, y) such that x < y
21               then
22                  let G_p = {v| p has received (v, ∗) in this round }
23                  if ∃(x, y) such that x < y and (x, y) received in this round
                          and x = Min{v|v ∈ G_p}
24                  then decide x
```

Then we get:

Theorem 3. *Uniform consensus is solvable with innumerate processes with general-omission failures if and only $l > 2t$.*

5 Conclusion and Perspectives

One natural extension of this work is to consider partially synchronous models [14,15]. Some results for consensus with anonymous processes for a specific partially synchronous model are given in [11].

Concerning numerate processes in models like [15] in which the communication between all correct processes is eventually synchronous we conjecture that

consensus is solvable with a majority of correct processes. Concerning only crash failures, it may be noticed that sending regularly "alive" messages an anonymous failure detector [6,7] giving eventually the exact number of correct processes may be implemented, and then with this failure detector and a majority of correct processes consensus may be implemented.

When processes are innumerate in the Byzantine failures case, it has been proved [10] by a partitioning argument the lower bound of $l > (n + 3t)/2$ for consensus. This proof may be adapted to crash and omission failures giving a lower bound of $l > (n + 2t)/2$. This bound indicates that in partially synchronous models, to solve the consensus the homonymy of processes must be very restricted.

One more technical issue is the ability to have early stopping algorithms. We guess that with innumerate processes early-stopping algorithms are not possible or with very poor bounds.

In some way, the results of this paper shows that two mechanisms help to solve the consensus: identifiers of processes and the ability of processes to count identical messages they receive. Anonymity of processes may be balanced by the ability to count the number of identical messages received.

References

1. Aguilera, M.K., Toueg, S.: A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. Inf. Process. Lett. 71(3-4), 155–158 (1999)
2. Angluin, D.: Local and global properties in networks of processors (extended abstract). In: STOC, pp. 82–93. ACM (1980)
3. Attiya, H., Gorbach, A., Moran, S.: Computing in totally anonymous asynchronous shared memory systems. Information and Computation 173(2), 162–183 (2002)
4. Attiya, H., Welch, J.: Distributed Computing: fundamentals, simulations and advanced topics, 2nd edn. Wiley (2004)
5. Boldi, P., Vigna, S.: An Effective Characterization of Computability in Anonymous Networks. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 33–47. Springer, Heidelberg (2001)
6. Bonnet, F., Raynal, M.: Anonymous Asynchronous Systems: The Case of Failure Detectors. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 206–220. Springer, Heidelberg (2010)
7. Bonnet, F., Raynal, M.: The price of anonymity: Optimal consensus despite asynchrony, crash, and anonymity. TAAS 6(4), 23 (2011)
8. Buhrman, H., Panconesi, A., Silvestri, R., Vitányi, P.M.B.: On the importance of having an identity or, is consensus really universal? Distributed Computing 18(3), 167–176 (2006)
9. Chaum, D., van Heyst, E.: Group Signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991)
10. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Kermarrec, A.-M., Ruppert, E., Tran-The, H.: Byzantine agreement with homonyms. In: PODC, pp. 21–30. ACM (2011)
11. Delporte-Gallet, C., Fauconnier, H., Tielmann, A.: Fault-tolerant consensus in unknown and anonymous networks. In: ICDCS, pp. 368–375. IEEE Computer Society (2009)

12. Delporte-Gallet, C., Fauconnier, H., Tran-The, H.: Byzantine Agreement with Homonyms in Synchronous Systems. In: Bononi, L., Datta, A.K., Devismes, S., Misra, A. (eds.) ICDCN 2012. LNCS, vol. 7129, pp. 76–90. Springer, Heidelberg (2012)
13. Delporte-Gallet, C., Fauconnier, H., Tran-The, H.: Homonyms with Forgeable Identifiers. In: Even, G., Halldórsson, M.M. (eds.) SIROCCO 2012. LNCS, vol. 7355, pp. 171–182. Springer, Heidelberg (2012)
14. Dolev, D., Dwork, C., Stockmeyer, L.: On the minimal synchronism needed for distributed consensus. Journal of the ACM 34(1), 77–97 (1987)
15. Dwork, C., Lynch, N.A., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM 35(2), 288–323 (1988)
16. Dwork, C., Moses, Y.: Knowledge and common knowledge in a Byzantine environment: Crash failures. Information and Computation 88(2), 156–186 (1990)
17. Guerraoui, R., Kouznetsov, P., Pochon, B.: A note on set agreement with omission failures. Electr. Notes Theor. Comput. Sci. 81, 48–58 (2003)
18. Guerraoui, R., Ruppert, E.: Anonymous and fault-tolerant shared-memory computing. Distributed Computing 20(3), 165–177 (2007)
19. Kumar, G.V.: Elements of Distributed Computing. Wiley-Interscience (2002)
20. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
21. Neiger, G., Toueg, S.: Automatically increasing the fault-tolerance of distributed algorithms. Journal of Algorithms 11(3), 374–419 (1990)
22. Okun, M., Barak, A.: Efficient algorithms for anonymous Byzantine agreement. Theory of Computing Systems 42(2), 222–238 (2008)
23. Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. IEEE Trans. Software Eng. 12(3), 477–482 (1986)
24. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys 22(4), 299–319 (1990)
25. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distributed Computing 2(2), 80–94 (1987)