Problème du Consensus dans le Modèle Homonyme

Hung Tran-The

Université Paris Diderot - PARIS 7
UFR de Mathématiques

THÈSE

pour obtenir le grade de

DOCTEUR de l'**Université Paris Diderot - Paris 7**

Spécialité : **Informatique**

présentée par

Hung TRAN-THE

PROBLÈME DU CONSENSUS DANS LE MODÈLE HOMONYMES

Directeurs de Thèse **Carole DELPORTE-GALLET** et **Hugues FAUCONNIER**

Thèse soutenue publiquement le **6 Juin 2013**, devant le jury composé de :

M.	Joffroy	BEAUQUIER	Examinateur
Mme.	Carole	DELPORTE-GALLET	Directrice de thèse
M.	Hugues	FAUCONNIER	Directeur de thèse
M.	Pierre	FRAIGNIAUD	Examinateur
M.	Matthieu	LATAPY	Examinateur
M.	Michel	RAYNAL	Rapporteur
M.	Franck	PETIT	Rapporteur
M.	Sam	TOUEG	Examinateur

Remerciements

Comme une évidence pour débuter ces remerciements, je voudrais remercier chaudement Carole Delporte et Hugues Fauconnier, directeurs de ma thèse. Je vous remercie beaucoup pour votre encadrement et votre soutien quotidien durant ces trois années. Vous avez consacré beaucoup de temps à m'écouter, me lire, et me corriger très soigneusement. Vous avez su me faire profiter de votre expérience scientifique avec bienveillance. Je suis fier d'avoir été doctorant sous votre direction.

Je tiens tout particulièrement à remercier Franck Petit et Michel Raynal pour avoir accepté de rapporter cette thèse. Je remercie également Joffroy Beauquier, Matthieu Latapy et Sam Toueg d'avoir accepté de faire partie des examinateurs. Je souhaite remercier Pierre Fraigniaud de me faire l'honneur d'accepter la Présidence du jury. Je les remercie chaleureusement de l'attention qu'ils ont bien voulu porter à mes travaux.

J'ai beaucoup apprécié l'accueil chaleureux des membres du LIAFA. Je remercie en particulier Noelle Delgado qui m'a beaucoup aidé. Un grand merci aux responsables des cours IF1, POO et Logique, Matthieu Picantin, Hugues Fauconnier et Peter Habermehl avec qui j'ai passé une bonne année d'enseignement comme Ater à l'université Paris 7.

Je salue les thésards et les stagiaires et en particulier les occupants du bureau 4057. Je vous remercie d'avoir écouté mon exposé et de m'avoir donné des corrections et des conseils.

Enfin, je tiens à remercier ma mère, mon père, ma famille, qui a toujours su m'apporter un soutien sans faille.

XIN CAM ON TAT CA CAC BAN!!!

MERCI A VOUS TOUS!!!

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 $1 \leq l \leq n$. We call this model *homonymous* model. To determine the power of homonymous model as well as the importance of identifiers in distributed computing, this thesis studies the consensus problem, one of the most famous distributed computing problem.

We give necessary and sufficient conditions on the number of identifiers for solving consensus in a distributed system with t faulty processes in the synchronous case. We show that in crash, send omission and general omission failures model, the uniform consensus is solvable even if processes are anonymous. Thus, identifiers are not useful in that case. However identifiers become important in Byzantine failures model: $3t + 1$ identifiers is necessary and sufficient for Byzantine agreement.

Surprisingly the number of identifiers must be greater than $\frac{n+3t}{2}$ in presence of three facets of uncertainty: partial synchrony, Byzantine failures and homonyms. This demonstrates two differences from the classical model (which has $l = n$): 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.

We show two ways to notably reduce the number of identifiers for Byzantine agreement. Firstly, removing the ability for a Byzantine process to send multiple messages to the same recipient in a round, $t+1$ identifiers are sufficient, even in the partially synchronous model. The second way is to increase the knowledge of the system for each process assuming each process knows how many processes share the same identifier.

Finally, we consider the Byzantine agreement in a natural extension of homonymous model, assuming that Byzantine processes can forge identifiers of correct processes.

Résumé

Un système distribué est constitué d'entités (ordinateurs, processus, capteurs...) qui communiquent entre elles par exemple en envoyant et recevant des messages. Dans la plupart des protocoles développés dans de tels systèmes, les différentes entités ont un nom unique qui permet de les identifier sans ambiguité. Ainsi le récepteur d'un message sait qui en est l'émetteur et l'émetteur sait vers qui le message a été envoyé. Cette hypothèse suivant laquelle il est toujours possible d'associer un nom unique à chacune des entités considérées est une hypothèse très forte qui est souvent difficile à réaliser réalisable et parfois non souhaitable.

Tout d'abord, dans de nombreux cas, avoir des identifiants uniques pour chaque entité est une hypothèse discutable. Par exemple, les identifiants peuvent provenir soit d'un nom physique comme une adresse MAC, soit d'un nom logique comme une adresse IP. Dans les deux cas il peut y avoir des adresses dupliquées de manière volontaire ou non. Prenons le cas des adresses MAC, elles devraient être en théorie infalsifiables, car inscrites sur la carte d'interface réseau et accessibles en lecture seule. Tout constructeur de carte d'interface réseau achète à un organisme centralisateur IEEE, un "numéro constructeur" de 22 bits qui est le début de l'adresse MAC. Chaque constructeur peut ensuite utiliser les 3 octets restés libres, et donner ainsi une adresse MAC unique à chaque carte. Mais en pratique ce n'est pas toujours le cas.

D'abord pour des raisons économiques, un constructeur peu scrupuleux peut donner pour

des cartes produites à grande échelle la même adresses MAC. Par ailleurs, les systèmes d'exploitation peuvent permettre, via le pilote de la carte, de modifier volontairement ces adresses.

Lorsqu'il n'y a pas d'intention malveillante et que toutes entités sont sur le contrôle d'une administration réseau, il est assez facile de maintenir l'unicité des identifiants. En cas de duplication, une fois la source de l'adresse non unique identifié, l'administrateur modifie l'adresse via le pilote de la carte. Par contre ce n'est pas du tout la même chose quand il y a des intentions malveillantes ou que les entités sont sur un réseau à large échelle. Une attaque très simple consiste à dupliquer une adresse MAC existante sur le réseau sur la machine hostile. Ainsi, un switch standard verra la même adresse sur deux de ses ports et diffusera les paquets sur les deux ports.

La situation est la même lorsque l'adresse est une adresse logique. Les adresses IP étant des adresses logiques sans authentification, il n'est pas difficile de les dupliquer et une attaque classique (masquerading attack, spoofing...) consiste justement à utiliser de fausses adresses IP. Dans des systèmes pair-à pair comme Chord [65] ou Pastry [62], les adresses logiques sont obtenues par une fonction de hachage (par exemple SHA-1). Bien que la probabilité soit très faible (on peut sans doute la considérer comme négligeable) il est possible qu'il y ait des collisions, plusieurs entités obtenant alors la même adresse. Mais surtout ces fonctions sont aussi potentiellement cassables [67], un utilisateur malveillant peut connaitre pour une adresse, les différentes entrées à donner à la fonction pour obtenir en sortie cette adresse. Et donc un utilisateur malveillant voulant usurper une adresse donnée pourra l'obtenir de la fonction d'adressage sans être suspecté. Dans le cas des réseaux pair-à-pair, une attaque classique est la "sybille attaque" pour laquelle une entité malveillante va créer de nombreux identifiants pour par exemple subvertir un système de réputation. Il s'agit encore ici d'un cas où une entité (malveillante) obtient plusieurs identifiants (logiques). Il est clair que des systèmes cryptographiques peuvent permettre

d'assurer l'authentification des entités sut le réseau et de cette façon assurer un système avec des identifiants uniques certifiés. Cependant, le coût en ressources et en infrastructure (gestion des certificats, autorité de certifications et de validation) est généralement trop élevé pour être utilisé dans de nombreux cas.

Au-delà de la difficulté d'assurer un système avec des identifiants uniques pour chaque entité, ce n'est pas toujours souhaitable. Une question de plus en plus importante avec le développement du numérique est de maintenir le respect vie privée (privacy). La façon la plus naturelle d'assurer ce respect de la vie privée est d'assurer une forme d'anonymat et pour des raisons de confidentialité on peut vouloir de ne pas devoir révéler son identifiant. C'est le cas par exemple pour un vote, où il n'est pas souhaitable que le vote individuel soit connu et associé à l'identité de chaque participant. C'est aussi bien sûr aussi le cas si l'on veut préserver des données personnelles. Dans toutes ces situations on sera amené à assurer une sorte d'anonymat, au moins partiel.

Pour assurer cet anonymat partiel ou partant du constat que d'avoir des identifiants uniques est souvent impossible à assurer, on pourrait essayer de se passer totalement d'identifiants. Le système est alors anonyme et il est impossible de distinguer une entité d'une autre entité. Ce genre de système pourrait être tout à fait satisfaisant pour un système client serveur, où les clients veulent connaitre l'identifiant du serveur mais pas l'inverse. On notera cependant que ce système n'est pas réellement anonyme puisque les clients doivent a priori être capables d'identifier de façon unique les serveurs. Mais surtout malheureusement, très peu de problèmes peuvent trouver une solution dans un système totalement anonyme.

Prenons comme exemple le problème classique de l'élection de leader. Si on dispose d'un leader dans un système distribué celui ci peut servir de coordinateur entre les différentes entités. Lorsque l'on dispose d'identifiants uniques et en l'absence de défaillances ce problème est facile à résoudre: il suffit par exemple de choisir l'entité ayant l'identifiant le

plus petit. Par exemple, si les entités sont organisées suivant un anneau où chaque entité peut communiquer avec son prédécesseur et son successeur sur l'anneau en échangeant des messages, chaque entité peut connaitre les identifiants des autres entités. Le leader est alors celui qui a l'identifiant minimale. Mais sans des identifiants uniques dans un système anonyme il devient impossible d'élire un leader sur ce même anneau [3]. Plus généralement dans un système anonyme très peu de choses peuvent être calculées. Il est, par exemple, impossible de compter le nombre d'entités dans le système distribué. On trouvera dans [5, 42] de nombreux autres exemples de problèmes impossibles à résoudre dans des systèmes anonymes. Bien sûr, en présence de défaillances des entités, Il y aura encore moins de problèmes pouvant être résolus.

En présence de défaillances, les problèmes d'accord sont fondamentaux. De façon assez naturelle la plupart de problèmes de tolérance aux défaillances peuvent se ramener à la possibilité de réaliser un accord entre des entités. Ainsi le problème du consensus a été très largement étudié depuis plus de trente ans. Ce problème est en effet à la base des techniques de réplication active qui permettent d'assurer un service en présence de défaillances : toutes les entités répliquées ont le même comportement et toutes traitent les requêtes des clients dans le même ordre en maintenant ainsi la même succesion d'états internes, elles donneront ainsi les même réponses aux clients. De cette façon, un client, en interprétant (par exemple par un vote) les réponses aux requêtes provenant des entités répliquées obtiendra une réponse à ses requêtes. Et même si certaines entités sont défaillantes, celles qui sont correctes assureront de façon cohérente le service requis auprès des clients. D'un point de vue plus formel, il s'agit de la "state machine approach" définie par Lamport [46].

Il est bien connu que, même avec des indentifiants uniques, ce problème est impossible à résoudre lorsque le système est asynchrone c'est -à-dire si l'on ne connait pas de bornes sur les relais d'acheminement des messages [34]. Par contre, dans un système synchrone avec

des indentifiants uniques on peut facilement résoudre se problème. Mais pour résoudre le problème du consensus dans un système synchrone les indentifiants uniques jouent un rôle fondamental. En particulier, il a été prouvé dans [55] que dans les systèmes anonymes le consensus est impossible, même avec un seul processus malveillant et même dans le cas synchrone.

Comme on le voit avec un système anonyme la plupart des problèmes ne peuvent pas être résolus, aussi nous proposons dans cette thèse d'étudier un modèle intermédiaire. Dans ce modèle les processus ont bien des identifiants mais ils ne sont pas nécessairement uniques: plusieurs processus peuvent avoir la même identifiant. Si l est le nombre d'indentifiant et n est le nombre de processus; le modèle où tous les processus ont des identifiants distincts ($l = n$) et le modèle où tous les processus sont anonymes ($l = 1$) sont les deux extrêmes du modèle que nous proposons ici. Ce modèle que nous appelons modèle avec homonymes permet donc à la fois de prendre en considération les modèles classiques où chaque entité a un identifiant unique et le modèle où les entités sont totalement anonymes. Il nous permettra de déterminer quel est le nombre d'identifiants nécessaires, c'est-à-dire aussi le niveau d'homonymie, permettant de résoudre certains problèmes.

Nous pensons que ce modèle avec homonymes avec permet de mieux prendre compte la situation réelle des systèmes distribués et, les processus étant partiellement identifiés, de résoudre, suivant le niveau d'homonymie, des problèmes qui ne pourraient l'être dans un système totalement anonyme. Le fait d'identifier plusieurs processus avec le même identifiant peut aussi permettre de préserver la confidentialité. Pour reprendre l'exemple du vote, si plusieurs processus ont la même identité, les autres processus savent que l'un d'entre eux a fait un vote particulier mais ils ne savent pas lequel. D'une manière générale, en considérant que les entités ayant le même identifiant font partie d'un même groupe, on ne pourra distinguer entre les membres d'un même groupe, ce qui maintient une certaine confidentialité à l'intérieur d'un groupe.

Cette thèse porte donc sur l'étude de ce modèle avec homonymes. Comme l'anonymat peut être lié à des défaillances (volontaires ou non) il est naturel d'étudier ce modèle en présence de défaillances des processus. Par ailleurs, comme rappelé ci-dessus, dans le cadre de la tolérance aux défaillances, les problèmes d'accord jouent un rôle fondamental, nous concentrerons essentiellement sur ceux-ci.
De façon plus générale, l'étude du modèle avec homonymes nous donnera une meilleure compréhension de l'importance des identités dans les systèmes distribués.
Rappelons tout d'abord les résultats connus sur le consensus dans le modèle où les identifiants sont uniques

Consensus et travaux reliés Dans le problème du consensus, chaque processus propose une valeur initiale et une spécification classique du consensus est:

- (Terminaison) de façon ultime, tous les processus corrects [1] décident.
- (Accord) Si deux processus corrects décident ils décident la même valeur.
- (Validité) Toute valeur décidée est une des valeurs proposées.

Il existe plusieurs versions du consensus suivant le type de pannes que l'on considère et au cours de cette thèse on précisera la spécification du consensus. Ainsi lorsque les pannes sont des pannes byzantines (un processus byzantin peut dévier arbitrairement de son code), la propriété de *Validité* devient: si tous les processus corrects proposent la même valeur alors cette valeur est celle décidée (par tout processus).
Dans les modèles de pannes bénignes: pannes par arrêt (crash), pannes par omission de réception et/ou omission d'émission on considérera une version plus forte du consensus dans laquelle les processus incorrects s'ils décident, doivent décider de la même valeur que les processus correct. Ce consensus est nommé uniforme.

[1] Un processus correct est celui qui n'a pas de défaillances

Le problème du consensus a été d'abord étudié dans le modèle des pannes byzantines [47, 52]. Dans les systèmes synchrones, en considérant un modèle de rondes [2] il est possible de trouver des algorithmes efficaces pour résoudre le consensus en présence de pannes crash et de pannes par omission d'émission ou de réception, par contre le problème du consensus dans le modèle des pannes byzantines est plus difficile. Dans [47], il est prouvé que si n est le nombre de processus et t est le nombre maximal de processus byzantins, alors le consensus est possible si et seulement si $n > 3t$. L'algorithme prouvant la condition suffisante est en $t + 1$ rondes ce qui est optimal comme le montre [33]. Cependant, la taille des communications échangées dans cet algorithme est exponentielles en t. Des algorithmes polynômiaux en la taille des communications du consensus ont été présentés dans [29, 26, 63, 66].

Ces algorithmes sont basés sur des diffusions authentifiées [63], qui garantissent que tous les processus corrects reçoivent exactement les mêmes messages à "peu près" dans la même ronde. Les algorithmes de consensus en présence de pannes byzantines dans cette thèse sont inspirés de cette approche et adaptent au contexte des homonymes ces diffusions authentifiées.

Dans le système partiellement synchrone où, à partir d'un certain moment qui n'est pas connu, les calculs se font en rondes synchronisées, des algorithmes de consensus ont été proposés dans plusieurs modèles des pannes [31].

Dans systèmes synchrones, en présence de pannes crash et de pannes par ommisision d'émission, contrairement au consensus qui est résolu même quel que soit le nombre de pannes, Neiger et Toueg [53] ont montré que le consensus uniforme ne peut être résolu si la moitié ou plus de la moitié de processus tombent en pannes par omission d'émission ou de réception. Plusieurs algorithmes de consensus uniforme sont proposés dans [57, 56, 60].

[2] Dans un modèle de rondes, à chaque rondes tous les processus corrects envoient un message, reçoivent les message des autres processus corrects et changent d'états.

Dans les systèmes anonymes, à notre connaissance, le consensus uniforme pour des défaillances par omissions d'émission ou de réception n'a pas été étudié. Pour le modèle des pannes byzantines, dans [55] il est observé que le consensus byzantin est impossible même avec un seul processus byzantin. Une solution dans un cas de partiel anonymat où les processus n'ont pas d'identifiant, mais chaque processus a un canal identifié distinct de communication avec les autres processus est proposée dans ce même article.

Contribution Dans le cas synchrone, nous donnons les conditions nécessaires et suffisantes pour résoudre le consensus pour tous les modèles des pannes, à partir des pannes crash, des pannes par omission d'émission, des pannes par omission d'émission ou de réception, jusqu'aux pannes byzantines. Notre résultat montre que dans les modèle avec pannes bénignes (pannes crash, des pannes par omission d'émission, et des pannes par omission d'émission ou de réception) des identifiants uniques ne sont pas nécessaires pour résoudre le consensus même uniforme si les processus sont capables de compter des messages identiques reçus dans une ronde, mais des identifiants uniques deviennent nécessaires avec des pannes byzantines. Plus précisément, si l est le nombre d'identifiants utilisées, n est le nombre de processus, et t est le nombre maximal de processus en pannes, alors le consensus byzantin est possible si et seulement si $l > 3t$. Il est intéressant de comparer les résultats : dans le cas des homonymes c'est le nombre d'identifiants qui compte et non le nombre de processus et la borne de moins d'un tiers de processus byzantins devient une borne de moins d'un tiers de processus byzantins par rapport au nombre *d'identifiants* Dans une certaine mesure ce résultat indique que l'existence d'homonymes n'empêche pas le consensus et montre que les modèles avec homonymes sont tout à fait envisageables dans le cas du synchrone.

Par contre dans des modèles partiellement synchrones, le consensus byzantin est possible si et seulement si $l > \frac{n+3t}{2}$. Cette borne indique que dans le modèle partiellement synchrone,

le consensus devient réellement plus difficile en présence d'homonymes: essentiellement, il faut que presque tous les processus aient des identifiants uniques. Cette borne montre aussi que contrairement à l'intuition, augmenter le nombre de processus corrects peut rendre le consensus impossible à résoudre.

Dans le chapitre 5, nous montrons que l'on peut améliorer la borne précédente si on enlève la capacité aux processus byzantins d'envoyer plusieurs messages dans une ronde. Nous montrons que $t+1$ identifiants sont suffisants pour résoudre le consensus byzantin au lieu de $\frac{n+3t}{2}$ identifiants. Comme pour les résultats précédents, il est intéressant de comparer ce résultat avec ceux concernant le modèles où chaque processus a une identité unique. On constate d'une part que dans ce modèle où la puissance des processus byzantins en tant qu'adversaires est strictement plus faible et d'autre part on retrouve la borne de $t+1$ que l'on a dans les modèles avec authentication mais ici concernant non le nombre de processus corrects, mais le nombre d'identifiants.

Dans le chapitre 6, on suppose que la répartition des homonymes par identifiant est connue. On donne une condition nécessaire et suffisante pour que le consensus byzantin soit réalisable en se basant sur une mesure, que l'on appellera *coefficient d'accord* d'un identifiant. Ce coefficient d'accord est l'estimation du nombre de processus corrects dans l'ensemble des processus ayant le même identifiant.

Dans tous ces travaux on a supposé que les processus byzantins ne pouvaient prendre qu'un seul identifiant et ne pouvaient pas prendre les identifiant de plusieurs processus. Il est assez naturel d'envisager que les processus byzantins peuvent aussi usurper l'identité d'autres processus. Ils peuvent alors perturber l'algorithme en jouant le rôle de plusieurs processus d'identifiants différents. Dans le chapitre 7 on a étudie dans ce cadre si le consensus reste possible dans un système synchrone. Lorsque k identifiants peuvent être usurpés par les t processus byzantins, on prouve que le consensus byzantin peut être résolu si et seulement si $l > 2t + k$. Mais en rajoutant un mécanisme d'authentification on peut

tolérer plus de processus byzantins: le consensus peut être résolu si et seulement si $l > t+k$. On constate a nouveau que l'existence d'homonymes n'empêche pas le consensus.

Dans l'annexe, comme extension de la thèse, on considère le problème de l'élection d'un leader sur un anneau dans notre modèle avec homonymes (mais sans défaillances). Ce travail est inspiré du résultat classique qui montre que l'élection d'un leader dans un anneau anonyme est impossible et qui donne des conditions permettant cette élection [3]. On montre que si l'élection d'un leader dans un anneau avec des homonymes est possible si et seulement si le nombre d'identifiants est supérieur au plus grand diviseur propre du nombre de processus n. En particulier, si n est un nombre premier alors deux identifiants sont suffisantes pour l'élection d'un leader.

Tous ces travaux montrent que pour résoudre de nombreux problèmes classiques de l'algorithmique répartie, il n'est pas nécessaire que tous les processus aient des identifiants distincts. On donne des bornes sur le nombre d'identités pour résoudre certains problèmes comme le consensus et l'élection d'un leader. Ces bornes sont dans de nombreux cas raisonnables: par exemple dans un système synchrone ces bornes permettent d'avoir beaucoup de processus partageant le même identifiant à condition que le nombre d'identifiants soit égal à au moins trois fois le nombre de processus byzantins. On peut donc envisager sans pénalités importantes de tels systèmes qui tout en permettant de résoudre des problèmes comme le consensus autorisent un partage d'identifiants qui garantissant, par l'indistingabilité entre processus ayant le même identifiant, la confidentialité.

Le modèle des homonymes, généralisation du modèle classique où tous les processus ont des identifiants distincts et du modèle où aucun processus n'a pas d'identité, peut être un modèle très pertinent pour remplacer les modèles classiques dans le futur.

Contents

Chapter 1

Introduction

1.1 Introduction

So far, the distributed computing community has considered two kinds of distributed systems, namely systems with unique identifiers where all processes have distinct identifiers and more rarely, anonymous systems where processes are anonymous and have no identifier [6, 10].

Systems with unique identifiers These systems can facilitate communication between processes. A process can send a message to a particular process and the receiver process can identify the sender of the message. However, in pratique, assuming that all processes have unique identifiers might be too strong because it may be difficult to keep the identifiers of all processes distinct when the number of processes in a network increases, or a network may accidentally assign the same identifier to different processes. For example, very simple systems giving MAC addresses as identifier are not reliable, because MAC addresses may be voluntary or not duplicated and do not always ensure the unicity of identifiers. Moreover, this kind of system is subject to Sybil attacks where the malicious processes may create multiple fake identifiers in particular for peer-to-peer and sensor networks

[30, 54]. The peer-to-peer systems aim to provide service to any user who wants to use the service (instead of, for example, only to a predetermined group of 15 users). Sybil attacks have already been observed in the real world [48] in the Maze peer-to-peer system. Researchers have also demonstrated [64] that it is surprisingly easy to launch sybil attacks in the widely used eMule system. In sensor networks, a large subset applications requires security. Security in sensor networks is complicated by the broadcast nature of the wireless communication. In the Sybil attack, a malicious node behaves as if it were a larger number of nodes, for example by impersonating other nodes or simply by claiming false identifiers. Thus, defending against sybil attacks is quite challenging for the systems with unique identifiers.

Anonymous Systems This kind of system is considered less frequently. One of the first works that addressed anonymous systems is from D. Angluin [3], where the connection between anonymous computations and the topological theory of graph coverings as shown. Based on this theory, functions and relations that are computable in anonymous networks were completely characterized in follow up papers, e.g., [11, 69].

As an example of anonymous system, web servers [61], some peer-to-peer file-sharing systems assume the peers are anonymous [14]. These systems are motivated from the privacy needs of users. More and more internet users will make the choice that the benefits of being connected to millions of strangers do not outweigh the amount of personal information revealed to those strangers. For example, in certain client-server services, the clients wish know the identifier of the server, but not vice versa. Evidently, anonymity is not the unique technique to guarantee privacy. For example, encrypting communication to and from web servers can hide the content of the transaction from an Internet service provider. It's a way to preserving privacy. However the provider can still learn the IP addresses of the client (identifier). With additional effort, this information can be

combined with other data to invade the privacy of clients even further. Moreover, there are some kinds of private information that can only be protected with anonymity technologies. For example, your web browsing habits might reveal a lot of information about you, even if the contents of your connection with each web site are fully protected by some encryption scheme.

Despite these advantages, unfortunately, no many problems in the anonymous systems are solvable.

The leader election problem is a simple example of a problem that is unsolvable in an anonymous system (see [3]). Leader election is the problem of electing a unique leader in a distributed network. It is required that all processes execute the same local algorithm. Leader election is a fundamental problem and has numerous applications in distributed computing. For example, it is an important tool for breaking symmetry in a distributed system.

In counting problem, nodes must determine the size of the network n and in naming they must end up with unique identifiers. The paper [51] showed that without a leader, counting is impossible to solve and that naming is impossible to solve even with a leader and even if nodes know n.

Some other unsolvable problems are also considered in [5, 42]. None of these considered process failures. Consensus is one of the most famous distributed computing problem in the presence of failures. This problem will be considered in detail in Section 1.2. In reliable message passing communication system, the consensus problem has no solution in (anonymous or not) asynchronous systems in the presence of failures, even only one process can be crash faulty [34]. Similarly, this problem is unsolvable in reliable shared memory communication systems [49].

In particular, it was proven in [55] that in fully anonymous systems Byzantine agreement is impossible even with only one malicious process and even in synchronous systems.

The other problems in anonymous systems with crash failures have been also considered in [7, 12, 15, 39].

Intuitively, the impossibility results is because symmetry is difficult to be broken in presence of anonymity. Randomization is a well known technique to break symmetry. For example, the randomized algorithm for the leader election is considered in [1], the randomized algorithm for consensus in anonymous shared memory systems in the presence of crash failures is considered in [15]. However, randomization problem is beyond the scope of this thesis.

Homonymous model In general, the model where processes have distinct identifiers and the model where processes are anonymous are two extremes of the model that we propose here: namely, n processes use l different authenticated identifiers, where $1 \leq l \leq n$. We call this model *homonymous* model. In the case where $l = 1$, all processes have the same identifier and they are therefore anonymous. In the case where $n = l$, processes have distinct identifiers. If $1 < l < n$, several processes may be assigned the same identifiers. We hope that with several identifiers, homonymous model can circumvent the disadvantages of the two kinds of classical systems.

Consider a scenario of *homonymous* model where 3 organizations LIAFA, EPFL and PPS constitute a distributed system. Users of an association might hide their own identifier and use only their name of association as identifier participating in a distributed protocol. One way to implement this would be to use the group signatures [19]: for example, LIAFA has its own secret key that all employees of LIAFA know, but nobody outside LIAFA knows. Firstly, because users use only their name of association as identifier participating in a distributed protocol, others will know only that some user within the association is participating but will not know exactly which one. Thus, the model can still preserve some level of anonymity. Secondly, assume that a new person come to LIAFA. He can

participate in the distributed protocol using the LIAFA'id while the system need not create a new id. Thirdly, the model is useful if security is breached, for example, when a malicious person or the adversary obtains the LIAFA's private key and then transmit error messages using LIAFA's id.

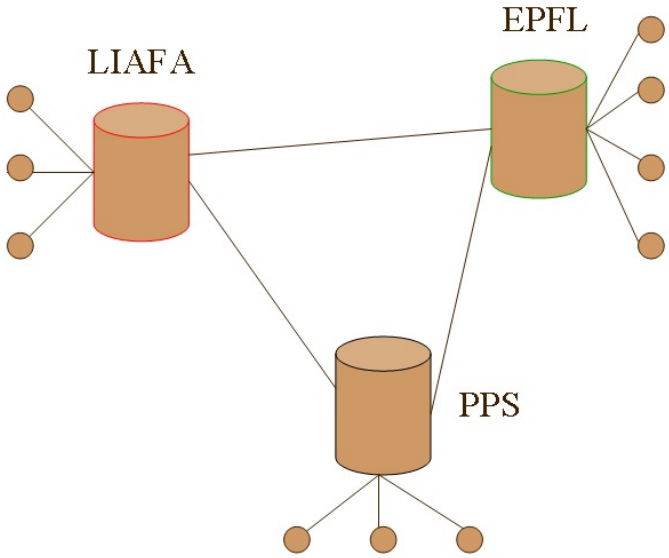

Figure 1.1: An example of homonymous model.

Moreover, homonymous model may be more useful if more problems of anonymous model become solvable. This is quite realizable because using several identifiers can be a way to break the symmetry.

Finally, from a theoretical point of view, studying homonymous model gives us a better understanding of the importance of identifiers in distributed computing.

1.2 Consensus Problem and Related Works

1.2.1 Consensus Problem

The problem of reaching consensus in a distributed system in the presence of failures is a fundamental problem and one of the most famous distributed computing problem. We consider a system of n processes and at most t faulty processes. The processes may be fail by halting prematurely (crash failure), by omitting to send messages when they should (send omission failures), by omitting to send or receive messages when they should (general omission failures), or by exhibiting arbitrary behaviors (Byzantine failures). A process that is not faulty is correct.

This coordination problem is defined as follows: each process proposes a value, and each correct process has to decide a value (termination), such that no two correct processes decide different values (agreement) and if the decided value is a proposed value (validity). To coordination, the processes can communicate by sending messages to one another (the message passing system) or by accessing shared objets (the shared memory system). The communication may be synchronous, asynchronous or partially synchronous [8, 31].

At first, the conditions may seem very easy to satisfy. Consider a naive algorithm as follows:

1. All processes select some process as leader (in classical model where all processes have distinct identifiers, leader selection is easy by choosing the smallest identifier.)

2. This leader sent its proposed value to all processes.

3. All processes decide on the value received from the leader.

If the leader is a correct process then all processes decide on the same value proposed by the leader. Hence, the consensus become be trivial. But if the leader is not correct process and it sends different messages to different processes then clearly, this algorithm

fails. We can think of the fact the processes should choose a correct process as the leader but unfortunately, no process has idea who faulty processes are and who correct processes are. The hardness of consensus depends on the form of asynchronism, types of failures and many other facets of uncertainly such as anonymity, dynamicity, scalability, mobility, etc. [43].

Uniform consensus: In this thesis, we also consider a stronger version of the consensus problem, the uniform consensus problem. In the consensus problem, the agreement condition, namely "no two non faulty processes decide differently", may allow two processes (a non faulty process and a faulty process) to disagree. Clearly, such disagreements are undesirable in many applications since they may lead the system to inconsistent states. Formally, the uniform agreement condition specifies that no two processes (whether faulty or not) decide differently.

We note that the term consensus is frequently used in the context of benign failures (crash, send omission and general omission failures) while for Byzantine failures, the term Byzantine agreement (BA) is often preferred. In this thesis, henceforth we use the term consensus for benign failures, the term Byzantine agreement for Byzantine failures.

Applications of consensus include whether to commit a transaction to a database, leader election, state machine replication, and atomic broadcasts.

1.2.2 Related Works

The consensus has been first studied in Byzantine failure model [47, 52] in systems with unique identifiers. Afterwards, numerous results have been stated for consensus in benign failures [34, 17, 25, 31]. The results are summarized in Table 1.1.

In the synchronous system, while the efficient algorithms for reach the consensus in benign failures model are simple, the Byzantine agreement is more difficult. An algorithm for reaching BA in $t + 1$ rounds for $n > 3t$ was presented in [47]. Shortly after, it was shown

	Crash, send omission, general omission	Byzantine
Synchronous	$n \geq t$	$n > 3t$
Partially synchronous	$n > 2t$	$n > 3t$
Asynchronous	Impossible	Impossible

Table 1.1: Necessary and sufficient conditions on the number of processes for solving the consensus in a system of n processes and tolerating t faulty processes.

that no deterministic algorithm can solve the problem in less than $t + 1$ rounds [33]. However, the original BA algorithm requires computation and communication that are exponential in t, thus leaving the design of more efficient algorithms as an open problem. One of the first polynomial BA algorithms was presented in [28] that required $2t + 3$ rounds. The BA algorithms presented in [29, 26, 63, 66] are of direct relevance to this thesis. These algorithms are based on a consistent broadcast primitive [63], which ensures that all the correct processes receive exactly the same messages at almost the same time. A consistent broadcast primitive allows to design simple polynomial algorithms for the consensus problem and some other coordination problems. The consensus algorithms presented in this thesis for Byzantine failure model are inspired by this approach.

Uniform consensus: Uniform consensus is trivially not solvable in systems with Byzantine failures because Byzantine processes have not limitation on their behaviors and then on their decisions. In the general omission failures model and synchronous systems, in contrast to consensus that is solvable no matter how many processes are faulty, Neiger and Toueg [53] showed that uniform consensus cannot be solved if half or more processes may fail. Several algorithms for uniform consensus are also proposed in [57, 56, 60]. In the crash failure model, it is not difficult to solve both consensus and uniform consensus in presence of any number of faulty processes. Interestingly, the paper [37] showed that in most partially synchronous systems, any algorithm that solves consensus also solves uniform consensus. However, there is a difference considering the early decision [27] in the consensus and the uniform consensus algorithms. The early decision examines the num-

	Crash	send omission	General Omission	Byzantine
Synchronous	$n \geq t$	$n \geq t$	$n > 2t$	Impossible
Partially synchronous	$n > 2t$	$n > 2t$	$n > 2t$	Impossible
Asynchronous	Impossible	Impossible	Impossible	Impossible

Table 1.2: Necessary and sufficient conditions on the number of processes for solving the uniform consensus for benign failures model and Byzantine agreement for Byzantine failures model in a system of n processes and tolerating t faulty processes.

ber of rounds needed for consensus in relation to the actual number of faulty processes in an execution of algorithm rather than to t, the maximal number of faulty processes. The paper [18] showed that compared with the best consensus algorithm, any uniform consensus algorithm takes at least one additional round to take a decision. Thus, uniform consensus algorithm is harder than consensus whatever the failure model is. The results are summarized in Table 1.2.

Consensus and uniform consensus in the anonymous system: the previous works focus on the crash failure model. In [21, 13], fault-tolerant consensus in asynchronous model is solved under the assumption of failure detectors (that is a well known approach to provide each process with information on failures [17]). In [23], fault-tolerant consensus is solved in unknown and anonymous networks and in several partially synchronous systems. To the best of our knowledge, uniform consensus in send omission and general omission failures model were not studied so far.

When the failure model is Byzantine, the paper [55] showed that Byzantine agreement is impossible even with only one Byzantine process. In the model with a restricted kind of anonymity where 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, Byzantine agreement can be solved when $n > 3t$.

In this thesis, with the goal to determine the power of the *homonymous* model, we focus on the consensus problem.

	Crash	send omission	General Omission	Byzantine
Synchronous	$n \geq t$	?	?	Impossible
Partially synchronous	$n > 2t$	?	?	Impossible
Asynchronous	Impossible	Impossible	Impossible	Impossible

Table 1.3: Necessary and sufficient conditions on the number of processes for solving the uniform consensus for benign failures model and Byzantine agreement for Byzantine failures model in an anonymous system of n processes and tolerating t faulty processes. The results that were not studied are denoted by ?.

	Crash, send omission	General omission	Byzantine
Innumerate processes	$l \geq 1$	$l > 2t$	$l > 3t$
Numerate processes	$l \geq 1$	$l \geq 1$	$l > 3t$

Table 1.4: Necessary and sufficient conditions on the number of identifiers for solving the uniform consensus for benign failures model and Byzantine agreement for Byzantine failures model in a system of n processes using l identifiers and tolerating t faulty processes.

1.3 Contributions

In the synchronous case, we give the complete picture on necessary and sufficient conditions on the number of identifiers for solving consensus in a system of n processes using l identifiers and tolerating t faulty processes. The results are summarized in Table 1.4. In our results, we consider both cases: (1) when processes are *numerate*, i.e. they can count the number of copies of identical messages that they received in a round and (2) when processes are *innumerate* (they have not this ability). In systems with unique identifiers, since senders can append their identifier to all messages, a receiver process can easily count copies of messages. This is not possible in homonymous model, so the distinction between numerate and innumerate processes is important.

For the case of benign failures, Table 1.4 shows that if only crash failures or send omission failures, uniform consensus is solvable even if processes are innumerate and have no identifiers while for general omission failures, the ability of counting or the identifier is

required. More precisely, if processes are numerate then uniform consensus is solvable without using identifier of processes (we need only the condition of number of processes as in systems with unique identifiers, $n > 2t$). If processes are innumerate, $2t + 1$ identifiers are necessary to reach the uniform consensus. For the case of Byzantine failures, we show that the Byzantine agreement is solvable if and only if $l > 3t$ whether processes are numerate or not. Thus, identifiers are not useful for crash and send omission failures or when processes are numerate however for general omission or for Byzantine failures identifiers become important.

We also consider the partially synchronous case for the Byzantine agreement. We show that the lower bound $l > \frac{n+3t}{2}$ is necessary and sufficient condition to reach to the agreement for Byzantine failures. This bound has several surprises: first, the number of required identifiers l depends on n as well as t. Second, more correct processes may make the problem harder. For example, if $t = 1$ and $l = 4$, agreement is solvable for 4 processes but not for 5. Finally, since $\frac{n+3t}{2}$ is strictly greater than $3t$, partially synchronous model is strictly weaker than synchronous model in homonymous model.

In Chapter 5, we remove the ability for a Byzantine process of sending multiple messages to the same recipient. A Byzantine process has no this ability is called *restricted* Byzantine process. Consequently, we can notably reduce the lower bound of the number of identifier necessary to reach agreement in partially synchronous systems from $\frac{n+3t}{2}$ to t (see the Table 1.5). These results come from the fact that in a system with unique identifiers, the Byzantine process has no advantage of sending multiple messages to a single recipient in a round: algorithms could simply discard such messages, however, in homonymous systems, that advantage is clear.

For the results above, we assume that no knowledge of the system except n, l, t is available for each process and each identifier must be assigned to at least one process. However, in chapter 6, we increase the knowledge of the system for each process assuming each

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

Table 1.5: Necessary and sufficient conditions for solving Byzantine agreement in a system of n processes using l identifiers and tolerating t Byzantine failures. In all cases, n must be greater than $3t$.

process knows the distribution of identifiers, i.e, each process knows how many processes share the same identifier. Considering the synchronous case, we show that this way may reduce the bound necessary to reach to consensus. For example, assuming that a system has $n = kl$ processes, l identifiers, t Byzantine processes such that for each identifier there are k processes that share it, then the necessary and sufficient condition to reach the consensus is $l > t(1 + 1/k)$ instead of $l > 3t$. More precisely, for each distribution of the identifiers we give a necessary and sufficient condition that enable us to solve the Byzantine agreement, using a new notation, namely the *agreement coefficient* of a group. The agreement coefficient denotes the estimation of the number of correct processes of a group. This problem is considered in detail in Chapter 7. Finally, we show that there exists a distribution of identifiers enabling to solve the Byzantine agreement if and only if $n > 3t$ and $l > \frac{(n-r)t}{n-t-min(t,r)}$ where $r = n \bmod l$. Note that $t < \frac{(n-r)t}{n-t-min(t,r)} < 2t$.

In Chapter 7, we present a natural extension of homonymous model in which some identifiers are forgeable and a Byzantine process may freely use any such identifiers. Here we assume that at most t processes may be Byzantine and at most k ($t \leq k \leq l$) of these identifiers are forgeable. Intuitively, we may consider the behavior of a group of processes with forgeable identifier as one of the group of processes with Byzantine processes. Then assuming that k forgeable identifiers, we get directly a solution for the Byzantine agreement if $l > 3k$ using the result in Table 2 for the synchronous model. But surprisingly, we give a better bound. We prove that $l > 2t + k$ is necessary and sufficient condition for

the BA. Moreover we extend this result to systems with authentication by signatures in which at least $l - k$ signatures are unforgeable and we prove that Byzantine Agreement is possible if and only if $l > t + k$. Interesting, the solvability of Byzantine agreement depends only on the number of identifiers and the number of forgeable identifiers. Hence adding correct processes does not help to solve Byzantine agreement.

Finally, in the appendix, as an extension of the thesis, we consider the leader election problem in a ring in our model with homonyms (without failures). This work is inspired by the classical result that shows that the leader election in an anonymous ring is impossible [3]. We show that the leader election in a ring with homonyms is possible if and only if the number of identifiers is greater than the largest proper divisor of number of processes. In particular, if the number of processes is a prime number then two identifiers are sufficient for the leader election.

Most contribution in this thesis originally appeared in the following papers:

- " Byzantine agreement with homonyms". Joint work with Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec and Eric Ruppert. Accepted in Distributed Computing journal.

- " Byzantine agreement with homonyms in synchronous systems. Joint work with Carole Delporte-Gallet, Hugues Fauconnier. Accepted in Theoretical Computer Science journal.

- " Uniform Consensus with Homonyms and Omission Failures". Joint work with Carole Delporte-Gallet, Hugues Fauconnier. Published in ICDCN 2013.

- " Byzantine agreement with homonyms in synchronous systems". Joint work with Carole Delporte-Gallet, Hugues Fauconnier. Published in ICDCN, vomume 7129 of LNCS, pages 76-90, 2012.

- “ Homonyms with forgeable identifiers”. Joint work with Carole Delporte-Gallet, Hugues Fauconnier. Published in SIRROCCO, vomume 7355 of LNCS, pages 171-182, 2012.

- “ Byzantine agreement with homonyms”. Joint work with Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec and Eric Ruppert. Published in PODC, pages 21-30. ACM, 2011.

Chapter 2

Model and Definitions

2.1 Homonymous Model

2.1.1 Definition

We consider a distributed message-passing system of n processes. Each process gets an identifier from a set of identifiers $\mathcal{L} = \{1, 2, \ldots, l\}$. We assume that $n \geq l$ and that each identifier is assigned to at least one process. In the case where $n > l$, one or more identifiers will be shared by several processes. In the case where $l = 1$, all processes have the same identifier, and they are therefore anonymous. We call such model *homonymous* model. The processes with the same identifier are said to be *homonyms*.

2.1.2 Communication in the homonymous model

We consider the homonymous model in the distributed message-passing communication in the synchronous and partially synchronous cases. We assume that the message-passing primitive is reliable, i.e. it does not create, duplicate, or alter messages.

Synchrony case The most of our results are done 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.

Partially synchrony cases In chapter 5 and 6, we also consider homonymous model under assumptions of partially synchronous model of Dwork, Lynch and Stockmeyer [31]: 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 [31], 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. Interestingly, this simulation does not use identifiers of processes and does depends on types of faults. Thus, our characterization of the values of n, l and the number of faulty processes t for consensus problem can be solved applies to the other models with partially synchronous communication too.

In homonymous model, a process knows the identifier of the sender of each message it receives. Given a message m, we denote by $m.val$ its value (or content) and by $m.id$ the identifier of the sender. Since processes with the same identifier are supposed to be indistinguishable in our model, the process cannot send different messages to two processes with the same identifier during a single round. Moreover, given any message some process p has received, it cannot determine if this message comes from the same sender process or from different sender processes with the same identifier of p .

Without loss of generality, we assume that processes are provided a broadcast: a process can send the same messages to all processes (including itself). If a process wishes to send different messages to processes with different identifiers, it can include the intended recipients' identifiers in the message itself.

Innumerate and numerate processes In systems with unique identifiers, the senders can append their identifier to all messages, making it trivial for the receiver to count copies of messages. This is not possible in the systems with homonyms. In our model, we distinguish the cases where processes are *innumerate* from the case where they are *numerate*. We say that a process is innumerate if the messages it receives in a round form a set of messages: the process cannot count the number of copies of identical messages it receives in the round. We say that a process is numerate if the messages it receives in a round form a multiset of messages: the process can count the number of copies of identical messages it receives in the round.

2.1.3 Failures Assumptions

A process is faulty during an execution if its behavior deviates from that prescribed by its algorithm, otherwise it is *correct*. Differently from correct processes that cannot send different messages to two processes with the same identifier during a single round, faulty processes are not constrained in this way: they can communicate to each process.
We consider here the following failure models:

- 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.

- Byzantine failure: A faulty process may deviate arbitrarily from its code. It may send messages different from its algorithm specification or fails to send the messages it is supposed. Moreover, in contrast to correct processes that can not direct different messages to two processes with the same identifier during a single round, Byzantine processes can send messages to each process. However, we assume that Byzantine process cannot forge identifiers.

- Restricted Byzantine failure: A Byzantine process is restricted to sending at most one message to each recipient in each round.

In pratique, 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. An intuitive explanation of the fact that it is more difficult to cope with receive omission failures than with send omission failures is the following: A process that commits only send omission failure continues to receive the messages sent by the correct process. Differently, a process that commits receive omission failures does not: it has an " autism " behavior [60]. For Byzantine processes, they are intended to model arbitrarily bad failures in the system which might, for example, corrupt messages going to individual processes.

2.1.4 Failure Pattern

In homonymous model, a failure pattern attaches to a distribution of identifiers.
Consider a mapping Id from the set of processes into the set $\mathcal{L}$. A mapping Id defines a partition of integer n into l parts $n_1, \ldots, n_l$ such that for each identifier j, n_j is the number of processes with identifier j. This partition of n into l parts will be called a *distribution of l identifiers* :

Definition 2.1.1. *Given a set of l identifiers, a distribution $D(n,l)$ is a partition of n into l parts denoted $D(n,l) = \langle n_1, \ldots, n_l \rangle$ with $n_1 \geq n_2 \geq \ldots \geq n_l > 0$ and $n = \Sigma_{i=1}^{i=l} n_i$.*

All properties that we consider depend only on distributions of identifiers and we do not distinguish between mappings Id having the same associated partition.
The failure pattern is defined as follows:

Definition 2.1.2. *Let $D(n,l) = \langle n_1, \ldots, n_l \rangle$ be a distribution, a* failure pattern *$F(D(n,l))$ for at most t faulty processes is a l-tuple $(t_1, \ldots, t_l)$ such that for every $i \in \{1, \ldots, l\}$ $0 \leq t_i \leq n_i$ and $\Sigma_{i=1}^{i=l} t_i \leq t$.*

Let $G(i)$ be the set of processes with identifier i. We name such a set a *group*. We consider the case of Byzantine processes. If $t_i = 0$ then all processes with identifier i are correct and in this case the group $G(i)$ is said *correct*, if all processes of $G(i)$ are Byzantine, namely if $t_i = n_i$, the group $G(i)$ is said *fully Byzantine*, and in all other cases $G(i)$ is said *partially Byzantine*.
The set of all possible failure patterns for distribution $D(n,l)$ and at most t Byzantine processes is called a failure environment, denoted $\mathcal{F}_t(D(n,l))$: the only condition is here that the number of Byzantine processes is at most t.

Definition 2.1.3. *The adversary for distribution $D(n,l)$ and at most t Byzantine processes, $\mathcal{F}_t(D(n,l))$, is the set of all failure patterns $(t_1, \ldots, t_l)$ such that $\Sigma_{i=1}^{i=l} t_i \leq t$.*

2.1.5 Homonymous model with forgeable identifiers

We consider an extension of homonymous model in which some identifiers are forgeable. We assume that Byzantine processes have the power to forge some identifiers. An adversary is the set of all failure patterns $(t_1, \ldots, t_l)$ such that $\Sigma_{i=1}^{i=l} t_i \leq t$ and a subset of $\mathcal{L}$ denoted $\mathcal{F}$ where Byzantine processes can forge some identifier of $\mathcal{F}$. In the following k designs an upper bound of the number of identifiers that can be forged: $|\mathcal{F}| = k$.

In this extended model, if $t_i = 0$ and $i \notin \mathcal{F}$ then group $G(i)$ is said *correct*, if all processes of $G(i)$ are Byzantine, namely if $t_i = n_i$, the group $G(i)$ is said *fully Byzantine*, and in all other cases $G(i)$ is said *partially Byzantine*.

Communication: Let id_f be an identifier in $\mathcal{F}$, a Byzantine process with identifier id may send a message m to a process p with identifier id' with the forged identifier id_f. In this case, the process p receives the message m with $m.id = id_f$. As a Byzantine process acts as an adversary it may divulge any information, then we assume here that if q is a Byzantine process then $Id(q)$ is also in $\mathcal{F}$. Consequently, if a process p receives a message m with $m.id = id$, p knows that this message has been either sent by a correct process with identifier id or sent by a Byzantine process which has forged the identifier id.

We name (n, l, k, t)-homonymous model such a model. In the following a correct group designs a group of processes with some identifier that contains only correct processes and whose its identifier is not forgeable.

2.2 Consensus Problem

In the case of send omission and general omission failures, the uniform consensus problem is defined in [56] by the following three properties:

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

In the case of Byzantine processes, we consider the classical *Byzantine agreement* problem [34, 58], defined by the following three properties.

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

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

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

An algorithm solves the consensus 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.)

Chapter 3

Uniform Consensus in omission failure model

3.1 Introduction

In this chapter, we consider the uniform consensus problem restricted to benign process failures of a system of n processes, l identifiers and at most t faulty processes.

We show that, concerning crash failures or send-omission failures, uniform consensus is solvable even if processes are anonymous and innumerate. Our algorithm is based on underlying principles in [59, 63]. Moreover the solution we propose is in $t+1$ rounds and is then optimal [2, 32]. Concerning general-omission failures, we show that in the case where processes are numerate, uniform consensus is solvable if and only if $n > 2t$, that is with the same bound as for processes having different identifiers. In the case where processes are innumerate, $2t+1$ identifiers are necessary. To prove that, we use a scenario argument. Moreover, our algorithm is early-stopping. It allows correct processes to decide and stop by round $min\{f+2, t+1\}$ and allows the faulty processes to decide and stop by round $min\{f+3, t+1\}$, where f is number of processes that are actually faulty during a

given execution ($f \leq t$).

3.2 Consensus with send-omission failures

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

Crash-tolerant uniform consensus protocols in models in which processes have distinct identifiers described in [8, 50, 44] are based on a "flood set" strategy. Each process p maintains a local variable that contains its current estimate 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 value among the estimates it has received. After $t + 1$ rounds, as there is at least one round without any new 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, In [53, 59, 63], 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 [38], the current estimate of each process is updated to the current estimate of the leader selected in each round. All solutions 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 [59, 63]. 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 and 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 6.2. 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 while *old* is set to 0. 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 $new < old$ then 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* of 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 minimum value of the previous value of *new* and all values v received from processes having changed their estimate in the previous round.

Finally, at round $t+1$, each process decides on the maximum current estimate value of variable *new* it has seen in the round $t+1$.

For the proof, 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 ", 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 alive p at the end of the round r, $old_p(r)$ be the value of variable *old* of the alive process p at the end of the round r, $V[r]$ be the set of values of variable *new* of all alive processes at the end of the round r: $V[r] = \{new_p(r) | p$ is process alive in round r $\}$, and v_{max} be the largest value of set $V[t]$. Henceforth, in all proofs and algorithms, if we talk about some process p in round r then that means that process p is

```
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}
```

Figure 3.1: Anonymous Consensus Protocol with at most t send omission processes

alive in that round.

We begin with two simple facts:

Fact 3.2.1. *For every process p and every round $1 \leq r < t$, $old_p(r+1) = new_p(r)$.*

Fact 3.2.2. *For every process p and every round $1 \leq r \leq t$, $new_p(r) \leq old_p(r)$.*

Lemma 3.2.1. *For every round $1 \leq r < t$, $V[r+1] \subseteq V[r]$.*

Proof. Consider the variable $new_p(r+1)$ where p is some process. There are only 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.
- p keeps its estimate in this round. Hence, $new_p(r+1) = old_p(r+1) = new_p(r)$.

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

We directly get:

Lemma 3.2.2. *For every process p and every round $1 \leq r \leq t$, p_0 is correct then $new_p(r) = v_{min}$.*

Lemma 3.2.3. *If a correct process p has input value v_0 or changes its estimate to some value v_0 in round $1 \leq r < t$ then for every process q, $new_q(t) \leq v_0$.*

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 8.3.1, we have $V[t] \subseteq V[1]$. Thus, for every process q, $new_q(t) \leq v_0$.
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 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 some process q. Set $G_q[r+1]$ contains at least one element and,

- If $Min\{v|v \in G_q[r+1]\} < old_q(r+1)$ then q sets *new* to $Min\{v|v \in G_q[r+1]\} \leq v_0$.
- If $Min\{v|v \in G_q[r+1]\} \geq old_q(r+1)$ then q keeps its estimate and we have $v_0 \geq Min\{v|v \in G_q[r+1]\} \geq old_q(r+1) \geq new_q(r+1)$.

Hence, for any process q, $new_q(r+1) \leq v_0$. By Lemma 8.3.1, we have $V[t] \subseteq V[r+1]$. Thus, for every process q, we get $new_q(t) \leq v_0$. □

Lemma 3.2.4. *If $t > 1$ and $r \geq 2$, and if some process 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 $new = v_0$ and

$new < old$. 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, the value of its variable new is less than or equal to v_0 and no process can change its estimate to v_0 twice. Thus, all the 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 3.2.5. *We have either* (a) *for every correct process q, $new_q(t) = v_{max}$, 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 $t = 1$, consider two cases:

- p_0 is correct and then by Lemma 8.3.4, $new_q(t) = v_{min}$ for all processes q.
- p_0 is faulty, thus either all correct processes have same the value new, or $V[t] = \{v_{min}, v_{max}\}$ and some correct process changes its estimate and its value new must be v_{min}, proving the Lemma.

If $t > 1$, 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.

Consider any correct process p such that $new_p(t)$ is v_0. Thus we have either $v_0 = v_{max}$ or $v_0 < v_{max}$. If $v_0 < v_{max}$ then let us consider 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 8.3.2, $new_q(t) \leq v_0$ for every process q. By definition of v_{max}, we must have $v_{max} = v_0$, contradicting the hypothesis that $v_0 < v_{max}$.

- p changes its estimate to v_0 in round t. By Lemma 8.3.3, we have a set of processes $\{q_1, \cdots, q_{t-1}\}$ such that q_i changes the value of its 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 $v_0 < v_{max}$. On the other hand, p_0 must be faulty because if p_0 is correct then by Lemma 3.3.2, $new_p(1) = v_{min}$ and p can not change its estimate in round t, contradicting the hypothesis. Therefore, we get a set of t faulty processes $\{q_1, ..., q_{t-1}\} \cup \{p_0\}$.

  Since a process changes its estimate only to a smaller value and all processes q_i change 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 8.3.2, 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}, we must have $v_{max} = v_0$, contradicting the hypothesis that $v_0 < v_{max}$. 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, after having changed the estimate to $v_0 = v_{min}$, all processes q_i do not change their estimate again because v_{min} is the minimum value. We get $new_s(t) = v_{min}$ for every faulty process $s \in \{q_1, ..., q_{t-1}\} \cup \{p_0\}$.

Thus, for every correct process p, we get always either

(a) $new_p(t) = v_{max}$

or

(b) $new_p(t) = v_{min}$ and $new_p(t) < v_{max}$. Moreover, p changes its estimate to v_{min} in round t and for every faulty process s, $new_s(t) = v_{min}$.

Therefore, we have either

(a) for every correct process q, $new_q(t) = v_{max}$,

or

(b) $V[t] = \{v_{min}, v_{max}\}$ and some correct process changes its estimate to v_{min} in round t.

Moreover, for every faulty process s, $new_s(t) = v_{min}$. □

Proposition 3.2.1. ***Uniform agreement****: No two processes decide different values.*

Proof. By Lemma 3.2.5, either

- If for every correct process q, $new_q(t) = v_{max}$ then, every process p receives message v_{max} in round $t + 1$ and decides v_{max} at Line 15.
- or $V[t] = \{v_{min}, v_{max}\}$ and for every faulty process s, $new_s(t) = v_{min}$. Hence, v_{max} comes from some correct process. At the beginning of round $t+1$, this process sends v_{max} to all and every process receives the value v_{max} and decides v_{max}.

□

Proposition 3.2.2. ***Uniform Validity****: If a process decides v, then v is proposed by some process.*

Proof. Since $n > t$, there is at least one correct process p. p sends its variable *new* to all processes in round 1. From Line 15, if a process decides then the decision value belongs to the set $V[t]$. By Lemma 8.3.1, $V[t] \subseteq V[1]$. Moreover, $V[1]$ is a subset of the inputs proposed by processes. Therefore, if a process p decides v, then v is the input value of some process. □

Proposition 3.2.3. ***Termination****: Every correct process eventually decides.*

Proof. By the algorithm of Figure 3.1, all processes that do not crash decide at Line 15. □

From the previous propositions we deduce:

Theorem 3.2.1. *Uniform consensus is solvable within $t + 1$ rounds with send-omission failures of any number of processes even if all processes are anonymous.*

3.3 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 to solve uniform consensus. 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 $n > 2t$.

3.3.1 Numerate processes

We assume in this subsection that processes are anonymous and numerate.

Differently from the consensus problem that allows a faulty process to decide differently from the correct processes, the uniform consensus problem requires that the deciding processes, correct or not, decide the same value. Algorithms in general-omission failures model solving uniform consensus when processes have distinct identifiers appear for example in [57, 56, 60]. These protocols are based on a flood set strategy with the elimination of the processes suspected to be faulty. They also use the fact that the processes are "self-trusting ", in the sense that a process first suspects the other processes before wondering if it is itself faulty. However, in our case, a process cannot suspect another process because the processes are fully anonymous.

We present an early stopping algorithm in Figure 3.2 that solves the consensus in general omission model. It based on the same principles as Figure 3.1. Here, 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 $t + 1$.

We now present the steps of the proof that the protocol of Figure 3.2 satisfies the specifica-

```
Code for process p
Variable:
1  input = {v};                                          /* v is the proposed value */
2  new = input;
3  old = 0;
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    if it receives less than n − t messages in this round then decides ⊥ and stops
14    if it receives at least n − r + 2 pairs (v, ∗) in this round then decides v
15 ROUND t + 1
16    send (new, old) to all processes
17       if for some v, n − t pairs (v, ∗) are received in this round
18       then  decides v
19          else if at least n − t pairs
20                   and one of them is (x, y) such that x < y are received
21                   then
22                      let G_p = {v| p received (v, v_1) in this round }
23                      let H_p = {v| p received (v, v_1) in this round and v < v_1}
24                      if Min{v|v ∈ G_p} = Min{v|v ∈ H_p} then decide Min{v|v ∈ G_p}
```

Figure 3.2: Anonymous Consensus Protocol with at most t general-omission processes

tion of uniform consensus. We use the same notations for $p_0, v_{min}, V, new_p(r)$ and $old_p(r)$ as in the proof of Theorem 3.2.1. Let $C[r]$ be the set of values of variable new of all correct processes at the end of the round r: $C[r] = \{new_p(r) | p \text{ is a correct process }\}$, 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 3.2.1.

Lemma 3.3.1. *$V[r+1] \subseteq V[r]$ for every round $1 \leq r < t$.*

Lemma 3.3.2. *If p_0 is correct then $new_q(t) = v_{min}$ for every correct process q.*

Lemma 3.3.3. *If a correct process changes its estimate to some value v_0 in round $1 \leq r < t$ or has input v_0 then $new_q(r') \leq v_0$ for every correct process q and every round $r' > r$.*

Lemma 3.3.4. *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 3.3.5. *We have either* (a) *$new_q(t) = c_{max}$ for every correct process q, or* (b) *$V[t] = \{v_{min}, c_{max}\}$ and there are t faulty processes such that for every faulty process s, $new_s(t) = v_{min}$. Moreover, d some correct process changes its estimate to v_{min} in round t.*

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

- p_0 is correct then by Lemma 3.3.2, $new_q(1) = v_{min}$ for all correct processes.
- p_0 is faulty. At the end of round 1, either all correct processes have the same value *new*, or $V[t] = \{v_{min}, c_{max}\}$ and some correct process changes its estimate then its value *new* must be 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.
Consider any correct process p such that $new_p(t)$ is v_0. Thus either $v_0 = c_{max}$ or $v_0 < c_{max}$.
If $v_0 < c_{max}$ then let us consider following two cases:

- v_0 was the input value of p or p changes its estimate to v_0 in a round $1 \leq r < t$ then by Lemma 3.3.3, $new_q(t) \leq v_0$ for every correct process q. By definition of c_{max}, we 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 3.3.4, 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 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 must be faulty because if p_0 is correct then by Lemma 3.3.2, $new_p(1) = v_{min}$ and p cannot 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 a process changes only its estimate to a smaller value and all processes q_i changes 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 3.3.3, 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}, we must have $v_{max} = v_0$, contradicting the hypothesis that $v_0 < v_{max}$. 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, after having changed the estimate to $v_0 = v_{min}$, all processes q_i do not change their estimate again because v_{min} is the minimum value. We get $new_s(t) = v_{min}$ for every faulty process $s \in \{q_1, ..., q_{t-1}\} \cup \{p_0\}$.

Thus, for every correct process p, we get always either

(a) $new_p(t) = c_{max}$

or

(b) $new_p(t) = v_{min}$. Moreover, p changes its estimate to v_{min} in round t and there are t faulty processes such that for every faulty process s, $new_s(t) = v_{min}$.

Therefore, we have either

(a) for every correct process q, $new_q(t) = c_{max}$,

or

(b) $V[t] = \{v_{min}, c_{max}\}$ and there are t faulty processes such that for every faulty process s, $new_s(t) = v_{min}$. Moreover, d some correct process changes its estimate to v_{min} in round t. □

Furthermore:

Lemma 3.3.6. *If at round $r : 2 < r < t + 1$, a process p decides v_0 at Line 14 then*

(1) no process changes its estimate to a value less than v_0 in round $r - 1$,

(2) for every correct process s and $r' \geq r - 1$, $new_s(r') = v_0$.

Proof. As p decides v_0 at Line 14, it must receive at least $n - r + 2$ pairs $(v_0, *)$. As $n - r + 2 > t$, at least one correct process sent $(v_0, *)$ to p in round r.

To prove (1), suppose that some process changes its state to the value $v_1 < v_0$ in round $r - 1 > 1$. By Lemma 3.3.4, there is some set A of $r - 2$ processes that changed their estimate to v_1 before round $r - 1$. By 3.3.3, if p_0 is correct then for every correct process s, $new_s(r - 2) = v_{min}$, if processes of A are correct then for every correct process s, $new_s(r - 1) \leq v_1$. In both cases, no correct process can send $(v_0, *)$ in round r because $v_{min} \leq v_1 < v_0$, contradicting (*). Thus, we have $r - 1$ faulty processes, including p_0 and $r - 2$ processes in A. Moreover, at the beginning of round r, the estimates of all these processes are less than or equal to v_1. Therefore no process among them sends $(v_0, *)$ to p. Therefore, at most $n - r + 1$ processes may send $(v_0, *)$ to p, contradicting the fact that p receives at least $n - r + 2$ pairs $(v_0, *)$, proving (1).

Now, prove (2). To derive a contradiction, suppose that there is some correct process p' such that $new_{p'}(r') \neq v_0$ with $r' \geq r$. We showed that there is one correct process q that sent $(v_0, *)$ to p in round r. Thus, either its input value equal to v_0 or process q changed its estimate to v_0 in some round before r. By Lemma 3.3.3, $new_s(r') \leq v_0$ for every correct process s and every round $r' \geq r$. Thus, $new_{p'}(r') < v_0$. Similarly, to have current estimate $new_{p'}(r')$, either the input value of p' is equal to $new_{p'}(r')$ or it changed the estimate to a value equal to $new_{p'}(r')$ in some round before r.

- if the input value of p' is equal to $new_{p'}(r')$ or p' changes its estimate in some round before r, then by Lemma 3.3.3 the estimate of every correct process is less than

or equal to $new_{p'}(r')$. But $new_{p'}(r') < v_0$, contradicting the fact that at least one correct process sends $(v_0, *)$ in round $r > 2$.

- if p' changes its estimate in some round $\geq r$ then by Lemma 3.3.4, there is one process that changes its estimate to a value equal to $new_{p'}(r')$ in round $r-1$. But $new_{p'}(r') < v_0$, contradicting (1).

□

Lemma 3.3.7. *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, then for every correct process q, $new_q(1) = v_{min}$. They do not change their estimate again 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 3.3.4, we have a set of processes $\{q_1, \cdots, q_{t-1}\}$ such that q_i changes the value of its variable *new* to v_0 in the round i, $1 \leq i \leq t-1$. Moreover, faulty process p_0 that never changes its estimate is different from all processes in the set $\{q_1, \cdots, q_{t-1}\}$. If all these processes are faulty then we have a set of $t+1$ faulty processes (including p_0, q and $\{q_1, \cdots, q_{t-1}\}$), contradicting the hypothesis that there are at most t faulty processes. Therefore, at least one process in set $\{q_1, \cdots, q_{t-1}\}$ is correct. Let p be such a process. Process p changes its estimate to v_0 in round $1 \leq r < t$: $new_p(r) = v_0$. But for every $1 \leq r < t$, $new_p(r) \leq new_p(t)$. Thus, $v_0 = new_p(r) \leq new_p(t) = c_{max}$.

□

Lemma 3.3.8. *If two processes decide at Line 18 then they decide the same value.*

Proof. Suppose that process p decides v_0 at Line 18 and process q decides v_1 at Line 18. 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 $(n-t)+(n-t) > n$, we get $v_0 = v_1$. □

Proposition 3.3.1. *No two processes decide different values.*

Proof. Let $r > 1$ be the first round during which some process decides. There are two cases to consider.

CASE 1: $r \leq t$. Some process p decides v at Line 15 (it receives $n-r+2$ pairs $(v, *)$ in round r). By Lemma 3.3.6, we have:

(1) no process changes its estimate to a value less than v since round $r-1$,

(2) for every correct process s and every round $r_1 \geq r$, $new_s(r_1) = v$.

To derive a contradiction, suppose that some process q decides $v' \neq v$ in round r'.

- if q decides v' at round $r' \geq r$ at Line 15. Thus, q receives at least $n-r'+2$ pairs $(v', *)$ in round r'. As $n-r'+2+n-r+2 > n$, at least one process sends $(v, *)$ to p in round r and sends $(v', *)$ to q in round r'. Thus, $r' > r$ and $v' < v$ because a process changes only its estimate to a value less than its current estimate. By Lemma 3.3.4, some process has changed its estimate to v' in round $r-1$, contradicting (1).

- if q decides at round $t+1$:

  - if q decides at Line 18 then q receives at least $n-t$ pairs $(v', *)$. As $n-t > t$ there is one correct process that sent $(v', *)$ to q. By (2), $v' = v$, contradicting the hypothesis.

  - if q decides at Line 24 then q receives at least $n-t$ pairs and one of them is pair (v', y) such that $v' < y$ in round $t+1$. As $n-t > t$, at least one correct process sent a pair to q. By (2), that pair is $(v, *)$. Thus, $v' \leq v$. But by (1), no process changes its state to a value less than v since round $r-1$, contradicting the fact that some process changed its estimate to v' and sent (v', y) to q.

CASE 2: $r = t + 1$. By Lemma 3.3.5, either:

- At the end of round t, the value of variable *new* of every correct process is the same c_{max}. At round $t + 1$, all correct processes receive at least $n - t$ pairs $(c_{max}, *)$. As no process decide earlier than round $t + 1$, all correct processes decide c_{max} at Line 18. Suppose that a faulty process q decides.

  - If it decides at Line 18 then by Lemma 3.3.8 it decides c_{max}.
  - If it decides at Line 24 then let $x = Min\{v | v \in G_q\}$: it receives at least $n - t$ pairs and one among them is (x, y) such that $x < y$. As q receives at least $n - t$ pairs, at least one pair comes from a correct process. Hence, $x \leq c_{max}$. On the other hand, if pair (x, y) comes from a correct process then $x = c_{max}$ else it comes from a faulty process. By Lemma 3.3.7, we get $x \geq c_{max}$. Thus, in all cases, $x = c_{max}$. That means that q decides c_{max} as correct processes.

- $V[t] = \{v_{min}, c_{max}\}$, there are t faulty processes such that for every faulty process s $new_s(t) = v_{min}$. Moreover, some correct processes changes its estimate to v_{min} in round t. Suppose that it is correct process p and so $new_p(t) = v_{min}$. Thus, at most $n - t - 1$ processes send pairs $(v, *)$ such that $v \neq v_{min}$ in round $t + 1$. It implies that no process cannot decide at Line 18.

  As p changes its estimate to v_{min} in round t, it sends (v_{min}, y) with $v_{min} < y$ to all processes at the beginning of round $t + 1$. Every correct process receives at least $n - t$ pairs (from correct processes)and one of them is (v_{min}, y) such that $v_{min} < y$ (comes from p). Thus every correct process decides v_{min} at Line 24.

  Now, suppose that a faulty process q' decides at Line 24. Then it must receive at least $n - t$ pairs in round $t + 1$. On the other hand, at least $t + 1$ processes have the value *new* equal to v_{min} at the end of round t (including all the t faulty processes and one correct process that changes it estimate to v_{min}). Thus, among these $n - t$

pairs, one of them is $(v_{min}, *)$ and we have $Min\{v|v \in G_{q'}\} = v_{min}$. Therefore, if a faulty process q' decides then it must decide v_{min}.

□

Proposition 3.3.2. *Every correct process decides.*

Proof. By Lemma 3.3.5, we have at the end of round t, either:

- the value *new* of every correct process is the same then all correct processes decide at Lines 18.
- 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 receives at least $n-t$ pairs and one of them is (v_{min}, y) such that $v_{min} < y$ from p. It decides v_{min} at Line 24.

□

Proposition 3.3.3. *if a process decides v then v is the input value of some process.*

Proof. if a process decides, it decides some value in set $V[t]$. By Lemma 3.3.1, $V[t] \subseteq V[1]$. 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. □

Thus, we have:

Theorem 3.3.1. *If processes are numerate, uniform consensus is solvable in $t+1$ rounds if $n > 2t$ even if all processes are anonymous.*

Proposition 3.3.4. ***Early Stopping**: Let us assume that there are at most $f \leq t$ faulty processes. Thus, no process executes more than $Min\{f+3, t+1\}$ rounds. More precisely, no correct process executes more than $Min\{f+2, t+1\}$ rounds and no faulty process executes more than $Min\{f+3, t+1\}$ rounds.*

Proof. Consider only the case where $f \leq t-1$. As we have seen in the proof of Lemma 3.3.5 replacing t by f, we have either *(a)* for every correct process q $new_q(f) = c_{max}$, or *(b)* $V[f] = \{v_{min}, c_{max}\}$, some correct process changes its estimate to v_{min} in round t and for every faulty process s, $new_s(f) = v_{min}$.

- if $new_q(f) = c_{max}$ for every correct process q. At round $f+2$, every correct process receives at least $n-f$ values c_{max} and decides c_{max} and stops at Line 14. At round $f+3$, no faulty process receives enough $n-t$ messages and so stops at Line 13.

- if $V[f] = \{v_{min}, c_{max}\}$ and some correct process changes its state to v_{min} in round f. By Lemma 3.3.3, $new_q(f+1) \leq v_{min}$ for every correct process q. By definition of v_{min}, $new_q(f+1) = v_{min}$ for every correct process q. At round $f+2$, every correct process receives at least $n-f$ values c_{max}, decides c_{max} and stops at Line 14. At round $f+3$, every faulty process does not receive more that $t < n-t$ messages and stops at Line 13.

□

3.3.2 Innumerate processes

Proposition 3.3.5. *Uniform consensus is no 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.

1. In repartition R, all identifiers in I are of only one process, identifiers in $J - \{l\}$ are identifiers of only one process too and identifier l is the identifiers 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.

2. 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 form 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: no processes in $R(J)$ receives message from processes in $R(I)$ and no process in $R(I)$ receives 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 no process in $S(I)$ receives message from $S(J)$ and no process in $S(J)$ receives 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 uniform consensus is solvable when $l > 2t$. The algorithm is very similar to the one presented in Figure 3.2 with only difference at Lines 13, 14, 17 and 19. The quorums $n - t$ and $n - r + 2$ pairs are replaced by $l - t$ and $l - r + 2$, respectively. Then we get:

Theorem 3.3.2. *Uniform consensus is solvable with innumerate processes with general-omission failures if and only $l > 2t$.*

3.4 Conclusion

The results in this chapter show that in crash failure or send omission failure models, the uniform consensus is possible even if processes are fully anonymous and innumerate but in general-omission failure model, the ability of processes to count identical messages they receive or identifiers of processes is required.

Chapter 4

Agreement in Byzantine failure model

In this chapter, we consider the consensus problem in the case of Byzantine failure model. For the synchronous case, $3t + 1$ identifiers are necessary to reach the consensus in a system of n processes, up to t of which can be Byzantine, using a scenario argument. The matching synchronous algorithm is obtained by a simulation that transforms any synchronous Byzantine agreement algorithm designed for a classical system with unique identifiers into one in homonym model with $l > 3t$ identifiers. For the partially synchronous case, we prove using a partitioning argument that the lower bound becomes $l > \frac{n+3t}{2}$. Using this bound, we present a new Byzantine agreement algorithm.

4.1 The Synchronous model

In this section we prove the necessary and sufficient condition for solving agreement in a synchronous system.

4.1.1 Impossibility

We prove that having $l > 3t$ is necessary for solving synchronous Byzantine agreement, regardless of whether the processes are numerate or innumerate, using a scenario argument, in the style of Fischer, Lynch and Merritt [34].

Proposition 4.1.1. *Synchronous Byzantine agreement is unsolvable even with numerate processes if* $l \leq 3t$.

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

- The set A is the set of processes with identifiers from 1 to t
- The set B is the set of processes with identifiers from $t + 1$ to $2t$
- The set C is the set of processes with identifiers from $2t + 1$ to $3t$

Now, we shall define three executions α, β and γ as follows:

- In execution α, for every identifier i, $G(i)$ contains only one process excepts the group $G(3t)$ that contains $n - 3t + 1$ processes. Processes in B are Byzantine. All correct processes have input 1. For Byzantine processes in B, they run as follows: for $t < i < 2t$, Byzantine process with identifier i run $\mathcal{A}_i(0)$. Byzantine process with identifier $2t$ has the same behavior concerning the messages sent to processes in B and C as one correct process with input 0 while it has the same behavior concerning the messages sent to processes in A as $n-3t+1$ processes with identifier $2t$ having input 0. (Here, we use the fact that each Byzantine process can send multiple messages to each correct process in a round.)

- In execution β, for every identifier i, $G(i)$ contains only one process excepts group $G(2t)$ that contains $n-3t+1$ processes. Processes in C are Byzantine. All processes in A have input 1 while processes in B have input 0. For Byzantine processes in C, they run as follows: for $2t < i < 3t$, Byzantine process with identifier i has the same behavior concerning the messages sent to processes in A as one correct process with input 1 while it the same behavior concerning the messages sent to processes in B as one correct process with input 0. Byzantine process with identifier $3t$ has the same behavior concerning the messages sent to processes in B and C as one correct process with input 0 while it has the same behavior concerning the messages sent to processes in A as $n-3t+1$ processes with identifier $3t$ input 1. (Here, we use the fact that each Byzantine process can send multiple messages to each correct process in a round.)

- In execution γ, for every identifier i, $G(i)$ contains only one process excepts group $G(2t)$ that contains $n-3t+1$ processes. Processes in A are Byzantine. All correct processes have input 0. Byzantine process with identifier i in A run $\mathcal{A}_i(1)$.

The correct processes in A cannot distinguish α from β. By the *validity* property, eventually, all correct processes decide 1 in execution α. Therefore, processes in A must also decide 1 in execution β. Thus all correct processes decide 1 in execution β.
The correct processes in B cannot distinguish β from γ. By the *validity* property, eventually, all correct processes decide 0 in execution γ. Therefore, processes in B must also decide in execution β. Thus all correct processes decide 0 in execution β, contradicting the previous paragraph.

□

4.1.2 Algorithm

Next, we present an algorithm that solves Byzantine agreement assuming $l > 3t$. 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 $l > 3t$, e.g., [47]). The principle of the transformation is as follows: processes are divided into groups according to their identifiers. Each group simulates a single process. For the Byzantine failure model, a group may be a correct group, a partially Byzantine group or a Byzantine group (see the Chapter 2). All processes within correct group can reach agreement and cooperatively simulate a single correct process while a partially Byzantine group or Byzantine group may simulate a Byzantine process. The correctness of our simulation relies on the fact that more than two-thirds of the simulated processes will be correct (since $l > 3t$), which is enough to achieve agreement.

Proposition 4.1.2. *Synchronous Byzantine agreement is solvable even with innumerate processes if $l > 3t$.*

Proof. We transform any Byzantine agreement algorithm $\mathcal{A}$ for the classical model with unique identifiers into an algorithm $\mathcal{T}(A)$ for systems with homonyms. Consider any such $\mathcal{A}$ for a system with l processes $\{p_1, \ldots, p_l\}$ that terminates after a finite number of rounds. $\mathcal{A}$ can be specified by:

(1) a set of local process states,

(2) a function $init(i, v)$ that encodes the initial state of process p_i when p_i has input value v,

(3) a function $M(s, r)$ that determines the message to send in state s in round r,

(4) a transition function $\delta(s, r, RM)$ that determines the new state to which the process moves from state s after receiving a set of messages RM in round r, and

Code for processes with identifier i

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

2 **for all** r from 1 to k

SUPERROUND r

3 /* SELECTION ROUND $2r$ */

4 send $\langle s \rangle$ to all processes

5 receive $\langle RM \rangle$

6 $s =$ deterministic choice of some element $x.val$ such that $x \in RM$ and $x.id = i$

7 /* RUNNING ROUND $2r + 1$ */

8 send $\langle M(s, r) \rangle$ to all processes /* almost identical to original algorithm */

9 receive $\langle RM \rangle$

10 **for all** j in $\mathcal{L}$ /* eliminate messages from known Byzantine groups */

11 **if** there is more than one different message from identifier j in RM

12 **then** remove all of them from RM

13 $s = \delta(s, r, RM)$

SUPERROUND $k + 1$

14 send $\langle decide(s) \rangle$ to all processes

15 receive $\langle RM \rangle$

16 **if** there is a $v \neq \perp$ such that $|\{d \in RM : d.val = v\}| > t$

17 **then** decide such a v

18 **else** decide $\perp$

Figure 4.1: Synchronous Byzantine agreement algorithm $\mathcal{T}(\mathcal{A})$ with n processes and l identifiers.

(5) a decision function $decide(s)$ which is the decision in state s, or $\perp$ if there is no decision yet (once a correct process has decided in a state s, $decide(s')$ remains equal to this decision in all states s' reachable from s).

The computation proceeds in superrounds. Superround r is composed of the two rounds $2r$ and $2r + 1$. Let k be the number of superrounds of $\mathcal{A}$. In each superround r, where $1 \leq r \leq k$, each process sends $\langle M(s, r) \rangle$ to all processes. From the set $\langle RM \rangle$ that it receives, it moves to the new state basing on the transition function $\delta(s, r, RM)$. At the end of superround $k + 1$ [1], the function $decide(s)$ is executed to return decision value.

Our new algorithm $\mathcal{T}(\mathcal{A})$ is shown in Figure 4.1.

In rounds $2r$ called selection round (line 4 to 6) of superround r, the processes within

[1] this final superround is composed of one round

each group try to agree on a state for superround r. We will show that in each round, the selected state will be the same for the processes in correct group.

In rounds $2r + 1$ called running round (line 8 to 13), each process simulates one step of algorithm $\mathcal{A}$ with the state chosen in the preceding selection round and the messages received in the round.

Finally, in supperound $k + 1$(line 14 to 17), if there is a value decided by at least $t + 1$ processes with different identifiers then the process can decide that value (since at least one of the processes that decided the value must be correct). The superround $k + 1$ is useful for correct processes that belong to a group with a Byzantine process.

Let α_H be an execution of $\mathcal{T}(\mathcal{A})$. We first observe that in the selection 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 selection round. Every process in the group then makes the same deterministic choice for the new value of s in line 6. Let $winner_i^r$ be a process in $G(i)$ that sent a message containing s_i^r in the selection 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 all rounds r, p_i's state s has value s_i^r at the beginning of round r in α.

We construct the execution α inductively. In the first selection 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 the input to process p_i in α be v. 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 r 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 $G(i)$, we let p_i send the message specified by $\mathcal{A}$ to all other processes. By the hypothesis, p_i is in state s_i^r at the beginning of round r, so this message will be $M(s_i^r, r)$. For each incorrect group $G(i)$, we let p_i 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 RM from a process with identifier j at the end of the running round of r of α_H (if any). (There is at most one such message after $winner_i^{r+1}$ has executed line 12.)

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 superround r of α_H as p_i receives in r of α. (Below, we denote this common set by RM_i^r.) If $G(j)$ is correct, all processes in $G(j)$ send $M(s_j^r, r)$ to $winner_i^{r+1}$ in the running round of superround r. Since processes are innumerate, $winner_i^{r+1}$ will only receive a single copy of $M(s_j^r, r)$ 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 labeled with identifier i at the end of the running round of superround 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 superround $r+1$, $winner_i^{r+1}$ sends $\delta(s_i^r, r, RM_i^r)$ 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 $\delta(s_i^r, r, RM_i^r)$. This guarantees property 2 holds for $r+1$.

This completes the inductive construction of execution α satisfying property 1 and property 2. Now, we prove the correctness of the algorithm:

Agreement property: As $\mathcal{A}$ is a synchronous Byzantine agreement algorithm that tolerates t Byzantine failures, all correct processes eventually decide some value v in α. It follows from property 2 above that in α_H, eventually for all correct groups $G(i)$, s_i^{k+1} is

a state where $decide(s_i^{k+1})$ is v. As $l > 3t$, at least $t+1$ groups $G(i)$ are correct and all processes in these groups eventually send v in the supperround $k+1$. Thus, each correct process in α_H eventually decides v, even if it is in a group with a Byzantine process.

Validity property: If all correct processes in α_H have the same input value v then by property 1 all correct processes in α have input value v. By validity of $\mathcal{A}$, all correct processes eventually decide some value v in α. Like the proof of correctness of **Agreement** property, all correct processes in α_H eventually output v.

Termination property: all correct processes decide in line 18 or 19 in superround $k+1$.

□

From Proposition 4.1.1 and Proposition 4.1.2, we have the following theorem.

Theorem 4.1.1. *Synchronous Byzantine agreement is solvable if and only if* $l > 3t$.

4.2 The Partially Synchronous model

4.2.1 Impossibility

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

Proposition 4.2.1. *Partially synchronous Byzantine agreement is unsolvable even with numerate processes if* $l \leq \frac{n+3t}{2}$.

Proof. Byzantine agreement is impossible when $l \leq 3t$, even in the fully synchronous model, by Proposition 4.1.1. So, it remains to show that Byzantine agreement is impossible when $l > 3t$ and $l \leq \frac{n+3t}{2}$. To derive a contradiction, assume that there exists a Byzantine

agreement algorithm $\mathcal{A}$ for such a system. Let $\mathcal{A}_i(v)$ be the algorithm executed by a process with identifier i when it has input value v. Let $x = l - 3t$, $y = n - (2l - 3t)$. We divide the set $\mathcal{L} = \{1, 2, ..., l\}$ into 4 subsets:

- $\mathcal{L}_0 = \{1, ..., t\}$
- $\mathcal{L}_1 = \{t + 1, ..., 2t\}$
- $\mathcal{L}_2 = \{2t + 1, ..., 3t\}$
- $\mathcal{L}_3 = \{3t + 1, ..., l\}$

We consider an initial configuration as follows: all groups $G(i)$ contains only one process except the group $G(3t)$ that contains $1+x+y$ processes.(Note that as $(l-1)+(1+x+y) = (l-1)+1+l-3t+(n-(2l-3t)) = n$, there are always n processes in the system.)
We define the executions of this algorithm, α, β as follows:
In execution α, processes in $\mathcal{L}_0$ are Byzantine and send no messages. All other processes have input 1 and all of the messages sent are delivered. By the *validity* property of algorithm, all correct processes decide 1 by some round r_0.
In execution β, processes in $\mathcal{L}_1$ are Byzantine and send no messages. All other processes have input 0 and all of the messages sent are delivered. By the *validity* property of algorithm, all correct processes decide 0 by some round r_1.
Now, we define a configuration that satisfies the following properties:
(1) For each identifier $i \leq 3t - 1$, the group $G(i)$ contains only one process,
(2) $G(3t)$ contains $y + 1$ processes,
(3) For $3t + 1 \leq i \leq l$, G(i) contains two processes, we say that p_i^0 and p_i^1
We define an execution γ as follows: processes in $\mathcal{L}_2$ are Byzantine. Processes in $\mathcal{L}_1$, processes p_i^1 in $\mathcal{L}_3$ and group $G(3t)$ have input 1. Processes in $\mathcal{L}_0$, processes p_i^0 in $\mathcal{L}_3$ have input 0. Messages are not delivered between correct processes that have opposite

input value for the first $r = max(r_0, r_1)$ rounds of γ. All messages sent after round r are delivered. Byzantine processes run as follows: for Byzantine process with identifier i different from $3t$, it runs $\mathcal{A}_i(1)$ for groups of processes with input 1 as in α and runs $\mathcal{A}_i(0)$ for groups of processes with input 0 as in β. Byzantine process with identifier $3t$ has the same behavior concerning the messages sent to processes with input 1 as $x + 1$ processes with identifiers $3t$ in α while it has the same behavior concerning the messages sent to processes with input 0 as $x + y + 1$ processes with identifiers $3t$ in β.

The correct processes in $\mathcal{L}_1$ cannot distinguish α and γ for the first r rounds. Thus, they decide 1 in γ.

The correct processes in $\mathcal{L}_0$ cannot distinguish β and γ for the first r rounds. Thus, they decide 0 in γ, contradicting the agreement property.

□

4.2.2 Algorithm

In this section, we present an algorithm that solves Byzantine agreement in the partially synchronous model when $l > \frac{n+3t}{2}$. Our algorithm is based on the algorithm given by Dwork, Lynch and Stockmeyer [31] for the classical case where $n = l$, with several novel features. First, we begin with an *authenticated broadcast* primitive based on [63] and then use this primitive to implement an agreement algorithm.

4.2.2.1 Authenticated Broadcast

The algorithm is a straightforward generalization of the ones given in [31, 63] for systems with unique identifiers. The execution as being divided into superrounds, where each superround consists of two consecutive rounds. Let T be the first superround such that all messages sent during or after superround T are delivered. Two primitives $Broadcast(m)$ by a process with identifier i and $Accept(m, i)$ are used. 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 $\max(r + 1, T)$.

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

Proposition 4.2.2. *It is possible to implement authenticated broadcasts satisfying the correctness, unforgeability and relay properties in the basic partially synchronous model, provided* $l > 3t$.

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

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

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

4.2.2.2 Agreement algorithm

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

The Byzantine agreement algorithm is shown in Figure 4.2. Whenever a correct process sends a message, it sends it to all processes. The execution of the algorithm is broken into phases, each of which lasts four superrounds. Processes assigned the identifier $(ph \bmod l)+$

1 are the *leaders* of phase *ph*. In each phase, each process first performs a broadcast of a proposal containing the set of values it would be willing to decide (line 8). These are the values in its *proper* set, unless it has already locked a value, as described below, in which case it can only send its locked value. Each phase leader chooses a value that appears in proposals the leader has accepted from $l-t$ different identifiers (if such a value exists) and sends out a request for processes to lock that value (line 12) during superround 2 of the phase. Then, in superround 3 of the phase, all processes vote on which lock message to support, using a broadcast (line 16). In the third superround of the phase, each process that performed accept for votes for a particular value v from $l-t$ different identifiers sends $\langle \text{ack } v \rangle$ back to the leaders (line 20) and locks the value v (by adding the value to its *locks* set, along with the phase number associated with the lock). A leader that receives $l-t$ ack messages for the value it wanted locked can decide that value (line 23). Finally, each process that has decided sends a message to others (line 24); if any process receives such a message with the same decision value from $t+1$ identifiers, it can also decide that value (line 27). At the end of a phase, a process releases old locks (line 31) if it has accepted enough votes for a later lock request.

To cope with homonyms, our algorithm differs from the original algorithm of [31] in the following three ways. (1) The new algorithm uses a set of processes with $l-t$ 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 4.2.1, 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 [31], since each phase in that algorithm has a unique leader. (3) The decide messages are used to ensure that a correct process that shares its identifier with a Byzantine process can eventually decide. (This is similar to the mechanism used in Section 4.1.2.) We begin by

proving the property of quorums used by the algorithm.

Lemma 4.2.1. *Assume $l > \frac{n+3t}{2}$. If A and B are sets of identifiers and $|A| \geq l - t$ and $|B| \geq l - t$, then $A \cap B$ contains an identifier that belongs to only one correct process and no Byzantine processes.*

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

□

In the original algorithm of [31], 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 4.2.2. *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 $\langle \text{ack } v', ph \rangle$. (We may have $p = p'$.) According to line 18, there is a set A of $l - t$ identifiers j for which p performs $Accept(\langle \text{vote } v, ph \rangle, j)$. Similarly, there is a set B of $l - t$ identifiers j for which p' performs $Accept(\langle \text{vote } v', ph \rangle, j)$. By Lemma 4.2.1, $A \cap B$ contains an identifier j that belongs to only one correct process and no Byzantine processes. By unforgeability, the correct process with identifier j performed $Broadcast(\langle \text{vote } v, ph \rangle)$ and $Broadcast(\langle \text{vote } v', ph \rangle)$. Thus, $v = v'$.

□

Lemma 4.2.3. *If two correct processes decide on line 23 in the same phase, then they decide the same value.*

Proof. Suppose two correct processes p and p' decide values v and v', respectively, during some phase ph. Then, process p received $\langle \text{ack } v, ph\rangle$ from $l - t > t$ 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 4.2.2, $v = v'$.

□

Lemma 4.2.4. *Suppose there is a value v and a phase ph such that processes with $l - t$ different identifiers sent an $\langle \text{ack } v, ph\rangle$ message in phase ph. Then, at all times after phase ph, each correct process that sent $\langle \text{ack } v, ph\rangle$ has a pair (v, ph') with $ph' \geq ph$ in its locks set.*

Proof. To derive a contradiction, suppose the claim is false. Let A be the set of $l - t$ identifiers of the processes that sent an $\langle \text{ack } v, ph\rangle$ message in phase ph. Consider the first time the claim is violated: some correct process q that sent an $\langle \text{ack } v, ph\rangle$ message removes its lock on v (on line 31). This means that there is some $v' \neq v$ and $ph' > ph$ such that q has performed $Accept(\langle \text{vote } v', ph'\rangle, j))$ for $l - t > t$ 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 $l - t$ different identifiers j with $v' \in V_j$. Let B be this set of identifiers.

By Lemma 4.2.1, some identifier $j \in A \cap B$ belongs to only one correct process and no Byzantine processes. Let r be the correct process with this identifier j. Since r's identifier is in A, r sent $\langle \text{ack } v, ph\rangle$ in phase ph. Since r's identifier is in B, it follows from unforgeability that r performed a $Broadcast(\langle \text{propose } V_j, ph'\rangle)$ with $v' \in V_j$. According to line 7, this is possible only if $(v, *)$ is not in r's *locks* set at the beginning of phase ph'. This contradicts our assumption that all correct processes that sent an $\langle \text{ack } v, ph\rangle$ message in phase ph (including r) keep the value in v in their *locks* set from the time they execute line 20 of phase ph until they execute line 31 of phase ph'.

□

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

Lemma 4.2.5. ?? *At the end of any phase ph_3 that occurs after T, if (v_1, ph_1) is in the locks variable of a correct process p_1 and (v_2, ph_2) is in the locks variable of a correct process p_2, then $v_1 = v_2$.*

Proof. Since, at the end of phase ph_3, correct processes have locks associated with phases ph_1 and ph_2, we must have $ph_1 \leq ph_3$ and $ph_2 \leq ph_3$. If $ph_1 = ph_2$, then $v_1 = v_2$ by Lemma 4.2.2. So for the rest of the proof assume, without loss of generality, that $ph_2 > ph_1$. Before process p_2 added (v_2, ph_2) to its *locks* set in phase ph_2, it performed $Accept(\langle \text{vote } v_2, ph_2 \rangle, j)$ for $l - t$ different identifiers j. By the relay property of the authenticated broadcasts, p_1 will accept all of these messages by the end of phase ph_3 and remove (v_1, ph_1) from its *locks* set, if $v_1 \neq v_2$. Thus, v_1 must be equal to v_2.

□

Lemma 4.2.6. *Let p be a correct process. Let ph be a phase such that $(ph \bmod l) + 1$ 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 4.2.4, at most one value will be appear in the *locks* variables of correct processes at the end of phase $ph - 1$. We consider two cases.

Case 1: the *locks* set of some correct process q is non-empty at the end of phase $ph - 1$. Let (v, ph_v) be the (unique) entry in the *locks* set of q, where ph_v must be smaller than ph. Then q performed $Accept(\langle \text{vote } v, ph_v \rangle, j$ for $l - t > t$ different identifiers, including some identifier j that does not belong to any Byzantine process. Thus, some correct process q performed $Broadcast(\langle \text{vote } v, ph_v \rangle)$. So, q performed $Accept(\langle \text{propose } V_j, ph_v \rangle, j)$ from

$l - t \geq 2t + 1$ different identifiers j with $v \in V_j$. At least $t + 1$ of those identifiers do not belong to any Byzantine process. So correct processes with at least $t + 1$ different identifiers performed $Broadcast(\langle \text{propose } V_j, ph_v \rangle)$, which means v is in the *proper* set of correct processes with at least $t + 1$ different identifiers at the beginning of phase ph_v. Thus, by the end of phase $ph - 1$, v will be in the *proper* set of every correct process. It follows that, in phase ph, every correct process will perform $Broadcast(\langle \text{propose } V, ph \rangle)$ with $v \in V$, and process p will be able to find a value that it can send in superround 2 of phase ph.

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

□

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

Proposition 4.2.3. *Partially synchronous Byzantine agreement is solvable even with innumerate processes if* $l > \frac{n+3t}{2}$.

Proof. We prove each of the three correctness properties of the algorithm in Figure 4.2 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 $Broadcast(\langle \text{propose } V, * \rangle)$ message only if $V \subseteq \{v_0\}$. It follows from unforgeability that a correct process can perform an $Accept(\langle \text{propose } V, * \rangle, j)$ only if $V \subseteq \{v_0\}$ or a Byzantine process has identifier j. Thus, according to the test on line 15, a correct process can

perform $Broadcast(\langle \text{vote } v, * \rangle)$ only if $v = v_0$. Then, according to the test on line 18, a correct process can send a message $\langle \text{ack } v, * \rangle$ only if $v = v_0$. Then, according to the test on line 21, a correct process can decide v only if $v = v_0$. Thus, no correct process decides a value different from v_0 on line 23. A process can decide a value different from v_0 on line 27 only if at least one correct process has already decided that value on line 23, so no correct process will decide a value different from v_0 on line 27.

Agreement: If no correct processes ever decide, agreement is trivially satisfied. A correct process decides a value v on line 27 only if some correct process has previously decided v. Thus, for agreement, it suffices to prove that all values decided by correct processes on line 23 are identical. So, for the remainder of the proof of agreement, we only consider processes that decide on line 23.

Suppose phase ph_1 is the first phase during which some correct process decides. By Lemma 4.2.3 there is a unique value v_1 that correct processes decide during phase ph_1. Let p be a correct process that decides v_1 during phase ph_1. Then p received $\langle \text{ack } v_1, ph_1 \rangle$ messages from $l - t$ different identifiers. Let A be this set of $l - t$ identifiers.

Suppose some correct process p decides a value v in a phase $ph > ph_1$. We shall prove that $v = v_1$. Process p must have performed $Accept(\langle \text{propose } V_k, ph \rangle, k)$ from $l - t$ different identifiers k with $v \in V_k$. Let B be this set of $l - t$ identifiers. By Lemma 4.2.1, 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 $\langle \text{ack } v_1, ph_1 \rangle$ message in phase ph_1. By Lemma **??**, $(v_1, *)$ is in the *locks* set of q at the beginning of phase ph. Thus, no process with identifier k performs $Broadcast(\langle \text{propose } V_k, ph \rangle)$ unless $V_k \subseteq \{v_1\}$. By unforgeability, no correct process can perform $Accept(\langle \text{propose } V_k, ph \rangle, k)$ unless $V_k \subseteq \{v_1\}$. Since $k \in B$, process p did perform $accept(\langle \text{propose } V_k, ph \rangle, k)$ and $v \in V_k$. Thus, $v = v_1$. This completes the proof of the agreement property.

Termination: First, we show that if p is any correct process that does not share its identi-

fier with any other process, then p terminates. Let ph be a phase such that $(ph \bmod l)+1$ is p's identifier and phase $ph-1$ occurs after T. By Lemma 4.2.6, there is some value v such that p sends a $\langle \text{lock } v, ph\rangle$ message in superround 2 of phase ph. Every correct process receives this message, and no other lock messages are received from a process with identifier $(ph \bmod l)+1$ in this phase. According to the test in line 10, p must have performed $Accept(\langle \text{propose } V_j, ph\rangle, j)$ for $l-t$ different identifiers j with $v \in V_j$ during superround 1 of phase ph. By the relay property, all correct processes must have performed these *Accept* actions by the end of superround 2 of phase ph. Thus, every correct process performs $Broadcast(\langle \text{vote } v, ph\rangle)$ during superround 3 of phase ph and all correct processes accept this broadcast. Thus, all correct processes send $\langle \text{ack } v, ph\rangle$ in round 1 of superround 3 of phase ph. Process p receives all of these messages and decides v.
There are at least $2t+1$ correct processes that do not share their identifier with any other process (since $n \leq 2l-3t$). By the argument above, these will all decide. By the agreement property, they will all decide on the same value v. Eventually, all of these $2t+1$ processes will send $\langle \text{decide } v\rangle$ messages in superround 4 of each phase, and all correct processes will receive these messages and decide on line 27.

□

Combining Proposition 4.2.1 and 4.2.3, and the classical result that Byzantine agreement is impossible when $n \leq 3t$ even if $l = n$, yields the following theorem (for numerate or innumerate processes).

Theorem 4.2.1. *Partially synchronous Byzantine agreement is solvable if and only if* $l > \frac{n+3t}{2}$ *and* $n > 3t$.

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

```
1   locks = ∅
2   ph = 0                                                              /* phase number */
3   proper = {v}                                          /* v is the value proposed by the process */
4   Note: in each round, proper is updated as described on page 53
5   while true
6       /* beginning of superround 1 of phase */
7       let V be the set of values v ∈ proper such that there is no pair (w, ∗) ∈ locks for any w ≠ v
8       Broadcast(⟨proposeV, ph⟩)                                       /* superround 1 */
9       /* beginning of superround 2 of phase */
10      if i = (ph mod l) + 1 and there is some value v_lock such that
11              the process has performed Accept(⟨proposeV_j, ph⟩, j) from l − t different identifiers j with v_lock ∈ V_j
12          then send ⟨lockv_lock, ph⟩ to all processes                  /* round 1 of superround 2 */
13      /* beginning of superround 3 of phase */
14      if there is some value v for which the process received ⟨lockv, ph⟩ from identifier (ph mod l) + 1 and
15              has performed Accept(⟨proposeV_j, ph⟩, j) for l − t different identifiers j with v ∈ V_j
16          then choose one such v and perform Broadcast(⟨votev, ph⟩)   /* superround 3 */
17      /* beginning of superround 4 of phase */
18      if for some v, the process has performed Accept(⟨votev, ph⟩, j) from l − t different identifiers j
19          then  add (v, ph) to locks and remove any other pair (v, ∗) from locks
20                send ⟨ackv, ph⟩ to all processes                      /* round 1 of superround 4 */
21      if i = (ph mod l) + 1 and
22              the process has received ⟨ackv_lock, ph⟩ from l − t different identifiers in this round
23          then decide v_lock (but continue running the algorithm)
24      if the process has already decided some value v
25          then send ⟨decidev⟩ to all processes                        /* round 2 of superround 4 */
26      if for some v, the process has received ⟨decidev⟩ from t + 1 different identifiers j in this round
27          then decide v (but continue running the algorithm)
28      for each (v_1, ph_1) ∈ locks
29          if for some v_2 ≠ v_1 and ph_2 > ph_1, the process has performed Accept(⟨vote v_2, ph_2⟩, j) for l − t
30                  different identifiers j
31              then remove (v_1, ph_1) from locks
32      ph = ph + 1
```

Figure 4.2: Byzantine agreement algorithm for the partially synchronous model.

Chapter 5

Agreement in restricted Byzantine failure model

In this chapter, we consider the agreement in Restricted Byzantine failure model where each Byzantine process can only send at most one message to each recipient in each round. Concerning numerate processes, we show that $t + 1$ identifiers are enough to reach agreement even in a partially synchronous model. We also show that this bound is tight using a valency argument. Note that in Chapter 4, we showed that for Byzantine failures without restriction, $3t + 1$ is required to reach to the agreement for synchronous system et $l > n + 3t/2$ is required for partially synchronous system. Therefore, we can see that if the power of Byzantine process is partially restricted then, we may reduce remarkably the number of identifiers needed to reach agreement.

Concerning innumerate processes, we show that the restriction of the power of Byzantine processes does not help to solve the agreement. In particular, our results imply that, even for $t = 1$, anonymous Byzantine agreement is impossible in the synchronous and partially synchronous models (see the Table 5.1).

	Synchronous	Partially synchronous
Innumerate processes	$l > 3t$	$l > \frac{n+3t}{2}$
Numerate processes	$l > t$	$l > t$

Table 5.1: Necessary and sufficient conditions for solving Byzantine agreement in a system of n processes using l identifiers and tolerating t Byzantine failures. In all cases, n must be greater than $3t$.

5.1 Numerate Processes

First, we consider the model where processes can count copies of identical messages.

5.1.1 Impossibility

Proposition 5.1.1. *Synchronous Byzantine agreement is unsolvable with numerate processes against restricted Byzantine processes if $l \leq t$ or $n \leq 3t$.*

Proof. Classical results show that if $n \leq 3t$ then synchronous consensus is impossible [34]. We show that it is also impossible if $l \leq t$. To derive a contradiction, assume that there exists an algorithm $\mathcal{A}$ that solves Byzantine agreement with $l \leq t$. We consider only executions of $\mathcal{A}$ with some fixed set of l Byzantine processes such that each of them has a distinct identifier.

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 *0-valent* if, starting from C, the only possible decision value that correct processes can have is 0; it is *1-valent* if, starting from C, the only possible decision value that correct processes can have is 1. C is *univalent* if it is either 0-valent or 1-valent.

First, we prove following two lemmata:

Lemma 5.1.1. *There is a bivalent initial configuration.*

Proof. For $0 \leq j \leq n - l$, 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-l} is 1-valent. Therefore, there is j so that C_0^j is 0-valent and C_0^{j+1} is not 1-valent. Only one correct process is in a different state in these two initial configurations. We consider a initial configuration C where this process is Byzantine at the beginning while all other processes are correct. From this configuration, there is an execution where the Byzantine process behaves as a correct process with input 0 and then all correct processes must decide 0. On the other hand, there is also an other execution where the Byzantine process behaves as a correct process with input 1 and then all correct processes must decide 1. Thus, C is bivalent. □

Lemma 5.1.2. *Let C_0 and C_1 be two configurations of $\mathcal{A}$ such that the state of only one correct process is different in C_0 and C_1. Then, there exist executions α_0 and β that start from C_0 and C_1, respectively, which both produce the same output value.*

Proof. Let p be the correct process whose state is different in C_0 and C_1 and let i be the identifier assigned to p. Let s_0 and s_1 be the state of p in C_0 and C_1, respectively. Let b be a Byzantine process that has identifier i.

Let α be the execution from C_0 in which b starts in state s_1 and follows p's algorithm, and all other Byzantine processes send no messages. Let β be the execution from C_1 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 β, since p and b send the same messages in α as b and p send in β. Thus, each correct process other than p must output the same decision in α and β. □

Now, we note C_k the configuration at end of round k. From C_k, the system can reach to different configurations C_{k+1}. In an execution of $\mathcal{A}$, C_{k+1} is completely determined by (1) C_k, (2) the set of correct processes C, and (3) the messages send by the Byzantine

process to the correct processes in round $k+1$. We note a configuration C_k is decision configuration if C_k is bivalent and every configuration C_{k+1} is univalent.

By Lemma 5.1.1 there is an initial configuration C_0. For every execution of $\mathcal{A}$ from this configuration, all correct processes must decide the same value. Therefore, there is an execution of $\mathcal{A}$ in which there a decision configuration C_d, where $d \geq 1$. Then, some configuration C_{d+1} is 0-valent and some successor configuration C'_{d+1} is 1-valent. For $0 \leq j \leq n-l$, let C^j_{d+1} be the successor of C_d where, in round $d+1$, the Byzantine processes send the same messages to the first j correct processes as they do in C'_{d+1}, and send the same messages to the rest of the processes as they do in C_{d+1}. Then, $C^0_{d+1} = C_{d+1}$ is 0-valent and $C^{n-l}_{d+1} = C'_{d+1}$ is 1-valent. Choose j so that C^j_{d+1} is 0-valent and C^{j+1}_{d+1} is 1-valent. Only one correct process is in a different state in these two configurations, so by Lemma 5.1.2, some execution from C^{j+1}_{d+1} decides v. Thus, C^{j+1}_{d+1} is bivalent, contradicting the assumption.

This contradiction completes the proof of Proposition 5.1.1. □

5.1.2 Algorithm

Proposition 5.1.2. *Partially synchronous Byzantine agreement is solvable with numerate processes against restricted Byzantine processes if $l > t$ and $n > 3t$.*

The algorithm used to prove this proposition is similar to the one presented in Section 4.2.2.2. We first introduce in Section 5.1.2.1 a more powerful version of authenticated Broadcast, which can be implemented in systems with numerate processes against restricted Byzantine processes. Then, we use this broadcast to give the Byzantine Agreement algorithm in Section 5.1.2.2.

5.1.2.1 Authenticated Broadcasts with Multiplicities

In this more powerful version of authenticated broadcast, accept actions have an extra parameter 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 the number of correct processes that broadcasts 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: $broadcast(i, m, r)$, where i is the identifier of the broadcaster, m is the message and r is the superround number, and $accept(i, \alpha, m, r)$ where α is a strictly positive integer. α is an estimate of the number of processes with identifier i that broadcast m in superround r.

The authenticated broadcast primitive is specified as follows. Consider any execution that uses authenticated broadcast. 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.

1. *Correctness:* If α ($\alpha > 0$) correct processes with identifier i perform $Broadcast(i, m, r)$ in superround $r \geq T$ then every correct process performs $Accept(i, \alpha', m, r)$ with $\alpha' \geq \alpha$ during superround r.

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

3. *Unforgeability:* If α ($\alpha \geq 0$) correct processes with identifier i perform $Broadcast(i, m, r)$ in superround r and some correct process performs $Accept(i, \alpha', m, r)$ in superround

```
Code for process with identifier i ∈ {1, 2, ..., l}
Variable:
1   a[h, m, r] = 0 for all h, m and r
Main code:
2   for all R from 1 to ∞
3       M = ∅
4       for all h ∈ {1, 2, ..., l}
5           for all m ∈ possible messages
6               for all k ∈ {1, ..., R/2}
7                   if a[h, m, k] ≠ 0 then M = M ∪ {(echo, h, a[h, m, k], m, k)}
8       if R = 2r then
9           To perform Broadcast(i, m, r) : M = M ∪ {(init, i, m, r)}
10      send ⟨M⟩ to all processes
11      Let V be the set of valid messages received in the round
12      for all h ∈ {1, 2, ..., l}
13          for all m ∈ possible messages
14              if (R = 2r) then
15                  a[h, m, r] =number of occurrences of (init, h, m, r) in V
16              for all k ∈ {1, ..., R/2}
17                  Let W be the multiset of (echo, h, ∗, m, k) occurring in V
18                  if |W| ≥ n − 2t then
19                      Let α1 be the max of α such that
20                          there is at least n − 2t messages (echo, h, α', m, k) in W with α' ≥ α
21                      a[h, m, k] = max(α1, a[h, m, k])
22                  if R is odd and |W| ≥ n − t then
23                      Let α2 be the max of α such that
24                          there is at least n − t messages (echo, h, α', m, k) in W with α' ≥ α
25                      Accept(h, α2, m, k)
```

Figure 5.1: Authenticated broadcast primitive for numerate processes and restricted Byzantine processes.

r' then $r \leq r'$ and $\alpha' \leq \alpha + f_i$.

4. *Unicity:* for each i, m and r, each correct process performs at most one $Accept(i, *, m, r)$ per superround.

We give an implementation of authenticated broadcast in Figure 5.1. In the algorithm, we call a message sent by some process with identifier i at round R *valid* if

- it contains at most one tuple $(init, i, m, r)$ and $2r = R$ in that tuple, and

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

- $\alpha > 0$ in each message $(echo, *, \alpha, *, *)$

All messages sent by correct processes are valid.

We now prove that the implementation in Figure 5.1 satisfies the specification of authenticated broadcast when $n > 3t$ and $l > t$.

Let $\Pi_R(h, m, k)$ be the set of correct processes that send a message containing $(echo, h, *, m, k)$ in round R.

Lemma 5.1.3. *If a correct process performs Accept$(*, \alpha, *, *)$ then $\alpha > 0$.*

Proof. To perform *Accept*$(*, \alpha, *, *)$ line 25, correct processes consider only valid messages $(echo, *, \beta, *, *)$ with $\beta > 0$. □

Lemma 5.1.4. *Assume that α ($\alpha > 0$) correct processes with identifier i perform Broadcast(i, m, r) in round $2r$. Let $R \geq 2r+1$ be a round. For every q in $\Pi_R(i, m, r)$, if q sends $(echo, i, a_q, m, r)$ in round R then $a_q \leq \alpha + f_i$.*

Proof. We prove this lemma by induction.

- $R = 2r+1$: Each of the α correct processes with identifier i who performs $Broadcast(i, m, r)$ in superround r sends $(init, i, m, r)$ in round $2r$. Each correct process receives at most $\alpha + f_i$ valid messages containing $(init, i, m, r)$ in round $2r$. Thus, if a process q in $\Pi_{2r+1}(i, m, r)$ sends $(echo, i, a_q, m, r)$ in round $2r + 1$ then $a_q \leq \alpha + f_i$.

- Let $R > 2r + 1$. Assume the lemma is true for round $R - 1$. Let q be a correct process in $\Pi_R(i, m, r)$. In line 21 of round $R - 1$, process q either did not change $a[i, m, r]$ or set it to α_1. If $a[i, m, r]$ did not change, then q sends the same tuple $(echo, i, a_q, m, r)$ that it sent in the previous round, and the claim follows from the

induction hypothesis. Otherwise, suppose q changed $a[i, m, r]$ to α_1 in round $R-1$. Then, q must have received at least $n - 2t > t + 1$ messages containing tuples of the form $(echo, h, \alpha', m, k)$ with $\alpha' \geq \alpha_1$ in the previous round. At least one of those messages was from a correct process, which had $\alpha' \leq \alpha + f_i$ by the induction hypothesis. Thus, $\alpha_1 \leq \alpha + f_i$ and the claim follows.

□

Lemma 5.1.5. *Assume that α correct processes with identifier i perform $Broadcast(i, m, r)$ in round $2r \geq T$. In round $R \geq 2r + 1$, we have:*
(1) if $\alpha > 0$ then $\Pi_R(i, m, r)$ is the set of correct processes,
(2) $\Pi_{R-1}(i, m, r) \subseteq \Pi_R(i, m, r)$, and
(3) for every q in $\Pi_R(i, m, r)$, q sends $(echo, i, a_q, m, k)$ in round R with $a_q \geq \alpha$ in round R.

Proof. We prove this lemma by induction.

- $R = 2r+1$: Each of the α correct processes with identifier i who performs $Broadcast(i, m, r)$ in superround r sends $(init, i, m, r)$ in round $2r$. Since $2r \geq T$, every correct process receives at least α messages containing $(init, i, m, r)$ in round $2r$. From the algorithm, a correct process never sends $(echo, i, *, m, r)$ in round $2r$, so $\Pi_{2r}(i, m, r) = \emptyset$. Thus, we have (2). If $\alpha = 0$, (3) comes from Lemma 5.1.3 and (1) is trivially satisfied. If $\alpha > 0$, every correct process q sends $(echo, i, a_q, m, r)$ in round $2r + 1$ with $a_q \geq \alpha$. Thus, we have (1) and (3).

- Let $R > 2r+1$. Assume properties (1), (2) and (3) are true for round $R-1$. Property (2) follows from the fact that $a[h, m, k]$ never decreases. Property (1) for round R follows from (1) and (2) in the induction hypothesis. To prove (3), consider any process q in $\Pi_R(h, m, r)$. If $\alpha = 0$, (3) comes from Lemma 5.1.3, so assume $\alpha > 0$.

In line 21 of round $R-1$, process q either did not change $a[i,m,r]$ or set it to α_1. If $a[i,m,r]$ did not change, then q sends the same tuple $(echo, i, a_q, m, r)$ that it sent in the previous round, and the claim follows from the induction hypothesis. Otherwise, suppose q changed $a[i,m,r]$ to α_1 in round $R-1$. By properties (1) and (3) of the induction hypothesis each of the $n-t$ correct processes sends $(echo, i, \alpha', m, r)$ with $\alpha' \geq \alpha$ in round $R-1$. Since $R > T$, all of these messages are received by q, so $\alpha_1 \geq \alpha$. Thus, q sends $(echo, i, \alpha_1, m, r)$ with $\alpha_1 \geq \alpha$ in round R.

□

Proposition 5.1.3. *[Unicity] For each i, m and r, each correct process performs at most one accept$(i, *, m, r)$ per superround.*

Proof. This follows directly from the code.

□

Proposition 5.1.4. *[Correctness] If α ($\alpha > 0$) correct processes with identifier i performs Broadcast(i, m, r) in superround $r \geq T$ then every correct process performs Accept(i, α', m, r) with $\alpha' \geq \alpha$ in superround r.*

Proof. Each of the α correct processes with identifier i who performs $Broadcast(i, m, r)$ in superround $r \geq T$ sends $(init, i, m, r)$ in round $2r$. By Lemma 5.1.5, every correct process q sends $(echo, i, \alpha_q, m, r)$ in round $2r+1$, with $\alpha_q \geq \alpha$. All of these messages are delivered. Thus, every correct process will set α_2 to a value greater than or equal to α on line 23 and then performs $Accept(i, \alpha_2, m, r)$ at the end of superround r.

□

Proposition 5.1.5. *[Relay] If a correct process performs accept(i, α, m, r) in superround $r' \geq r$ then every correct process performs accept(i, α', m, r) with $\alpha' \geq \alpha$ in superround* $\max(r', T) + 1$.

Proof. Assume some correct process p performs $Accept(i, \alpha, m, r)$ in superround r'. Then it must do so in round $2r' + 1$ (since a correct process accepts only in the second round of the superround). Process p must have received at least $n - t$ messages containing tuples of the form $(echo, i, \alpha', m, r)$ with $\alpha' \geq \alpha$ in this round. Among the $n - t$ senders of these messages, at least $n - 2t$ are correct. Since the value stored in each sender's $a[i, m, r]$ variable can only increase, each of these $n - 2t$ correct senders also sends a tuple of the form $(echo, i, \alpha', m, r)$ with $\alpha' \geq \alpha$ in round $\max(r', T)$. All of these messages are delivered. Thus, for each correct process, the value of $a[i, m, r]$ is at least α after the process executes line 21 in superround $\max(r', T)$. Then, in superround $\max(r', T) + 1$, each of the $n - t$ correct processes sends a tuple of the form $(echo, i, \alpha', m, r)$ with $\alpha' \geq \alpha$. All of these messages are delivered. Thus, each correct process performs $accept(i, \alpha', m, r)$ with $\alpha' \geq \alpha$ in superround $\max(r', T) + 1$.

□

Proposition 5.1.6. *[Unforgeability] If α ($\alpha \geq 0$) correct processes with identifier i perform $Broadcast(i, m, r)$ in superround r and some correct process performs $Accept(i, \alpha', m, r)$ in superround r' then $r \leq r'$ and $\alpha' \leq \alpha + f_i$.*

Proof. Assume that some correct process q performs $Accept(i, \alpha', m, r)$ in superround r'. Then it received at least $n - t$ messages containing tuples of the form $(echo, h, \alpha'', m, k)$ with $\alpha'' \geq \alpha'$. Because $n - t \geq t + 1$, one of those messages came from a correct process. By Lemma 5.1.4, the α'' in that message is less than or equal to $\alpha + f_i$, so $\alpha' \leq \alpha + f_i$ and $\alpha' \geq 0$ follows directly from the code.

□

From Propositions 5.1.3, 5.1.4, 5.1.5 and 5.1.6, we obtain:

Theorem 5.1.1. *The algorithm in Figure 5.1 implements authenticated broadcast in the partially synchronous model if $l > t$ and $n > 3t$.*

5.1.2.2 Algorithm

The approach used to design the algorithm is similar to the one used in Section 4.2.2 , but various thresholds must be adjusted to take advantage of the restriction on the Byzantine processes, and to make use of the processes' ability to count copies of identical messages. In combination, these two factors allow us to weaken the condition on number of identifiers from $l > n+3t/2$ (which is required in Section 4.2.2) to $l > t$. The safety of our algorithm depends on the condition $n > 3t$, while liveness is guaranteed by the condition $l > t$.

A partially synchronous algorithm for Byzantine agreement when $n > 3t$ and $l > t$ is shown in Figure 5.2. It uses the authenticated broadcast primitive described in the previous subsection and follows the same general pattern as the algorithm of Dwork, Lynch and Stockmeyer [31]. Each iteration of the main loop is called a *phase*, which takes four superrounds.

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

Consider a process p executing the algorithm. During superround r', p may performs $Accept(i, \alpha_i, m, r)$. For each identifier i, α_i is p's estimate of the number of processes with identifier i that performed $Broadcast(i, m, r)$. We say that the number of *witnesses* that p has for (m, r) in superround r' is the sum, over all i, of the α_i's that appear in all $Accept(i, \alpha_i, m, r)$ 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 $Broadcast(*, m, r)$

and exceed that number by at most t.

We now prove the correctness of the algorithm Figure 5.2. Consider an execution of the algorithm. Let $f = \sum_{i=1}^{l} f_i$ be the total number of Byzantine processes in the execution.

Lemma 5.1.6. *If some correct process p has $n-t$ witnesses for (m, r) in some superround $r' \geq r$, then there are at least $n-t-f$ correct processes that performed $Broadcast(*, m, r)$ in round r.*

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

□

Lemma 5.1.7. *If some correct process has $n - t$ witnesses for (m, r) and some correct process has $n - t$ witnesses for (m', r'), then some correct process performed both $Broadcast(*, m, r)$ and $Broadcast(*, m', r')$.*

Proof. By Lemma 5.1.6, there is a set A of at least $n - t - f$ correct processes that performed $Broadcast(*, m, r)$ and there is a set B of at least $n - t - f$ correct processes that performed $Broadcast(*, m', r')$. Since there are $n - f$ correct processes, $|A \cap B| = |A| + |B| - |A \cup B| \geq (n - t - f) + (n - t - f) - (n - f) = n - 2t - f \geq n - 3t > 0$, so there is at least one process in $A \cap B$.

□

Lemma 5.1.8. *If the messages $\langle ack, v, ph\rangle$ and $\langle ack, v', ph\rangle$ are both sent by correct processes, then $v = v'$.*

Proof. Suppose a correct process p sends $\langle \text{ack}, v, ph\rangle$ and a correct process p' sends $\langle \text{ack}, v', ph\rangle$. According to line 20, process p has $n-t$ witnesses for (vote $v, 4ph+2$) in superround 3 of

phase ph. Similarly, p' has $n-t$ witnesses for (vote $v, 4ph+2$). By Lemma 5.1.7, there is at least one correct process that performed $Broadcast(*, \text{vote } v, 4ph+2)$ and performed $Broadcast(*, \text{vote } v', 4ph+2)$ in superround 3 of phase ph, so $v = v'$.

□

Lemma 5.1.9. *If two correct processes decide on line 27 in the same phase then they decide the same value.*

Proof. Suppose two correct processes p and p' decide values v and v', respectively, during some phase ph. Then process p received $n-t$ copies of $\langle \text{ack}, v, ph \rangle$, so some correct process must have sent $\langle \text{ack}, v, ph \rangle$. Similarly, some correct process must have sent $\langle \text{ack}, v', ph \rangle$. By Lemma 5.1.8, $v = v'$.

□

Lemma 5.1.10. *At the end of each phase, the locks set of a correct process contains at most one pair.*

Proof. Let p be a correct process. We first prove that in each phase ph, p can add at most one pair to its *locks* set. For each pair (v, ph) added in phase ph, p has $n-t$ witnesses for (vote $*, 4ph+2$). By Lemma 5.1.7 this condition can be true for at most one value v. When p adds this unique pair (v, ph) to *locks*, it removes all other pairs $(v, *)$. Then in line 28 to 30, p will remove all other pairs (v', ph') from the *locks* set.

□

Lemma 5.1.11. *Suppose that some correct process receives $n-t$ $\langle ack, v, ph \rangle$ messages. Then at all times after phase ph, each correct process that sent $\langle ack, v, ph \rangle$ in phase ph has a pair (v, ph') with $ph' \geq ph$ in its locks set.*

Proof. Let A be the set of correct processes that sent $\langle \text{ack}, v, ph \rangle$ in phase ph. By 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 q that sent an $\langle \text{ack}, v, ph \rangle$ message removes its lock on v (on Line 22). This means that there is some $v' \neq v$ and $ph' > ph$ such that q has $n - t$ witnesses for (vote $v', 4ph' + 2$). By Lemma 5.1.6, some correct process performs $Broadcast(*, \text{vote } v', 4ph' + 2)$. That process has $n - t$ witnesses for (propose $v', 4ph'$) in superround 2 of phase ph'. By Lemma 5.1.6 there is a set B of $n - t - f$ correct processes that perform $Broadcast(*, \text{propose } v', 4ph')$. Since there are $n - f$ correct processes, $|A \cap B| = |A| + |B| - |A \cup B| \geq (n - 2t) + (n - t - f) - (n - f) = n - 3t > 0$. Thus, there is at least one correct process r that sends $\langle \text{ack}, v, ph \rangle$ in phase ph and performs $Broadcast(*, \text{propose } v', 4ph')$ in phase ph'. According to Line 7, this is possible only if $(v, *)$ is not in r's *locks* set at the beginning of phase ph'. This contradicts our assumption that all correct processes that sent an $\langle \text{ack}, v, ph \rangle$ message in phase ph keep the value v in their *locks* set from the time they execute at Line 15 of phase ph until they execute at Line 30 of phase ph'.

□

Lemma 5.1.12. *At the end of any phase ph_3 that occurs after T, if (v_1, ph_1) is in the locks variable of a correct process p_1 and (v_2, ph_2) is in the locks variable of a correct process p_2, then $v_1 = v_2$.*

Proof. Since, at the end of phase ph_3, correct processes have locks associated with phases ph_1 and ph_2, we must have $ph_1 \leq ph_3$ and $ph_2 \leq ph_3$. If $ph_1 = ph_2$ then $v_1 = v_2$ follows from Lemma 5.1.8 (because just after p_i adds (v_i, ph_i) to its *locks* set, it sends $\langle \text{ack}, v_i, ph_i \rangle$). So for the rest of the proof assume, without loss of generality, that $ph_2 > ph_1$. Before process p_2 added (v_2, ph_2) to its *locks* set in phase ph_2, it has $n - t$ witnesses for (vote $v_2, 4ph_2 + 2$). By the relay property of the authenticated broadcast, p_1 will accept all of these messages by the end of phase ph_3 and remove (v_1, ph_1) from its locks set, if $v_1 \neq v_2$. Thus, v_1 must be equal to v_2.

□

Lemma 5.1.13. *Let p be a correct process. Let ph be a phase such that $ph \bmod l + 1$ 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 8.3.2, at most one value will appear in the *locks* variables of correct processes at the end of phase $ph - 1$. We consider two cases.

Case 1: the *locks* set of some correct process q is non-empty at the end of phase $ph - 1$. Let (v, ph_v) be the entry in the *locks* set of q, where ph_v must be smaller than ph. Then q has $n-t$ witnesses for (vote $v, 4ph_v + 2$) in superround 3 of phase ph_v. Thus, some correct process performed $Broadcast(*,$ vote $v, 4ph_v + 2)$. That process must have $n - t \geq 2t + 1$ witnesses for (propose $v, 4ph_v$). By Lemma 5.1.6, at least $t + 1$ different correct processes performed $Broadcast(*, \text{propose } v, 4ph_v)$, 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 $Broadcast(*, \text{propose } v, 4ph)$, and process p will be able to find a value that it can send in superround 2 of phase ph.

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

□

Proposition 5.1.7 (Validity)**.** *If all correct processes propose v then no correct process decides a value different from v.*

Proof. Suppose all correct processes have the same input value v_0. Then no correct process ever adds any other value to its *proper* set. So, a correct process can perform

$Broadcast(*, \text{propose } v, *)$ only if $v = v_0$. Thus, according to the test on Line 15, a correct process can perform $Broadcast(*, \text{vote } v, *)$ only if $v = v_0$. Then, according to the test on Line 20, a correct process can send a message $\langle \text{ack}, v, * \rangle$ only if $v = v_0$. Thus, no correct process decides a value different from v_0 on Line 27.

□

Proposition 5.1.8 (Agreement)**.** *If two correct processes decide v and v' then $v = v'$.*

Proof. If phase ph_1 is the first phase during which some correct process p decides. By Lemma 5.1.9, 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 5.1.6, there is a set B of $n-t-f$ correct processes that perform $Broadcast(*, \text{propose } v_2, 4ph_2)$. Thus, there is some correct process $h \in A \cap B$ that has sent an $\langle \text{ack}, v_1, ph_1 \rangle$ in phase ph_1.

By Lemma 5.1.11 and 5.1.10, $(v_1, *)$ is in the lock set of h at the beginning of phase ph_2 and this is the only pair in the lock set. Thus, h performs only $Broadcast(*, \text{propose } v_1, 4ph_2)$ in superround 1 of phase ph. But $h \in B$, so it performs $Broadcast(*, \text{propose } v_2, 4ph_2)$. Thus, $v_1 = v_2$

□

Proposition 5.1.9 (Termination)**.** *All correct processes decide.*

Proof. As there are $l > t$ 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 $ph \bmod l + 1 = k$ and phase $ph - 1$ occurs after T. By Lemma 5.1.13, 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 11, p_j must have $n - t$ witnesses for (propose $v_j, 4ph$) during superround 1 of phase ph. By the

relay property, all correct processes must have $n - t$ witnesses for (propose $v_j, 4ph$) at the end of superround 2 of phase ph.

Each correct process receives the same set of lock messages from all processes with identifier k and deterministically chooses one of them. Let v be the value chosen by all correct processes. All correct processes then perform $Broadcast(*, \text{vote } v, 4ph + 2)$. Thus, every correct process has $n - t$ witnesses for (vote $v, 4ph + 2$) in superround 3 and according to the test at Line 20, 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.

□

From Proposition 5.1.1 and 5.1.2, above. We prove the following two theorems for this model.

Theorem 5.1.2. *Synchronous Byzantine agreement is solvable with numerate processes against restricted Byzantine processes if and only if $l > t$ and $n > 3t$.*

Theorem 5.1.3. *Partially synchronous Byzantine agreement is solvable with numerate processes against restricted Byzantine processes if and only if $l > t$ and $n > 3t$.*

5.2 Innumerate Processes

We consider the case of innumerate processes.

Theorem 5.2.1. *Synchronous Byzantine agreement is solvable with innumerate processes against restricted Byzantine processes if and only if $l > 3t$.*

Proof. The synchronous algorithm given in Section 4.1.2 obviously still works if the Byzantine processes are restricted.

To derive a contradiction, assume that for some l and t with $l \leq 3t$, 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 $\mathcal{A}_i$ 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 $\{q_0, q_1, ..., q_{l-1}\}$ be these processes. Let q_i run algorithm $\mathcal{A}_i$. We prove that this will solve Byzantine agreement in S. This contradicts the classical impossibility result [31, 47], since the number of processes is $l \leq 3t$. 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 α_S has ł Byzantine processes, the properties of Byzantine agreement are vacuously satisfied. Assume that α_S has $b < l$ 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 $(n - l + 1)$ processes have the identifier 0, and the other processes have identifiers $1, \ldots, l-1$ (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 $(b + 1 \leq i \leq l - 1)$ is correct and has as input $Input(i)$, and

3. all processes with identifier 0 are correct and have input $Input(0)$.

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

The process q_i in α_S and a process with identifier i in α_H have the same state at the beginning of each round. As α_H satisfies the specification of Byzantine agreement, the execution α_S satisfies the specification of the Byzantine agreement. This completes the proof.

□

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

Theorem 5.2.2. *Partially synchronous Byzantine agreement is solvable with innumerate processes against restricted Byzantine processes if and only if* $l > \frac{n+3t}{2}$ *and* $n > 3t$.

Proof. The partially synchronous algorithm if $l > \frac{n+3t}{2}$ and $n > 3t$ given in Section 4.2.2 obviously still works if the Byzantine processes are restricted. The impossibility result can be proved in exactly the same way as in Section 4.2.1. □

5.3 Conclusion

This chapter showed results concerning to necessary and sufficient conditions to solve agreement in restricted Byzantine failure model. Results give us a good way to reduce the number of identifiers needed to the agreement. In next chapter, we shall present an other way to reduce the number of identifiers when processes have more knowledge of the system.

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

```
1   locks = ∅ ;
2   ph = 0;
3   proper = {v}
4   /* v is the input value for the process (see page 72 for how this variable is updated each round) */
5   while true
6       /* beginning of superround 1 of phase */
7       Let V be the set of values v ∈ proper such that
8           there is no pair (w, ∗) ∈ locks for any w ≠ v
9       for each v ∈ V do Broadcast(i, propose v, 4ph)
10      /* beginning of superround 2 of phase */
11      if i = ph mod l + 1
12          and there is some value v such that there are at least n − t witnesses for (propose v, 4ph)
13          then send ⟨lock, v, ph⟩ to all processes /* round 1 of superround 2 */
14      /* beginning of superround 3 of phase*/
15      if there is some value v
16          for which the process received ⟨lock, v, ph⟩ from processes with identifier ph mod l + 1
17              and there are at least n − t witnesses for (propose v, 4ph)
18          then choose deterministically one such value v and perform Broadcast(i, vote v, 4ph + 2)
19      /* beginning of superround 4 of phase */
20      if for some v, there are at least n − t witnesses for (vote v, 4ph + 2)
21          then
22              add (v, ph) to locks and remove any other pair (v, ∗) from locks
23              send ⟨ack, v, ph⟩ to all processes /* round 1 of superround 4 */
24      if for some v:
25              There are at least n − t witnesses for (propose v, 4ph) and
26              received n − t messages ⟨ack, v, ph⟩ in this round
27          then decide v (but continue running the algorithm)
28      for each (v1, ph1) ∈ locks
29          if for some v2 ≠ v1 and ph2 > ph1, there are n − t witnesses for (vote v2, 4ph2 + 2)
30              then remove (v1, ph1) from locks
31      ph = ph + 1
```

Figure 5.2: Partially Synchronous Byzantine agreement algorithm with n processes and l identifiers.

Chapter 6

Homonymous model with the distribution of identifiers available

6.1 Introduction

We consider here an homonymous model where processes not only know the set of identifiers but also how many processes share the same identifier. To get a better understanding of our results let us consider the following example. Assume we have a set of n clients and a set of l servers. Each server has its own authenticated identifier (coming for example from some authenticated digital signature system) and communication between servers is reliable and authenticated. Each client is assigned to exactly one server, but the identity of the clients cannot be verified. Each client communicates with each other but only through the server at which it is associated. At most t of the clients may be Byzantine and try to act as adversaries of the system and may fool their associated server. Then assume we have to administrate such a system and we have the choice of the number of clients we assign to each server, we call such a choice a *distribution of the identifiers* and it is in fact a partition of n into l parts. Are some distributions among servers better than

others? Is it possible to get necessary and sufficient conditions on distributions to ensure the solvability of Byzantine Agreement? Does the knowledge of the distribution of clients among servers improve the solvability of the problem? We give an answer to all these questions. We prove that:

1. For each distribution of the identifiers we give a necessary and sufficient condition that enable us to solve the Byzantine agreement.

2. For n and l we define a distribution D_{max} that is the best of all in the sense that there is a distribution enabling to solve the Byzantine agreement if and only if D_{max} enables us to solve the Byzantine agreement.

3. For n, l and t there exists a distribution of identifiers enabling to solve the Byzantine agreement if and only if $n > 3t$, $l > \frac{(n-r)t}{n-t-min(t,r)}$ where $r = n \bmod l$.

Our result can imply that the Byzantine agreement is possible if $l = t + 1$ and $\lfloor \frac{n}{t+1} \rfloor > t$. This result is to compared with the result of chapter 4 that $l > 3t$ is required to reach to the Byzantine agreement for synchronous system.

Roadmap Section 6.2 contains several definitions. Proofs of impossibility are in Section 7.2. Algorithms for Byzantine agreement are in Section 6.4. In the Section 6.5 we give the main results of this paper and present some consequences. Section 6.6 concludes this section.

6.2 Definitions

In the section Model and Definitions, we considered the definition of Distribution of identifiers and the adversary. We continue here consider the notions of solving problem and agreement coefficient.

6.2.1 Solving problem

We always assume that each process p knows its own identifier $Id(p)$, the distribution D and t the maximal number of Byzantine processes.

Informally a run of algorithm $\mathcal{A}$ for distribution D and failure pattern F for at most t Byzantine processes is a run of $\mathcal{A}$ for which the distribution of identifiers is given by D and Byzantine processes are given by failure pattern F. Then, we say that algorithm $\mathcal{A}$ *solves problem* $\mathcal{P}$ *for distribution* D *and adversary* Ad if and only if all runs of $\mathcal{A}$ for D and any failure pattern of Ad satisfy the specification of $\mathcal{P}$. By extension $\mathcal{A}$ *solves problem* $\mathcal{P}$ *for distribution* D if and only if it solves $\mathcal{P}$ for D and adversary $\mathcal{F}_t(D)$ (*i.e.* all failure patterns with at most t Byzantine processes).

6.2.2 Agreement coefficient

In this section we give some definitions and properties of distributions and adversaries.

Consider distribution $D = \langle n_1, \ldots, n_l \rangle$, the *index* of a distribution counts up to t the number of groups of processes that contain more than one process. For example for the distribution $D = \langle 5, 5, 3, 2, 1, 1 \rangle$ and $t = 5$, we have $index(D) = 4$ but if $t = 3$ then $index(D)$ is equal to t. More precisely:

Definition 6.2.1. $index(D) = |\{i | 1 \leq i \leq t \wedge n_i \geq 2\}|$.

By convention if for all i, $n_i < 2$, $index(D)$=0.

Let $F = \langle t_1, \ldots, t_l \rangle$ be a failure pattern.

Definition 6.2.2. $c_i(F, D)$ *the* agreement coefficient of group $G(i)$ *for failure pattern* F *is:*

$$c_i(F, D) = \begin{cases} n_i, & \text{if } G(i) \text{ is correct.} \\ 1, & \text{if } G(i) \text{ is partially byzantine.} \\ 0, & \text{if } G(i) \text{ is fully Byzantine.} \end{cases}$$

By extension the *agreement coefficient* of failure pattern F for distribution D is $c(F, D) = \sum_{i=1}^{i=l} c_i(F, D)$.

F_t is a failure pattern of special interest in which there is at most one Byzantine process by group and Byzantine processes are in the biggest groups:

Definition 6.2.3. $F_t = \langle t_1 = 1, \ldots, t_t = 1, t_{t+1} = 0, \ldots, t_l = 0 \rangle$ *if* $l > t$ *and* $F_t = \langle t_1 = 1, \ldots, t_t = 1 \rangle$ *if* $l \leq t$

For this failure pattern we get directly a relation between its agreement coefficient and the index.

Proposition 6.2.1. $c(F_t, D) = \sum_{i=t+1}^{i=l} n_i + index(D)$.

Proof. For $1 \leq i \leq index(D)$, each group $G(i)$ contains one Byzantine process and at least one correct process then $\sum_{1 \leq i \leq index(D)} c_i(F_t, D) = index(D)$. For $index(D) < i \leq \min(l, t)$, each group $G(i)$ contains only one process and this process is Byzantine then $\sum_{index(D) < i \leq \min(l,t)} c_i(F_t, D) = 0$. For $\min(l, t) < i \leq l$, each group $G(i)$ is correct then $c_i(F_t) = n_i$. Summing up we get $c(F_t, D) = index(D) + 0 + \sum_{i=t+1}^{i=l} n_i$. □

This failure pattern minimizes the agreement coefficient:

Proposition 6.2.2. *If* $l > t$*, for any failure pattern* F*,* $c(F, D) \geq c(F_t, D)$.

Proof. Let $F = \langle t_1, t_2, \ldots, t_l \rangle$ be any failure pattern. Consider the transformation from F to F' defined by the following rules:

(a) If for some identifier i, $t_i = n_i > 1$ in F, as $l > t$, there is an identifier j such that $t_j = 0$. Then F' is identical to F except that in F', $t_i = n_i - 1$ and $t_j = 1$.

(b) If for some identifiers i and j, $n_i = t_i = 1$ and $n_j > 1$ and $t_j = 0$ in F, then the failure pattern F' is identical to F except that in F', $t_i = 0$ and $t_j = 1$.

(c) If for some identifiers i and $j > i$, $t_j = 1$ and $t_i = 0$ in F, then the failure pattern F' is identical to F except that in F', $t_i = 1$ and $t_j = 0$.

It is easy to verify that for each rule, $c(F, D) \geq c(F', D)$. A repetitive application of rule (a), (b) and (c) to any F leads to F_t, hence by induction $c(F, D) \geq c(F_t, D)$. □

6.3 Impossibility result

In this section, we prove that for a given distribution D, if there exists a failure pattern F of $\mathcal{F}(D)$ such that $c(F, D) \leq 2t$ then there is no algorithm solving Byzantine agreement for this distribution D.

Directly from [47] there is no algorithm solving Byzantine agreement if $n \leq 3t$, moreover, if $l \leq t$ then all groups $G(i)$ may be fully or partially Byzantine and in this case it is easy to verify by a valency argument as in [22] that there is no algorithm solving Byzantine agreement:

Proposition 6.3.1. *If $l \leq t$ or $n \leq 3t$ there is no algorithm solving Byzantine agreement.*

Hence in the following we always assume $l > t$ and $n > 3t$. We get the main impossibility result:

Proposition 6.3.2. *Let D be a distribution, if there exists a failure pattern F of $\mathcal{F}_t(D)$ such that $c(F, D) \leq 2t$ then there is no algorithm solving Byzantine agreement for distribution D and at most t Byzantine processes.*

Proof. By contradiction assume $\mathcal{A}$ is an algorithm solving the Byzantine agreement for the distribution D and a failure pattern F of $\mathcal{F}_t(D)$ such that $c(F, D) \leq 2t$.

In particular $\mathcal{A}$ solves Byzantine agreement for $c(F_t, D)$ and by Proposition 6.2.2, $c(F_t, D) \leq c(F, D)$, then we have $c(F_t, D) \leq 2t$.

The proof is based on a partition argument using essentially failure pattern F_t. The partition is given by the following Lemma:

Lemma 6.3.1. *If $l > t$ and $c(F_t, D) \leq 2t$, there is a partition of $\mathcal{L}$ into two sets B and C such that the number of processes in C is less than or equal to t and all these processes are correct and either:*

(a) B contains only partially Byzantine groups or fully Byzantine groups and $|B| \leq t$

or

(b) all correct groups are singletons and if t_p is the number of partially Byzantine groups in B, t_f is the number of fully Byzantine groups in B and t_c is the number of correct groups in B, then (i) $t_p + t_f = t$, (ii) $t_c \leq t_f$, and (iii) $t_p > 0$.

Proof. Consider the following cases:

- $index(D) = t$ or $l \leq 2t$: then let C be the set $\{t+1, \ldots, l\}$. C contains only correct processes. Let $B = \mathcal{L} - C$, as $l > t$ neither C nor B are empty, proving that B and C are a partition of $\mathcal{L}$. Clearly condition (a) is ensured.

  Consider now the number of processes in C. By Proposition 6.2.1 and the hypothesis,

  $$\sum_{i \in C} n_i \leq c(F_t, D) - index(D) \leq 2t - index(D)$$

  If $index(D) = t$, we get directly $\sum_{i \in C} n_i \leq t$. If $index(D) < t$, C contains only singletons then $\sum_{i \in C} n_i = l - t$, as we assume that $l \leq 2t$ we get again $\sum_{i \in C} n_i \leq t$.

- $index(D) < t$ and $l > 2t$: in this case all correct groups are singletons. Let B be the set $\{1, \ldots, l-t\}$ and C be the set $\{l-t+1, \ldots, l\}$. As before as $l > t$, neither B nor C are empty. Moreover all groups in C are singletons and are correct, then

the number of processes in C is less than or equal to t and all these processes are correct.

Let t_p the number of partially Byzantine groups for B, t_f be the number of fully Byzantine groups for B, and t_c the number of correct groups for B.

As $index(D) < t$, group $G(i)$ is partially Byzantine if and only if $i \leq index(D)$ then $t_p = index(D)$. In the same way group $G(i)$ is fully Byzantine if and only if $index(D) < i \leq t$, then $t_f + t_p = t$ proving (i).

As t_c and C contain only singletons, $\sum_{i=t+1}^{i=l} n_i = t_c + t$. From Proposition 6.2.1 and hypothesis $c(F_t, D) \leq 2t$:

$$c(F_t, D) = index(D) + \sum_{i=t+1}^{i=l} n_i = t_p + t_c + t \leq 2t$$

and then $t_c \leq t - t_p = t_f$ proving (ii).

By contradiction assume that $t_p = 0$ then all groups are singletons and the total number of processes is $t_f + t_c + t = 2t + t_c \leq 2t + t = 3t$ a contradiction proving (iii).

□

Hence following the Lemma, we can partition the set of identifiers into two sets C and B. There are two cases to consider:

Case (a) of the Lemma: The set C contains all the correct groups and $\sum_{i \in C} n_i \leq t$, B and C being a partition of $\mathcal{L}$, $B \neq \emptyset$ and $C \neq \emptyset$. Set B contains only partially Byzantine groups and fully Byzantine groups and $|B| \leq t$.

As $n > 3t$, there are at least $2t + 1$ processes in B and thus there are at least t correct processes in B.

We now define three executions of the algorithm $\mathcal{A}$:

1. In execution α, the failure pattern is F_t. All correct processes have input 1. For every identifier i in B, the Byzantine process in group $G(i)$ has the same behavior concerning the messages sent to processes in C as $n_i - 1$ correct processes with input 0. (Here, we use the fact that a Byzantine process can send multiple messages to each correct process in a single round and therefore a process cannot distinguish the messages coming from a Byzantine process with identifier i from those coming from $n_i - 1$ processes with identifier i). If $n_i = 1$ then the Byzantine process with identifier i sends nothing to processes in C. By the validity property, all the correct processes decide 1.

2. In execution β, all the processes with identifier in C are Byzantine and all the other ones are correct. All the correct processes have input 0. Each Byzantine process behaves as a correct process having input value 1. By validity, all the correct processes decide 0.

3. In execution γ, the failure pattern is F_t. All the processes in C have 1 as input. All the correct processes in B have 0 as input. For every identifier i of B the Byzantine process in group $G(i)$ has the same behavior concerning messages sent to process in C as $n_i - 1$ correct processes with input 1, and the same behavior concerning the messages sent to processes in B as a correct process with input 0. As before, if $n_i = 1$ then the Byzantine process with identifier i sends nothing to processes in C.

Consider failure pattern F_t. The correct processes in C cannot distinguish γ from α and they decide 1 as in execution α. But the correct processes in B cannot distinguish γ from β. Thus, in execution γ, they decide 0 as in execution β. Contradicting the agreement property.

Case (b) of the Lemma: With failure pattern F_t, set C consists of t correct singleton groups and set B contains t_p partially Byzantine groups, t_f fully Byzantine groups and t_c

correct groups such that in B all correct groups are singletons, $t_f + t_p = t$, $t_c \leq t_f$ and $t_p > 0$.

Let C_1 be the set of the t (correct) processes in C and let B_1 be the set of t Byzantine processes in F_t. As every correct or partially Byzantine group contains at least one correct process, define B_2 as a set of $t_c + t_p \leq t$ processes in $B \setminus B_1$ and finally let B_3 be the set of processes in partially Byzantine groups of B not in $B_1 \cup B_2$.

Remark that $B_1 \cup B_2 \cup B_3 \cup C_1 = \mathcal{L}$. All these sets being pairwise disjoint we have $|B_3| = n - |B_1| - |B_2| - |C_1|$ then as $|C_1| = t$, $|B_1| = t$, $|B_2| \leq t$ and $n > 3t$ we deduce $|B_3| \geq 1$, proving that B_3 is not empty.

As in the previous case, we construct three executions:

1. In execution α the failure pattern is F_t and the processes of B_1 are Byzantine processes. Each Byzantine process with identifier i sends the same messages to the processes in C_1 as $(n_i - 1)$ processes with input 0. All the other processes have input 1. By validity, all the correct processes must decide 1.

2. In execution β, all the processes of C_1 are Byzantine. A Byzantine process with identifier i has the same behavior as in execution α where it has input 1. All the other processes have input 0. By validity, all the correct processes decide 0.

3. In execution γ the failure pattern is F_t and the processes in B_2 are Byzantine. A Byzantine process with identifier i sends the same message to the processes in C_1 as $(n_i - 1)$ processes with input 1 while it sends to other processes the same messages as in execution β where it has input 0. All the processes of C_1 have input 1. All the other processes have input 0.

The correct processes in C_1 cannot distinguish execution γ from execution α, and they decide 1 as in α. The correct processes in B_3 cannot distinguish execution γ from execution β, and they decide 0 as in β. Contradicting the agreement property. □

6.4 Byzantine Agreement

In this section we propose an algorithm for Byzantine agreement for any adversary Ad such that for all $F \in Ad$ we have $c(F, D) > 2t$. We proceed in two steps. In the first one, we define an authenticated broadcast primitive whose the specification and the code are given in Section 6.4.1. In the second one, we give in Section 6.4.2 an algorithm solving the Byzantine agreement using this authenticated broadcast primitive.

6.4.1 Authenticated broadcast primitive

Our authenticated broadcast is derived from the authenticated broadcast of [63] defined in the classical case where each process has a different identifier ($n = l$).

The synchronous computation proceeds in superrounds. A superround is composed of two rounds. More precisely the superround r is composed of the synchronous round $2r$ and $2r + 1$. In order to ensure that a message $m = (i, v, r)$ will be accepted in the superround r, all the processes of the group i have to invoke propose (i, v, r) in the superround r.

More precisely, our authenticated broadcast is defined by two primitives: $Propose(i, v, r)$ and $Accept(i, v, r)$, where i is the identifier of the group that proposes value v and r is the superround number in which the message has been proposed. For message $m = (i, v, r)$, when a process p performs $Propose(m)$ we say that process p *proposes* m, in the same way, when a process p performs $Accept(m)$ we say that process p *accepts* m. The authenticated broadcast primitive is specified as follows:

1. *Correctness:* If all processes in correct group $G(i)$ propose (i, v, r) in superround r then every correct process accepts (i, v, r) during superround r.

2. *Relay:* If some correct process accepts (i, v, r) during superround $r' \geq r$ then every correct process accepts (i, v, r) by superround $r' + 1$.

3. *Unforgeability:* If some correct process accepts (i, v, r) in superround $r' \geq r$ then all correct processes with identifier i have proposed (i, v, r) in superround r.

We describe below the authenticated broadcast algorithm in the synchronous model for the adversary Ad such that for all failure patterns F of Ad, $c(F, D) > 2t$. The code of this algorithm is presented in Figure 6.1.

Recall first the principles of the algorithm of [63]. To propose a value v in superround r, process p sends message $(init, p, v, r)$ to all processes (including itself). A process receiving such a message becomes a "witness" to (p, v, r) and sends a message of type *echo* to all processes. Any process that has $t + 1$ witnesses for (p, v, r) becomes itself witness. Once a process receives strictly more than $(2t + 1)$ witnesses, it accepts (p, v, r).

A generalization of the authenticated broadcast to our model is not straightforward. The key solution is still to estimate the number of witnesses of messages but in a more intricate way:

- We have as before two types of messages: *init* messages and *echo* messages. To propose a value v, a process with identifier i sends an *init* message. More precisely to propose v in superround r, this process sends $(init, i, v, r)$. If a process becomes a witness of $m = (i, v, r)$ then it sends in all rounds greater than $2r$, $(echo, i, v, r)$.

  In every round, every correct process sends a message and this message is in fact a (possibly empty) set of *echo* or *init* messages. If group $G(j)$ is correct, then all processes receive exactly n_j messages from this group.

- Differently from the classical model, in our model, to become a witness for some message $m = (i, v, r)$ in round $2r$, a process must receive this messages from *all* processes of the group $G(i)$ namely exactly n_i messages $(init, i, v, r)$ from $G(i)$ (Lines 17−18). In this case, the process becomes a witness of $m = (i, v, r)$ and sends $(echo, i, v, r)$ in the following rounds.

- Later a process may become witness for message $m = (i, v, r)$ in round $R > 2r$ if it receives enough messages from witnesses of m in round R. But as a Byzantine process may act as witness and sends many $(echo, i, v, r)$ messages, the counting of the $(echo, i, v, r)$ messages gives only an estimate of the number of witnesses of message m.

  More precisely, $Rec[i]$ is the set of messages from identifier i received by process p. $Rec[i]$ is a multiset of messages: each element of $Rec[i]$ is the set of messages coming from one of the processes with identifier i. If this set of messages coming from a process contains (1) twice or more identical messages or (2) messages $(*, *, *, r)$ with $2r$ greater than the round number, we knows that it comes from a Byzantine process and we remove this set of messages (Line 12).

  Basically p counts (Line 20) for each $m = (j, v, x)$ the number of $(echo, j, v, x)$ it receives from group $G(i)$.

  If $|Rec[i]| \neq n_i$ then group $G(i)$ cannot be a correct group and due to Byzantine processes in the group, the number of *echo* messages for message m is no really significant.

  $P[i, m]$, the number of positive witnesses for $m = (j, v, r)$ coming from group $G(i)$ is defined by:

$$P[i,m] = \begin{cases} \alpha, & \text{if } |Rec(i)| = n_i \text{ and} \\ & Rec(i) \text{ contains } \alpha \text{ messages } (echo, j, v, r) \\ 1, & \text{if } |Rec(i)| \neq n_i \text{ and} \\ & Rec(i) \text{ contains at least one } (echo, j, v, r) \\ 0, & \text{if } Rec(i) \text{ does not contain } (echo, j, v, r). \end{cases}$$

  The *number of witnesses for* $m = (j, v, r)$ *in round* R is $\sum_{i \in \mathcal{L}} P[i, m]$. If this sum is

greater than t, p becomes witness of m (Line 28).

- In the same way $N[i,m]$ is the number of negative witnesses for $m=(j,v,r)$ coming from group $G(i)$:

$$N[i,m] = \begin{cases} \beta, & \text{if } |Rec(i)| = n_i \text{ and} \\ & \quad Rec(i) \text{ contains } n_i - \beta \text{ messages } (echo, j, v, r). \\ 1, & \text{if } |Rec(i)| \neq n_i. \end{cases}$$

  The *number of negative witnesses for* $m=(j,v,r)$ *in round* R is $\sum_{i\in\mathcal{L}} N[i,m]$. If this sum is less than or equal to t, m is accepted (Line 29).

We now prove the correctness of our algorithm. Consider a distribution D and a failure pattern $F=(t_1,\ldots,t_l)$ such that $\sum_{i=1}^{i=l} t_i \leq t$ and $c(F,D) > 2t$.
Begin with two simple facts:

Fact 6.4.1. *$\xi = |\{i|\ G(i)$ is partially Byzantine $\}|$ and $\mathcal{C}$ be the set of correct groups. Then $\sum_{i\in\mathcal{C}} n_i > 2t - \xi$.*

Fact 6.4.2. *If group $G(i)$ is correct, for each message m the sum of the number of positive and negative witnesses is n_i.*

Introduce some notations for the number of negative and positive witnesses for messages: given $m=(i,v,r)$, let $N_p^R(m)$ be $\sum_{j\in\mathcal{L}} N_p[j,m]$ for process p in round R and let $P_p^R(m)$ be $\sum_{j\in\mathcal{L}} P[j,m]$ for process p in round R. In the following, X_p^R for a variable X denotes the value of variable X for process p in round R. When p or R are clear from the context we omit them.

Lemma 6.4.1. *For every message $m=(i,v,r)$, for all rounds $R > 2r$, for every correct process p and q, if $N_p^R(m) \leq t$ then $P_q^R(m) \geq t+1$.*

```
// code for process p with identifier Id(p)
Initialization:
1   M = ∅                                                    // messages to send
2   V = ∅                                                    // messages to be proposed
To propose m:
3   V = V ∪ {m}
Main code:
4   for R = 0, ..., ∞                                        //R: round number
5       if (R = 0 mod 2) then                                // superround r = R/2
6           P = ∅                          // P: set of proposed messages for the superround
7           forall v ∈ V do P = P ∪ {(init, Id(p), v, R/2)}
8           V = ∅
9           send(M ∪ P)
10      else send (M)
11      Receive all messages for round R
12      let Rec[i] the multiset of all well-formed messages from identifier i
13      Msg = ⋃_{i∈L} {(j, v, x) | (echo, j, v, x) ∈ Rec[i]}
                                          //P: positive witnesses, N: negative witnesses
14      let P, N be integer arrays indexed by identifier and element of Msg
15      forall (i, m) ∈ L × Msg do P[i, m] = N[i, m] = 0
16      forall i ∈ L do
17          if (|Rec[i]| = n_i) ∧ (|{(init, i, v, R/2) ∈ Rec[i]}| = n_i) then
18                  M = M ∪ {(echo, i, v, R/2)}                          //direct witness
19          forall m = (j, v, x) ∈ Msg do
20              c = |{(echo, j, v, x) ∈ Rec[i]}|              //number of echo for (j, v, x)
21              if |Rec[i]| = n_i then                        //G(i) is perhaps a correct group
22                  P[i, m] = c
23                  N[i, m] = n_i − c
24              else                                          //G(i) is not a correct group
25                  P[i, m] = min(1, c)
26                  N[i, m] = 1
27      forall m = (i, v, x) ∈ Msg do
28          if Σ_{i∈L} P[i, m] > t then M = M ∪ {(echo, i, v, x)}                //witness
29          if Σ_{i∈L} N[i, m] ≤ t then Accept(i, v, x)
```

Figure 6.1: Authenticated broadcast for n processes and l identifiers.

Proof. We divide the set $\mathcal{L}$ of identifiers into 3 disjoint subsets: $\mathcal{L} = \mathcal{L}_c \cup \mathcal{L}_{pb} \cup \mathcal{L}_{fb}$, where $\mathcal{L}_c$ is the set of identifiers of correct groups, $\mathcal{L}_{pb}$ is the set of identifiers of partially

Byzantine groups, and $\mathcal{L}_{fb}$ is the set of identifiers fully Byzantine groups.

Let $m = (i, v, r)$ be a message and R be a round with $R > 2r$, assume $N_p^R(m) \leq t$. Consider first messages coming from correct groups and let j be in $\mathcal{L}_c$. By Fact 6.4.2, if group $G(j)$ is correct, we have $N_p[j, m] = n_j - P_p[j, m]$. Then

$$N_p^R(m) = \sum_{j \in \mathcal{L}_c} (n_j - P_p[j, m]) + \sum_{j \in \mathcal{L}_{pb} \cup \mathcal{L}_{fb}} N_p[j, m] \leq t$$

This implies that:

$$\sum_{j \in \mathcal{L}_c} P_p[j, m] \geq \sum_{j \in \mathcal{L}_c} n_j + \sum_{j \in \mathcal{L}_{pb} \cup \mathcal{L}_{fb}} N_p[j, m] - t$$

By Fact 6.4.1, $\sum_{j \in \mathcal{L}_c} n_j \geq 2t + 1 - |\mathcal{L}_{pb}|$, then:

$$\sum_{j \in \mathcal{L}_c} P_p[j, m] \geq t + 1 + \sum_{j \in \mathcal{L}_{pb} \cup \mathcal{L}_{fb}} N_p[j, m] - |\mathcal{L}_{pb}| \tag{6.1}$$

Consider messages coming from partially Byzantine groups and let j be in $\mathcal{L}_{pb}$. In the algorithm p distinguishes between groups that may be correct and groups that may not be correct. Then let $\mathcal{G}ood_p = \{j \in \mathcal{L}_{pb} : |Rec(j)| = n_j\}$ and $\mathcal{B}ad_p = \{j \in \mathcal{L}_{pb} : |Rec(j)| \neq n_j\}$, clearly $\mathcal{L}_{pb}$ is the disjoint union of $\mathcal{G}ood_p$ and $\mathcal{B}ad_p$. By definition, $N_p[j, m] = 1$ if $j \in \mathcal{B}ad_p$, then:

$$\sum_{j \in \mathcal{B}ad_p} N_p[j, m] = |\mathcal{B}ad_p| \tag{6.2}$$

From (6.1) and (6.2) we have:

$$\begin{aligned}\sum_{j\in\mathcal{L}_c} P_p[j,m] &\geq t+1+\sum_{j\in\mathcal{L}_{pb}} N_p[j,m]-|\mathcal{L}_{pb}| \\ &\geq t+1+|\mathcal{B}ad_p|+\sum_{j\in\mathcal{G}ood_p} N_p[j,m]-|\mathcal{L}_{pb}| \\ &= t+1-|\mathcal{G}ood_p|+\sum_{j\in\mathcal{G}ood_p} N_p[j,m]\end{aligned} \tag{6.3}$$

Now consider correct process q and $P_q[j,m]$. If $G(j)$ is a correct group, p and q receive exactly the same messages from this group then $P_p[j,m]=P_q[j,m]$. Then we have:

$$\begin{aligned}P_q^R(m) &\geq \sum_{j\in\mathcal{L}_c} P_q[j,m]+\sum_{j\in\mathcal{L}_{pb}} P_q[j,m] \\ &\geq \sum_{j\in\mathcal{L}_c} P_p[j,m]+\sum_{j\in\mathcal{G}ood_p} P_q[j,m]\end{aligned} \tag{6.4}$$

From (6.3) and (6.4), the process q has

$$P_q^R(m)\geq t+1+\sum_{j\in\mathcal{G}ood_p} N_p[j,m]+\sum_{j\in\mathcal{G}ood_p} P_q[j,m]-|\mathcal{G}ood_p|$$

.

If $j\in\mathcal{G}ood_p$, then $G(j)$ contains at least one correct process: either $P_q[j,m]\geq 1$ or $N_p[j,m]\geq 1$. Therefore, for all identifiers j in $\mathcal{G}ood_p$, $P_q[j,m]+N_p[j,m]\geq 1$. Then we have:

$$P_q^R(m)\geq t+1+\sum_{j\in\mathcal{G}ood_p} N_p[j,m]+\sum_{j\in\mathcal{G}ood_p} P_q[j,m]-|\mathcal{G}ood_p|\geq t+1$$

Proving that q has at least $t+1$ witnesses for (i,v,r) in round R. □

Lemma 6.4.2. *If in some round $R>2r$, a correct process p has at least $t+1$ witnesses for $m=(i,v,r)$ ($P_p^R(m)\geq t+1$) then at least one correct process sends message $(echo,i,v,r)$*

in round R.

Proof. By contradiction, assume no correct process sends message $(echo, i, v, r)$ in round R. Prove first that $P[j, m] \leq t_j$ for every identifier j. For this consider the following two cases:

- If $P[j, m] = 1$ it is sufficient to prove that $t_j \geq 1$, that is there is at least one Byzantine process in group $G(j)$. As set $Rec_p(j)$ contains at least one message $(echo, i, v, r)$ and we assume no correct process sends message $(echo, i, v, r)$ in round R, there is at least one Byzantine process in $G(j)$ that sent message $(echo, i, v, r)$ to p.

- If $P[j, m] > 1$ then, by definition of $P[j, m]$, $|Rec_p(j)| = n_j$ for process p. Among these n_j messages, as all correct processes send messages in each round, at least $n_j - t_j$ messages are sent by correct processes, and by hypothesis, the $P[j, m]$ messages $(echo, i, v, r)$ are not sent by correct processes, we get $P[j, m] + n_j - t_j \leq n_j$ proving $P[j, m] \leq t_j$.

Thus, for every identifier j, $P[j, m] \leq t_j$. By hypothesis, process p gets at least $t + 1$ witnesses for $m = (i, v, r)$ in round R, hence:

$$t + 1 \leq \sum_{j \in \mathcal{L}} P[j, m] \leq \sum_{j \in \mathcal{L}} t_j \leq t$$

a contradiction. □

Lemma 6.4.3. *For any message $m = (i, v, r)$, if all correct processes send $(echo, i, v, r)$ in round $R > 2r$ then all correct processes accept m in round R.*

Proof. Consider a correct process p and a message $m = (i, v, r)$. Let j be an identifier. There are two cases:

- If $Rec_p[j] \neq n_j$ then by definition $N_p[j,m] = 1$ and by Fact 6.4.2 there is at least one Byzantine process in group $G(j)$. Then $1 = N_p[j,m] \leq t_j$.

- $Rec_p[j] = n_j$. Since every correct process sends $(echo, i, v, r)$ in round R, process p receives at least $(n_j - t_j)$ messages $(echo, i, v, r)$ from processes of $G(j)$ and then $P_p[j,m] \geq (n_j - t_j)$. Thus $N_p[j,m] \leq t_j$.

In all cases, $N_p[j,m] \leq t_j$ for all j in $\mathcal{L}$. Summing up, we get $N_p^R(m) = \sum_{j \in \mathcal{L}} t_j \leq t$. Then, every correct process accepts m during round R. □

We now prove that the algorithm of Figure 6.1 satisfies the specification of authenticated broadcast primitive.

Proposition 6.4.1. *(Correctness) If all the processes in correct group $G(i)$ propose (i, v, r) in supperround r then every correct process accepts (i, v, r) in superround r.*

Proof. Assume all processes of correct group $G(i)$ propose $m = (i, v, r)$ in superround r. They send $(init, i, v, r)$ in round $2r$ and every correct process receives n_i messages $(init, i, v, r)$ in round $2r$ and adds $(echo, i, v, r)$ to the list of messages to send in next rounds (Line 18). In round $2r + 1$, every correct process sends $(echo, i, v, r)$ in Line 9 or Line 10.

By Lemma 6.4.3, every correct process accepts m during round $2r+1$ *i.e* during superround r. □

Proposition 6.4.2. *(Relay) If a correct process accepts (i, v, r) during superround $r' \geq r$ then every correct process accepts (i, m, r) by superround $r' + 1$.*

Proof. Suppose that some correct process p accepts $m = (i, v, r)$ during superround r'. Then it accepts m during round R with $R = 2r'$ or $R = 2r' + 1$.

In round R we have $N_p^R(m) \leq t$. By Lemma 6.4.1, for every correct process q, $P_q^R(m) \geq t+1$ then q adds $(echo, i, v, r)$ to the list of messages to send in next rounds (Line 28) and

sends $(echo, i, m, r)$ in round $R + 1$ (Line 9 or Line 10). By Lemma 6.4.3, every correct process accepts m in round $R + 1$. Moreover as $R = 2r'$ or $R = 2r' + 1$, every correct process accepts m by superround $r' + 1$. □

Proposition 6.4.3. *(Unforgeability) If some correct process accepts (i, v, r) in superround $r' \geq r$ then all correct processes with identifier i have proposed (i, v, r) in superround r.*

Proof. Assume some correct process p accepts $m = (i, v, r)$ in the round R with $R = 2r'$ or $R = 2r' + 1$. In round R, $N_p^R(m) \leq t$. From Lemma 6.4.1, for every correct process q, $P_q^R(m) \geq t + 1$ then from Lemma 6.4.2 at least one correct process sends $(echo, i, v, r)$ in round R.

Let p be the first correct process sending $(echo, i, m, r)$ in some round $R_1 \leq R$. There are two cases:

- process p becomes witness in Line 17: p received from $G(i)$ n_i messages $(init, i, v, r)$ in round $2r$. Then all correct processes in $G(i)$ send $(init, i, v, r)$ in this round, and so all correct processes in $G(i)$ have proposed (i, v, r).

- process p becomes witness in Line 28 because $\sum_{j \in \mathcal{L}} P_p^{R_1 - 1}[j, m] \geq t+1$. From Lemma 6.4.2, at least one correct process, say q, sends message $(echo, i, v, r)$ in this round $R_1 - 1$, but then q sent this message before p contradicting the definition of p.

Then if some correct process accepts $m = (i, v, r)$ in superround $r' \geq r$ then all correct processes with identifier i propose m in superround r.

□

From the three previous propositions:

Theorem 6.4.1. *If $l > t$ and $c(F, D) > 2t$, algorithm of Figure 6.1 satisfies the correctness, relay and unforgeability properties of the authenticated broadcast.*

6.4.2 Byzantine Agreement algorithm

In Figure 6.2, we present an algorithm that implements a Byzantine agreement in the synchronous system using authenticated broadcast primitive as specified in Section 6.4.1. We restrict ourselves to binary agreement, the extension to general agreement is standard (e.g. [50]).

The algorithm proceeds in synchronous superrounds as defined for authenticated broadcast in Section 6.4.1 using the *Propose* and *Accept* primitives. Value 1 is the only value that may be proposed by processes. If this value is accepted by enough groups of processes then the correct processes decide 1 else they decide 0.

More precisely, if the input value of a process p is 1, it proposes 1 in superround 1 (more exactly it proposes message $(id(p), 1, 1)$). If the input value is 0, the process does not propose anything. In a next superround r, processes may support value 1 (variable *support* equals to $true$) and propose message $(id(p), 1, r)$.

A process may become a supporter for value 1 only in odd superrounds. To become supporter for value 1 in supperround $2r + 1$, a process must (1) have accepted messages $(i, 1, 1)$ from at least $t + 1$ identifiers and (2) have accepted $r - 1$ messages $(i, 1, u)$ coming from different identifiers one for each u between 1 and $r - 1$.

A process may decide 1 only in even superrounds $2r$ as soon as it has (i) accepted messages $(i, 1, 1)$ from at least $t + 1$ identifiers and (ii) accepted r messages $(i, 1, u)$ coming from different identifiers one for each u between 1 and r.

If a process does not decide 1 at the end of superround $2t + 2$ it decides 0.

We now show that the algorithm Figure 6.2 satisfies the specification of Byzantine agreement. As before, V_p^r denotes the value of variable V for process p in superround r. Moreover, A_p^r is the value of variable A after its assignment in superround r (Lines 6, 9 and 21)

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

Variables:

1 $input = \{v\}$ // *v is the value proposed by the process*
2 $value = 0$
3 $support = false$

Main code:

SUPERROUND 1

4 $r = 1$
5 **if** $input = 1$ **then** $Propose(i, 1, 1)$
6 $A = \{h | h \in \mathcal{L}$ such that $(h, 1, 1)$ is accepted$\}$
7 **if** $\sum_{j \in A} n_j \geq t + 1$ **then** $support = true$

SUPERROUND r

8 **for** $r = 2$ **to** $2t + 1$
9 $A = \{h | h \in \mathcal{L}$ such that $(h, 1, 1)$ is accepted$\}$
10 **if** $r = 0 \bmod 2$
11 **then** //*r even*
12 **if** $support = true$ **then**
13 $Propose(i, 1, r)$
14 $support = false$
15 **if** (α) $\sum_{j \in A} n_j \geq t + 1$ and (β) there exist $(i_1, \ldots, i_{\frac{r}{2}})$ different identifiers such that $(i_1, 1, 2), \ldots, (i_j, 1, 2j), \ldots, (i_{\frac{r}{2}}, 1, r)$ are accepted.
16 **then** $value = 1$
17 **else** // *r odd*
18 **if** (α) $\sum_{j \in A} n_j \geq t + 1$ and (β) there exist $(i_1, \ldots, i_{\frac{r-1}{2}})$ different identifiers such that $(i_1, 1, 2), \ldots, (i_j, 1, 2j), \ldots, (i_{\frac{r-1}{2}}, 1, r - 1)$ are accepted.
19 **then** $support = true$

SUPERROUND $2t + 2$

20 $r = 2t + 2$
21 $A = \{h | h \in \mathcal{L}$ such that $(h, 1, 1)$ is accepted$\}$
22 **if** $support = true$ **then** $Propose(i, 1, 2t + 2)$
23 **if** (α) $\sum_{j \in A} n_j \geq t + 1$ and (β) there exist $(i_1, \ldots, i_{\frac{r}{2}})$ different identifiers such that $(i_1, 1, 2), \ldots, (i_j, 1, 2j), \ldots, (i_{\frac{r}{2}}, 1, r)$ are accepted.
24 **then** $value = 1$
25 **if** $value = 1$
26 **then** DECIDE 1
27 **else** DECIDE 0

Figure 6.2: Synchronous Byzantine Agreement algorithm with distribution $(n_1, ..., n_l)$ and at most t faulty processes.

Directly from the definitions and Fact 6.4.1:

Fact 6.4.3. *If $c(F, D) > 2t$ then the number of processes of all the correct groups is greater than t.*

Proposition 6.4.4. *(Validity) If all correct processes have the same initial valuev, then no value different from v can be decided by any correct process.*

Proof. Assume all correct processes have initial value 1, then in superround 1 all correct processes p propose $(id(p), 1, 1)$. The *correctness* of the authenticated broadcast ensures that every correct process p accepts $(i, 1, 1)$ for each correct group $G(i)$ in superround 1. Then, the set A_p^1 of identifiers j such that p accepted $(j, 1, 1)$(Line 6) contains at least the identifiers of all correct groups. Hence, using Fact 6.4.3, the number of processes with identifier in A_p^1 is greater than t, then every correct process becomes supporter for value 1 (sets *support* to *true*) in superround 1 (Line 7).

In superround 2, as *support* = *true* for all correct processes, every correct process p proposes $(Id(p), 1, 2)$. Again, the *correctness* of the authenticated broadcast ensures that every correct process p accepts $(i, 1, 2)$ for each correct group i in superround 2. As $l > t$ there is at least one correct group, then (1) p accepts $(i, 1, 2)$ for at least one identifier i. As $A_p^1 \subseteq A_p^2$, (2) the number of processes with identifier in A_p^2 is greater than t too. From (1) and (2), every correct process sets *value* to 1 in Line 16 and will decide 1 in Line 26.

Assume now that all correct processes propose 0. By contradiction, assume that some correct process p decides 1. Before deciding, this process sets *value* to 1 in Line 15 in some superround r. But in this superround condition Line 16 for p implies $\sum_{j \in A_p^r} n_j \geq t + 1$. As there is at most t Byzantine processes, there is at least one correct process q with identifier in A_p^r. By *unforgeability* property of authenticated broadcast, this process proposes $(Id(q), 1, 1)$ in superround 1. Contradicting the hypothesis that no correct process proposes 1. Thus, every correct process keeps *value* to 0 and decides 0. □

Proposition 6.4.5. *(Termination) All correct processes decide.*

Proof. A correct process decides at superround $2t + 2$ Line 26 or Line 27. □

Before proving Agreement property, we prove some preliminary properties of any execution.

Lemma 6.4.4. *If a correct process gets the condition* $\sum_{j \in A^x} n_j \geq t + 1$ *in superround* x *then at each superround* y *such that* $x < y \leq 2t + 2$ *each correct process gets the condition* $\sum_{j \in A^y} n_j \geq t + 1$.

Proof. Let p be a correct process and let j be some identifier in A_p^x. By definition of A_p^x, p accepted $(j, 1, 1)$ by superround x. By *relay* property, every correct process accepts $(j, 1, 1)$ by superround y with $x < y \leq 2t + 2$. Thus, for every correct process q, j is in A_q^y. It follows that $A_p^x \subseteq A_q^y$ at every correct process and then $\sum_{j \in A_q^y} n_j \geq t + 1$. □

Lemma 6.4.5. *If* $G(i)$ *is correct and every process of* $G(i)$ *proposes* $(i, 1, 2x)$ *in superround* $2x$ *then every correct process sets value to 1 by superround* $2x$.

Proof. By *correctness* property, (a) every correct process accepts $(i, 1, 2x)$ in superround $2x$.

Since a process q in $G(i)$ proposes $(i, 1, 2x)$, it becomes supporter for value 1 in superround $2x - 1 \geq 1$. It becomes supporter for value 1 either:

- in superround $x = 1$ in Line 7: q finds in superround 1 that $\sum_{j \in A_q^1} n_j \geq t + 1$. By Lemma 6.4.4, for every correct process q' we have $\sum_{j \in A_{q'}^2} n_j \geq t + 1$ ensuring in superround 2 part (α) of condition in Line 16. Part (β) of this condition is ensured by (a). Then every correct process sets *value* to 1 by superround 2.
- or in superround $2x - 1 > 1$ in Line 19: q finds the condition Line 18 true.

From part (α) of this condition and by Lemma 6.4.4, for every correct process q' we have $\sum_{j \in A_{q'}^{2x}} n_j \geq t+1$, then in superround $2x$, part (α) of condition in Line 16 is ensured.

As part (β) of condition in Line 18 is verified for q in superround $2x-1$, q has accepted, by superround $2x-1$, $x-1$ messages $(j'_1, 1, 2), \ldots, (j'_u, 1, 2u), \ldots, (j'_{x-1}, 1, 2(x-1))$ such that all the identifiers are distinct and different from i. By *relay* property, all these messages will be accepted by all correct processes by superround $2x$. Then with (a), every correct process accepts, by superround $2x$, x messages $(j'_1, 1, 2), \ldots (j'_u, 1, 2u), \ldots, (j'_{x-1}, 1, 2(x-1)), (i, 1, 2x)$ such that all the identifiers are distinct ensuring part (β) of condition in Line 16. Then every correct process sets *value* to 1 by superround $2x$.

□

Now, assume that, in the execution, some correct process sets *value* to 1 Line 19 or Line 24, and decides 1. Let r be the first superround in which some correct process p sets *value* to 1. As correct process sets *value* to 1 only in even superround, we assume that $r = 2k$.

Lemma 6.4.6. *If $r \leq 2t$, then every correct process sets value to 1 by superround $r+2$.*

Proof. Before setting *value* to 1 in superround $r = 2k$ p evaluates condition of the test in Line 15. By part (α) of this condition we have (i) $\sum_{j \in A^{2k}} n_j \geq t+1$. By part ($\beta$) of this condition, p has accepted, by superround $2k$, k messages $(j_1, 1, 2), \ldots (j_u, 1, 2u), \ldots, (j_k, 1, 2k)$ with distinct identifiers. Let L be this set of identifiers i.e. $L = \{j_1, \ldots j_k\}$.
As $l > t$, there is at least one correct group $G(h)$, we consider two cases:

- **Case $h \notin L$:** by Lemma 6.4.4 and (i), for every correct process q, we have (ii) $\sum_{j \in A_q^{2k+1}} n_j \geq t+1$ and in particular part (α) of the condition of Line 19 in superround $2k+1$ is satisfied for processes of identifier h. By *relay* property, we have (iii)

for all j_u in L, every correct process accepts $(j_u, 1, 2u)$ by superround $2k+1$ then in particular part (β) of the condition of Line 19 in superround $2k+1$ is satisfied for processes with identifier h.

Then every correct process with identifier h proposes $(h, 1, 2k+2)$ in superround $2k+2$. As group $G(h)$ is correct, by *correctness* property, all correct processes accept $(h, 1, 2k+2)$.

With (iii), this implies that for every correct process that does not set *value* to 1 before, part (β) of condition of Line 15 or Line 23 is ensured in superround $2k+2$. Then every correct process sets *value* to 1 by round $2k+2$.

- **Case** $h \in L$**:** as p has accepted $(j_u, 1, 2u)$ for each u from 1 to k, it accepted $(h, 1, 2x)$ for some x with $1 \leq x \leq k$. By *unforgability* property, every process with identifier h proposed $(h, 1, 2x)$ in superround $2x$. With help of Lemma 6.4.5, every correct process sets *value* to 1 by superround $2x$.

□

Lemma 6.4.7. *If $r = 2t+2$ then every correct process sets value to 1 by superround $2t+2$.*

Proof. If $r = 2t+2$, p accepted $t+1$ messages $(j_1, 1, 2)$, . . . , $(j_u, 1, 2u)$, . . . , $(j_{t+1}, 1, 2(t+1))$ with distinct identifiers. As $l > t$, for at least one identifier j_h, $G(j_h)$ is a correct group. By *unforgeability* property, every process of $G(j_h)$ proposed $(j_h, 1, 2h)$. By Lemma 6.4.5, every correct process has set *value* to 1 by superround $2h$, where $h \leq t+1$. □

We are now ready to prove the agreement property of the Byzantine Agreement algorithm.

Proposition 6.4.6. *(Agreement) If two correct processes decide v and v' then $v = v'$.*

Proof. Assume that some correct process sets *value* to 1 in Line 19 or in Line 24, and decides 1. Let r be the first superround where some correct process sets *value* to 1.

Lemma 6.4.6 shows that if $r \leq 2t$ then every correct process sets its *value* variable to 1 by superround $r + 2$, then all correct processes decide 1.

Lemma 6.4.7 shows that if $r = 2t + 2$ then every correct process sets its *value* to 1 by superround $2t + 2$.

Otherwise, if no correct process sets *value* to 1 then *value* is 0 for all correct processes and they decide 0. □

Theorem 6.4.2. *Let D be a distribution and $A \subseteq \mathcal{F}_t(D)$ be an adversary, if for all $F \in A$ we have $c(F, D) > 2t$ then there is an algorithm solving Byzantine agreement for adversary A and distribution D.*

6.5 Results and consequences

6.5.1 Results

From Proposition 6.3.2 and Proposition 6.4.2, we deduce that Byzantine agreement is solvable for distribution D and at most t Byzantine failures if and only if $n > 3t$, $l > t$ and $c(F, D) > 2t$ for every failure pattern $F \in \mathcal{F}_t(D)$.

By Proposition 6.2.1, we have: $c(F_t, D) = \sum_{i=t+1}^{l} n_i + index(D)$. Hence, we may characterize the distributions for which Byzantine agreement is solvable as follows:

Theorem 6.5.1. *Byzantine agreement is solvable for distribution $D = \langle n_1, \ldots, n_l \rangle$ and at most t Byzantine failure if and only if $n > 3t$, $l > t$ and $\sum_{i=t+1}^{i=l} n_i + index(D) > 2t$.*

Proof. By Proposition 6.2.1:

$$c(F_t, D) = \sum_{i=t+1}^{i=l} n_i + index(D)$$

First, assume that Byzantine agreement is solvable for distribution $D = \langle n_1, \ldots, n_l \rangle$ and at most t Byzantine processes. From Proposition 6.3.2, we have $\forall F : c(F, D) > 2t$ and in

particular $c(F_t, D) > 2t$. Then:

$$c(F_t, D) = \sum_{i=t+1}^{i=l} n_i + index(D) > 2t$$

.

Now, assume that $\sum_{i=t+1}^{i=l} n_i + index(D) > 2t$. By Lemma 6.2.2, we have $c(F, D) \geq c(F_t, D)$. But:

$$c(F_t, D) = \sum_{i=t+1}^{i=l} n_i + index(D) > 2t$$

Thus, $\forall F : c(F, D) > 2t$. By Proposition 6.4.2, Byzantine agreement is solvable for distribution D.

□

6.5.2 Consequences

A natural question that arises is: for a given number of processes n, a given set of identifiers of size l, does it exist a distribution that tolerates t Byzantine failures? We directly know that to give a positive answer to this question we need $n > 3t$ and $l > t$. But when these conditions are satisfied, does it always exist a distribution such that $c(F, D) > 2t$ for every failure pattern $F \in \mathcal{F}_t(D)$? In this subsection we answer to this question and we show that there exists such distribution if and only if $l > \frac{(n-r)t}{n-t-min(t,r)}$ where $r = n \bmod l$. Furthermore when it exists, we exhibit the distribution that satisfies this condition.

From now we assume $n > 3t$ and $l > t$.

Let $r = n \bmod l$ and

$$D_{max} = \begin{cases} \langle \frac{n}{l}, \ldots, \frac{n}{l} \rangle, & \text{if } l \text{ divides } n. \\ \langle n_1 = \lceil \frac{n}{l} \rceil, \cdots, n_r = \lceil \frac{n}{l} \rceil, \lfloor \frac{n}{l} \rfloor, \ldots, n_l = \lfloor \frac{n}{l} \rfloor \rangle, & \text{otherwise.} \end{cases}$$

We first examine the properties of D_{max}:

Lemma 6.5.1. *If $n \geq 2l$ then $index(D_{max}) = t$ else $index(D_{max}) = min(r, t)$.*

Proof. We consider two cases:

- If $n \geq 2l$ then $\lceil n/l \rceil > \lfloor n/l \rfloor \geq 2$, by definition $index(D_{max}) = t$.
- If $2l > n \geq l$ then $\lfloor n/l \rfloor = 1$ and $\lceil n/l \rceil = 2$ and we have $D =< \underbrace{2, \ldots, 2}_{r}, 1 \ldots 1 >$. Then, $index(D_{max}) = min(r, t)$.

□

Lemma 6.5.2. $\sum_{i=t+1}^{i=l} n_i = (l-t)\lfloor n/l \rfloor + max(0, r-t)$

Proof. We consider two cases :

- If $r \leq t$ then $\sum_{i=t+1}^{i=l} n_i = (l-t)\lfloor n/l \rfloor$
- If $r > t$ then $\sum_{i=t+1}^{i=l} n_i = (l-r)\lfloor n/l \rfloor + (r-t)\lceil n/l \rceil = (l-t)\lfloor n/l \rfloor + r - t$.

□

D_{max} maximizes $\sum_{i=t+1}^{i=l} n_i + index(D)$:

Lemma 6.5.3. *For any distribution D, $D = \langle m_1, \ldots, m_l \rangle$, of n processes and l identifiers, if $n \geq 2l$ we have $\sum_{i=t+1}^{i=l} m_i + index(D) \leq \sum_{i=t+1}^{i=l} n_i + index(D_{max})$*

Proof. Let D be a distribution of n processes and l identifiers: $D = \langle m_1, \ldots, m_l \rangle$ with $m_1 \geq m_2 \geq \ldots \geq m_l > 0$.
By definition we have $index(D) \leq t$. As $n \geq 2l$, by Lemma 6.5.1 and definition of $index(D)$, we get

$$index(D) \leq index(D_{max}) \tag{6.5}$$

We now consider $\sum_{i=t+1}^{i=l} m_i$.

- If $m_{t+1} \leq \lfloor n/l \rfloor$ then $\sum_{i=t+1}^{i=l} m_i \leq (l-t)\lfloor n/l \rfloor$.
- If $m_{t+1} > \lfloor n/l \rfloor$ then $\sum_{i=1}^{i=t} m_i \geq t\lfloor n/l \rfloor - t$. Hence,

$$\begin{aligned}\sum_{i=t+1}^{i=l} m_i &\leq n - t\lfloor n/l \rfloor - t \\ &\leq (l-t)\lfloor n/l \rfloor + r - t\end{aligned}$$

In all cases,

$$\sum_{i=t+1}^{i=l} m_i \leq (l-t)\lfloor n/l \rfloor + r - t \tag{6.6}$$

Thus, from (6.5), Lemma 6.5.1 and (6.6) we have:

$$\sum_{i=t+1}^{i=l} m_i + index(D) \leq \sum_{i=t+1}^{i=l} n_i + index(D_{max}).$$

□

Theorem 6.5.2. *There is a distribution enabling to solve Byzantine agreement if and only if D_{max} enables us to solve Byzantine agreement.*

Proof. We need only to consider the necessary condition because the sufficient condition is trivial.

Assume that there is a distribution D of n processes in l identifiers enabling to solve Byzantine agreement for at most t Byzantine processes. $D = \langle\, m_1, \ldots, m_l \,\rangle$ with $m_1 \geq m_2 \geq \ldots \geq m_l > 0$. By Theorem 6.5.1 we have: $\sum_{i=t+1}^{i=l} m_i + index(D) > 2t$. We show D_{max} enables to solve Byzantine agreement for at most t Byzantine processes proving that $c(F_t, D_{max}) > 2t$. We consider two cases:

- $2l > n \geq l$. In this case, $index(D_{max}) = min(r, t)$ and $n = l + r$.
  - If $r \leq t$ then $c(F_t, D_{max}) = r + (l-t)\lfloor n/l \rfloor = r + l - t = n - t > 2t$.

– If $r > t$ then $c(F_t, D_{max}) = t+(l-t)\lfloor n/l \rfloor+(r-t) = t+(l-t)+r-t = n-t > 2t$.

Hence, we always have: $c(F_t, D_{max}) > 2t$. It follows that if $2l > n \geq l$ then D_{max} enables always to solve Byzantine agreement.

- $n \geq 2l$. In this case, by hypothesis and Lemma 6.5.3 :

$$2t < \sum_{i=t+1}^{i=l} m_i + index(D) \leq c(F_t, D_{max})$$

By Proposition 6.4.2, D_{max} enables to solve Byzantine agreement for at most t Byzantine processes.

□

We now characterize the value for n, l, t for which this distribution exists:

Theorem 6.5.3. *There exists a distribution enabling to solve Byzantine agreement for at most t Byzantine failure if and only if $l > \frac{(n-r)t}{n-t-min(t,r)}$ where $r = n \bmod l$.*

Proof. We consider two cases:

- If $2l > n \geq l$ then $n = l + r$. Notice that in this case $c(F_t, D_{max}) = n - t$.

  If $r > t$ then $\frac{(n-r)t}{n-t-min(t,r)} = \frac{lt}{n-2t}$

  If $r \leq t$ then $\frac{(n-r)t}{n-t-min(t,r)} = \frac{(n-r)t}{n-t-r} \leq \frac{lt}{n-2t}$.

  As $n > 3t$, we have $t/(n-2t) < 1$. Then we get $\frac{(n-r)t}{n-t-min(t,r)} < l$. We also have $c(F_t, D_{max}) > 2t$. Consequently, the theorem is trivially proved.

- If $n \geq 2l$ then $n = kl + r$ with $k \geq 2$.

If $r \leq t$, by Lemma 6.5.1 and Lemma 6.5.2, the condition $\sum_{i=t+1}^{i=l} n_i + index(D_{max}) > 2t$ is equivalent to $(l-t)\lfloor n/l \rfloor > t$

$$\begin{aligned}\frac{(n-r)t}{n-t-min(t,r)} < l &\Leftrightarrow klt/(kl-t) < l \\ &\Leftrightarrow (l-t)k > t \\ &\Leftrightarrow (l-t)\lfloor n/l \rfloor > t\end{aligned}$$

If $r > t$, by Lemma 6.5.1 and Lemma 6.5.2, the condition $\sum_{i=t+1}^{i=l} n_i + index(D_{max}) > 2t$ is equivalent to $(l-t)\lfloor n/l \rfloor + r > 2t$

$$\begin{aligned}\frac{(n-r)t}{n-t-min(t,r)} < l &\Leftrightarrow klt/(kl+r-2t) < l \\ &\Leftrightarrow kt < kl - 2t + r \\ &\Leftrightarrow (l-t)k + r > 2t \\ &\Leftrightarrow (l-t)\lfloor n/l \rfloor + r > 2t\end{aligned}$$

Thus in both cases:

$$\sum_{i=t+1}^{i=l} n_i + index(D_{max}) > 2t \Leftrightarrow \frac{(n-r)t}{n-t-min(t,r)} < l \tag{6.7}$$

With Theorem 6.5.2 and Theorem 6.5.1, (6.7) proves the theorem.

□

In particular as $\frac{(n-r)t}{n-t-min(t,r)} < l$ if $2l > n \geq l$, we deduce that if $2l > n \geq l$, then there exists a distribution D enabling to solves Byzantine agreement for at most t Byzantine failures. If $n \geq 2l$, then there exists a distribution D enabling to solves Byzantine agreement for at most t Byzantine failures if and only if $l > \frac{(n-r)t}{n-t-min(t,r)}$ where $r = n \bmod l$. In both cases, if it exists, D_{max} is such a distribution.

Recall from [22] that if the number of identifiers is known but the size of groups of processes with the same identifier is not known, a necessary and sufficient condition to solve Byzantine agreement is $l > 3t$ (assuming $n > 3t$ too). Hence the knowledge of the distribution enables us to solve Byzantine agreement in more cases. In the interesting case for which we have many processes but few identifiers we get for example that when $n = kl$ Byzantine agreement is solvable as soon as $l > t(1 + 1/k)$.

6.6 Conclusion

We have proven that the knowledge of distribution helps to solve the Byzantine agreement and enables us to get better bounds. Here with this knowledge, adding correct processes to the system may help. In other words, contrary to the results of [22] the number of authenticated servers needed does not depend only on the number of Byzantine processes but also on the number of correct processes.

The efficiency of the algorithms was not the main purpose of this paper, nevertheless it is worth noting that the complexity concerning the number of rounds is in $2(t + 1)$, It is the same complexity as for solution using authenticated broadcast [63] in the classical case with unique identifiers. An interesting open problem is to determine if this bound can be improved achieving solutions for Byzantine Consensus within $(t + 1)$ rounds.

A first extension of this work can be to consider particular adversaries. For example, from a practical point of view it could be reasonable to assume that the number of Byzantine processes by identifier depends on the number of processes with this identifier, it is then interesting to determine in this case necessary and sufficient conditions for solving Byzantine agreement.

Another extension is to consider partially synchronous models like [25, 31] and try to extend our results to these models.

Chapter 7

Homonymous model with forgeable identifiers

7.1 Introduction

In the previous chapters, we always assume that the identifiers are unforgeable. A way to obtain that is to use the group signatures [19] that define a signature scheme for homonyms in which the identifiers can be verified. However, in pratique, it may be difficult to keep such secret key : a Byzantine process in a group can voluntary divulge this secret key and some Byzantine processes in other groups may then usurp the identifier. Moreover some members of groups may simply be negligent and involuntary divulge this key enabling Byzantine processes to usurp the identifier too. Then we consider in this chapter an extension of the original model of homonyms in which at most k identifiers are forgeable and may be usurped by Byzantine processes.

We consider both homonym model with and without authentication. From a more practical point of view, it is easy to implement homonyms with help of digital signatures as with [35] in which at each identifier is associated a public key and processes with the same

identifiers share corresponding private keys. In this way we get a (strictly) stronger authentication mechanism as defined in [58]. With this authentication mechanism a process cannot retransmit falsely messages.

To determine the power of the homonymous model of homonyms with forgeable identifiers, we consider the problem of Byzantine Agreement. Recall from the chapter 4 that Byzantine Agreement is solvable in the homonymous model if and only if $l > 3t$, then considering forgeable identifiers as groups of processes with Byzantine processes, we get directly a solution with l forgeable identifiers if $l > 3k$ and we could suppose that we have a solution if and only if $l > 3k$. But surprisingly, we prove a better bound, we prove that there is solution for the Byzantine Agreement with k forgeable identifiers if and only if $l > 2t + k$. In fact, this result comes from the fact that if a Byzantine process forges the identifier of a group of processes containing correct processes, this group of processes has the same behavior as a group of processes containing together Byzantine and correct processes. It is proven in chapter 5 that such groups of processes are weaker adversaries than groups containing only Byzantine processes.

From a more practical point of view, it is easy to implement homonyms with help of digital signatures as with [35] in which at each identifier is associated a public key and processes with the same identifiers share corresponding private keys. In this way we get a (strictly) stronger authentication mechanism as defined in [58]. With this authentication mechanism a process cannot retransmit falsely messages. More precisely, if the identifier is unforgeable, then it is not possible for any process q to wrongly pretend that it received message m coming from a process with this identifier. It is well known that with this kind of authentication, the Byzantine Agreement problem can be solved if and only if $n > 2t$ in the classical case in which all processes have unique and different unforgeable identifiers, giving a $n > 2k$ bound with k forgeable identifiers. With homonyms and at most k forgeable identifiers, we prove that Byzantine Agreement is solvable if and only if

$l > t + k$.

Roadmap In Section 7.2 we prove impossibility result in model without authentication. In Section 7.3, we propose a specification of Authenticated Broadcast and give a corresponding algorithm. Section 7.4 contains the algorithm for Byzantine Agreement using Authenticated broadcast. Then, in Section 7.5 we study the authentication case. Finally in Section 7.6, we discuss some related work and perspectives.

7.2 Impossibility

Following the spirit of the impossibility of Byzantine Agreement in [34], we prove our impossibility results in weak homonym model.

Proposition 7.2.1. *Byzantine Agreement is unsolvable in* (n, l, k, t)*-homonymous model if* $l \leq 2t + k$.

Proof. It suffices to prove there is no synchronous algorithm for Byzantine Agreement when $l = 2t + k$. To derive a contradiction, suppose there is an algorithm $\mathcal{A}$ for Byzantine agreement with $l = 2t+k$. Let $\mathcal{A}_i(v)$ be the algorithm executed by a process with identifier i when it has input value v.
We divide the set of processes into 4 subsets:

- The set A is the set of processes with identifiers from 1 to t
- The set B is the set of processes with identifiers from $t + 1$ to $2t$
- The set C consists of all the processes with identifiers from $2t + 1$ to $3t$
- The set D consists of all the processes with identifiers from $3t + 1$ to $2t + k$

We now define three executions of the algorithm $\mathcal{A}$.

In execution α, processes of A, B and D are correct and have input 1, all the processes in C are Byzantine and the identifiers in D may be forged. By validity, all the correct processes decide 1. Processes in C run $\mathcal{A}_{Id(c)}(0)$ for c in C. One process in C runs $\mathcal{A}_{Id(d)}(0)$ for each $d \in D$.

In execution α', processes of B, C and D are correct and have input 0, all the processes in A are Byzantine and the identifiers in D may be forged. By validity, all the correct processes decide 0. Processes in A run $\mathcal{A}_{id(a)}(1)$ for each a in A. One process in A runs $\mathcal{A}_{Id(d)}(1)$ for each $d \in D$.

In execution β, processes of A and C are correct. Processes of A have input 1 and processes of C have input 0. Processes in D are correct and have input 1. All the processes in B are Byzantine and the identifiers in D may be forged. Processes in B send the same messages to processes in A and processes in D that in α. Processes in B send the same messages to processes in C in α'. One process in B runs $\mathcal{A}_{Id(d)}(0)$ for each $d \in D$ and executes this code as such a process receives from B the same message that in α'.

For the correct processes in A executions α and β are undistinguishable and they decide 1. For the correct processes in C executions α' and β are undistinguishable and they decide 0. This contradicts the agreement property of the Byzantine agreement. □

7.3 Authenticated Broadcast

Our algorithms for Byzantine Agreement use an adaptation of Authenticated Broadcast as introduced by Srikanth and Toueg [63] in the classical case where each process has a different identifier ($n = l$).

More precisely, our authenticated broadcast is defined by two primitives: $broadcast(i, v, r)$ and $accept(i, v, r)$ where i is the identifier of some group that proposes value v and r is the superround number where the message has been proposed. We assume that a

correct process broadcasts at most one message in a superround. The specification of authenticated broadcast primitive is same of the one presented in Section 6.4:

1. *Correctness:* If all the processes in a correct group i perform $broadcast(i, v, r)$ in superround r then every correct process performs $accept(i, v, r)$ during superround r.

2. *Relay:* If a correct process performs $accept(i, v, r)$ during superround $r' \geq r$ then every correct process performs $accept(i, v, r)$ by superround $r' + 1$.

3. *Unforgeability:* If some correct process performs $accepts(i, v, r)$ in superround $r' \geq r$ then all correct processes in group i must $broadcast(i, v, r)$ in superround r.

Code for process p with identifier $i \in \{1, ..., l\}$

Variable:

1 $\mathcal{M} = \emptyset$;

Main code:

2 ROUND R

3 **if** $R = 2r$ **then if** $broadcast(i, v, r)$ **to perform**

4 **then send** $(\mathcal{M} \cup (init, i, v, r), R)$ to all

5 **else send** $(\mathcal{M} \cup (noinit, i, \perp, r), R)$ to all

6 **else send** $(\mathcal{M}, R)$ to all;

Reception of the messages of round R

7 **For all** $h \in \{1, ..., l\}$

8 **if** $(R = 2r)$ **then**

Let $\mathcal{M}[h]$ be the set of messages $(init, h, *, r)$ or $(noinit, h, *, r)$ received from processes in group h

9 **if** $M[h] = \{(init, h, v, r)\}$

10 **then** $\mathcal{M} = \mathcal{M} \cup (echo, h, v, r)$

11 **For all** $r \in \{1, ..., R/2\}$

12 **For all** $m \in$ possible messages

13 **if** $(echo, h, v, r)$ received from at least $l - 2t$ distinct groups

14 **then** $\mathcal{M} = \mathcal{M} \cup (echo, h, v, r)$

15 **if** $(echo, h, v, r)$ received from at least $l - t$ distinct groups

16 **then** $accept(h, v, r)$

Figure 7.1: Authenticated Broadcast algorithm in the (n, l, k, t)-homonym model.

The algorithm is described in Figure 7.1. A superround r is composed of the two rounds $2r$ and $2r + 1$.

To propose a value v in supperround r process p with identifier i sends message $(init, i, v, r)$ to all processes (including itself) (line 4). A process receiving such a message from some processes with identifier i becomes "witness" for (i, v, r) and sends a message of type *echo* to all processes (line 10). Any process having $l - 2t$ witnesses for (i, v, r) becomes itself witness (if $l - 2t > k$ at least one process in a correct group has sent this message) (line 13). When a process receives more than $(l - t)$ witnesses, it accepts (i, v, r) (at least $t + 1$ processes from correct groups have sent this message) (lines 15 to 16). In this way we ensure *correctness* and *relay* properties

To ensure the unforgeability property, a correct process that has no message to broadcast in a superround broadcast a *noinit* message in the corresponding even round. In this way, if some correct process with identifier i has no message to broadcast in superround r, every correct process gets $M[i]$ (line 9) different from one message *init* and will not become witness of any message $(i, *, r)$.

By a standard proof, we get :

Proposition 7.3.1. *If $l > 2t + k$, the algorithm Figure 7.1 implements authenticated broadcast in (n, l, k, t)-homonym model.*

Proof. We prove the properties of authenticated broadcast:

Correctness: If all processes in correct group i perform $broadcast(i, v, r)$ then they send $(init, i, v, r)$ in round $R = 2r$ (Line 4). All correct processes receive $(init, i, v, r)$ and at least all correct processes send $(echo, i, v, r)$ message in the second round of superround r. Since there are at most t Byzantine groups, there are at least $l - t$ correct groups. Thus, a correct process receives at least $l - t$ messages $(echo, i, v, r)$ from these groups at end of the second round of superround r and thus it $accept(i, v, r)$.

Unforgeability: We suppose that some correct process performs $accepts(i, m, r)$ in the round $2r' + 1$. Thus, it must receive $(echo, i, v, r)$ from at least $l - t$ distinct groups. Thus, among these groups, there is at least $l - t - k > 1$ group whose identifer isn't forgeable (because there are at most k forgeable identifiers of groups). Thus, some correct process of this group sent $(echo, i, v, r)$.

Suppose that p is first correct process that sends $(echo, i, v, r)$. There are two cases so that p sends $(echo, i, v, r)$:

- The set $M[i]$ of p contains only $(init, i, v, r)$. Thus, every correct process in the group i perform $broadcast(i, v, r)$.
- It received $(echo, i, v, r)$ from at least $l - 2t$ distinct groups. Since $l - 2t > k$, there is at least group that contains one correct process that sent $(echo, i, v, r)$ in this round and thus, it sent before p. It is contradiction. Thus, the case 2 is impossible.

Thus, if some correct process $accepts(i, v, r)$ in superround r' then all correct processes with identifier i must $broadcast(i, v, r)$ in superround r.

Relay: Suppose some correct process p performs $accept(i, v, r)$ during superround r'. Thus, p has received the message $(echo, i, v, r)$ from $l - t$ groups with different identifiers. At least $l - 2t$ of those groups contain correct process. Each of those $l - 2t$ processes continue to send $(echo, i, v, r)$ in every round after superround r. Thus, in superround $r' \geq r$, every correct process sends $(echo, i, v, r)$ and all of these messages are delivered, so every correct process receives at least $(echo, i, v, r)$ from at least $l - t$ distinct groups and performs $accept(i, v, r)$. □

7.4 Byzantine Agreement Algorithm

The algorithm is very similar to the one presented in Figure 6.2 in Section 6.4.2 of Chapter 6 except the two differences as follows:

1. the algorithm runs in $2k + 2$ superrounds instead of $2t + 2$ superrounds as the algorithm of Figure 6.2. The condition assure that among $k + 1$ identifiers, there is at least one identifier that all processes with that identifier are correct.

2. the conditions at Lines 7, 15 and 18 in Figure 6.2 replaced by $|A| \geq t+1$, $|A| \geq t+1$ and $|A| \geq t + 1$. Recall that $A = \{h | h \in \mathcal{L}$ such that $(h, 1, 1)$ is accepted$\}$ and the condition at these lines assures the validity property of agreement. The replacement is needed because the distribution of identifiers is unavailable as in Chapter 6 but there is the majority of correct groups.

Thus, we get:

Proposition 7.4.1. *Byzantine agreement is solvable if $l > 2t + k$.*

Finally, combining Proposition 7.2.1 and 7.4.1, we get:

Theorem 7.4.1. *Without authentication, Byzantine agreement is solvable in (n, l, k, t)-homonymous model if and only if $l > 2t + k$.*

7.5 Homonymous model with authentication

The classical model of Byzantine failures with authentication is introduced in [58] adapted to homonyms and forgeable identifiers. This model may be implemented with help of authentication tools such as Public Key Infrastructure and Digital Signatures Schemes. Messages m are authenticated by the identifier of the sender process. At each identifier is associated a secret key K and a signature scheme $Sign(K, m)$ that allows m to be signed with the key associated to the identifier. Each process can verify if a message carries the signature of a given identifier. We assume that the signatures of at most k identifiers can be forged. With this scheme of authentication, if id is not forgeable then it is not

possible for any process q to wrongly pretend that it received some message m coming from identifier id. Implementation of homonyms with authentication and some forgeable identifier is rather natural: members of groups share a secret key and every message from a group are sent to all members of some groups and the source group of the message may be verified by a signature scheme of the source group.

For the authentication case, we improve our bound: $l > t + k$ is necessary and sufficient to achieve Byzantine Agreement. (Recall that in the classical model with unique identifiers, the bound is $n > 2t$.)

The proofs of the lower bound is essentially the same as for the case without authentication in Section 7.2.

The Byzantine agreement algorithm in the case without authentication in Section 7.4 directly works with $l > t + k$ if we have an Authenticated Broadcast. It remains to get an Authenticated Broadcast with $l > k + t$.

In [63], in the classical case with unique identifiers, Authenticated Broadcast can be obtained simply with authentication: it suffices to verify the signatures: A process that receives a message (p, m, r) accepts it if it can verify p's signature (and then forwards this message). But if we apply this simple mechanism in our model, we do not get the unforgeability property. In a forgeable group with identifier i containing some correct processes, it is possible that some correct accepts (i, m, r). Indeed, this message has been sent by a Byzantine process that has forged the identifier i.

To implement Authenticated Broadcast, we use the signatures and the mechanism of witnesses as in our previous algorithm. The implementation is based on the one presented in Section 7.3 and some easy changes.

At some point, in algorithm 7.1 when a process receives some messages in round R, it will send *echo* in the next rounds. The process will forward all messages that produced this *echo*. In this way it gives a proof that it has the right to send *echo*. We get an

Authenticated Broadcast algorithm from algorithm Figure 7.1 by: (1) removing from the received message all messages with a bad signature, (2) forwarding the proof of each new *echo* message, (3) receipt of messages $(echo, h, m, r)$ at lines 13 and 15 is replaced by the receipt of message $(echo, h, m, r)$ and the proof of $(echo, h, m, r)$. This assures the validity of each received message $(echo, h, m, r)$.

We have:

Theorem 7.5.1. *With authentication, Byzantine Agreement is solvable in* (n, l, k, t)*-homonymous model if and only if* $l > t + k$*,*

7.6 Related works and perspectives

When processes share identifiers and some of these processes may be Byzantine, it is rather natural that some of these identifiers may be forged. Hence this work is a natural extension of [22] to forgeable identifiers.

At least for the authentication case, groups signature as introduced first in [20] are close to our model. Groups signature enables to sign messages on behalf of a group and clearly can be used to implement the model of homonyms. Note that group signatures generally ensures other properties than the one we consider here. Groups signatures may be a valuable way to implement models with homonyms.

In other works [9, 36, 40], a mixed adversary model is considered in the classical ($l = n$): the adversary can corrupt processes actively (corresponding to Byzantine process) and can forge the signature of some processes.

In some way, we combine here the idea of group signatures and forgeable signatures but contrary to group signatures the goal is not to develop protocol ensuring strong properties like anonymity or unforgeability but to develop algorithms (like agreement) in presence of groups of processes with Byzantine processes and forgeable identifiers.

Here we proved that with forgeable identifiers and homonyms Byzantine Agreement can be solved in a "reasonable" way (and without any assumption about cryptographic system). Interestingly, the solvability of Byzantine Agreement depends only on the number of identifiers and the number of forgeable identifiers. Hence adding correct processes does not help to solve Byzantine Agreement.

A natural extension of this work could be to consider partially synchronous models.

As Byzantine Agreement is the basis for replication systems in the classical models in which each process has its own identity, a natural question is to know if it is still the case and envisage to develop algorithms for more difficult problems.

Chapter 8

Conclusion

The homonymous model considered in this thesis is new. It generalizes both the classical (non-anonymous) and the anonymous model. In our model, several processes may share the same identifier. As an example of the homonymous model, a network of computers communicating by sending and receiving messages. Each computer has a MAC address. Computers are regrouped into groups such that each group has a multicast address as identifier of group. A computer of a group might hide its MAC address and use only its multicast address of group to communicate. A way to avoid that the identifier of group is forgeable, is to use the group signatures [19] that define a signature scheme for homonyms in which the identifiers can be verified.

In homonymous model, we completely characterized the solvability of consensus, precisely quantifying the impact of the adversary, with some surprising results. In the synchronous case, we given the complete picture, from crash and send omission failure model to Byzantine failure model, on necessary and sufficient conditions on the number of identifiers for solving consensus in a system of n processes using l identifiers and tolerating t faulty processes. Ours results show that for the benign failures, for example crash and omission

failures, that fact that processes are numerate is sufficient to solve the uniform consensus. Otherwise, the identity is really necessary to solve the problem for Byzantine failures. $l > 3t$ is necessary and sufficient condition to reach the Byzantine agreement. Interestingly, our generic algorithm does not depend on the number of processes n. Hence, in this case, the global parameter n is not necessary and adding more correct processes does not affect the algorithm. However, in partially synchronous case, the global parameter n is necessary and adding more correct processes may require more identifiers. It is different from the result in classical systems with unique identifiers. That difference originates from homonyms. We also showed two ways to notably reduce the number of identifiers for Byzantine agreement, either removing the ability for a Byzantine process to send multiple messages to the same recipient or increasing the knowledge of the system for each process assuming each process knows the distribution of identifiers. Finally, we considered the agreement problem in a rather natural case where Byzantine processes can forge identifiers of correct processes.

Our algorithms for uniform consensus problem in benign failures model are new. These algorithms are inspired from [59]. For the Byzantine agreement, our algorithms use versions of authenticated broadcast in [63]. However, the generalizations of the authenticated broadcast to our model is not straightforward. Recall that differently from classical systems with unique identifiers where if a Byzantine process sends multiple messages to a single recipient in a round then receiver process can easily detect that the sender of these messages is faulty process and algorithms could simply discard such messages, in homonymous model there is clear advantage. This thesis also contributes to many impossibility results using valency argument, scenario argument and partitioning argument.

Many challenging questions remain open. The paper [23] solves the uniform consensus in anonymous systems in partially synchronous case. The uniform consensus in anonymous systems in asynchronous case with the failure detector is considered in [13]. Also, the

uniform consensus in asynchronous case with the failure detector is considered in [4] but in homonymous systems as our model. All these papers consider the crash failures model. None of these considers the general omission failures model. We conjecture that in this failures model, the lower bound for uniform consensus is $l > (n + 2t)/2$ if processes are innumerate, however if processes are numerate, the lower bound is $n > 2t$ even if processes are anonymous. A more difficult open problem is solve k set agreement in these models. Concerning the complexity of the consensus algorithms, we know that $t + 1$ rounds is the lower bound in classical model with unique identifiers. For benign failures model, we showed that this lower bound remains true in our homonymous model giving algorithms in $t+1$ rounds in Chapter 3. However, for Byzantine failures model, our algorithms required more rounds. What is the tight lower bound on number of rounds to reach to agreement in presence of Byzantine failures? We conjecture that the existence of homonyms and Byzantine processes affect the complexity of solving this problem. Thus, it needs at least $t + 2$ rounds to solve Byzantine agreement.

Finally, how do homonyms affect the solvability of problems other than the consensus, for example the leader election problem, the naming and the counting problem? In the annexe of this thesis, we consider the leader election problem for a ring consisting of n processes $n \geq 2$ where each process is assigned an identifier in the set of l identifiers and processes are not faulty. We show that with a few identifiers, the leader election problem is possible. More precisely, the leader election problem is solvable if and only if l is greater than the largest proper divisor of n. In particular, if n is prime then only with two identifiers, we have a deterministic leader election algorithm. That result is positif in comparison to the classical result that deterministic algorithm for leader election is impossible if processes are anonymous. Therefore, our homonymous model may be a good way to break symmetry. Finding an efficient algorithm or solving leader election problem in homonymous model in other network topology remain open.

Appendix: Leader election in the homonymous model

8.1 Introduction

Leader election is the problem of electing a unique leader in a distributed network. Leader election is a fundamental problem in distributed computing and has numerous applications. For example, it is an important tool for breaking symmetry in a distributed system. Moreover, by choosing a process as the leader, it is possible to execute centralized algorithms in a decentralized environment. Leader election can also be used to recover from token loss for token-based algorithms, by making the leader responsible for generating a new token when the current one is lost.

Without the ability to break symmetry, deterministic leader election is impossible [3]. Therefore, deterministic leader election is impossible if processes are anonymous. For example, consider a synchronous system of anonymous processes. If all processes start in the same state with the same environment, they will always remain in the same state as one another. Similarly, in an asynchronous shared memory system of anonymous processors with atomic reads and writes, where all registers have the same initial contents, or in an asynchronous message passing system of anonymous processes, where all communication links contain the same nonempty sequence of messages, many schedules, for example, a

round robin schedule, will maintain symmetry among all the processes.

In share memory systems, although in the presence of a central demon, which acts as a scheduler (a way to break the symmetry), Dijkstra [24] observed that a deterministic leader election algorithm cannot exist if the ring size is a composite number. Several papers [16, 41] present leader election algorithms for anonymous rings of prime size.

Randomization is a well known technique to break symmetry and randomized algorithms for both problems have been considered on a variety of network topology (e.g. ring, tree, complete graph) in [1, 68, 45]. However, the randomization problem is beyond the scope of this thesis.

8.2 Model

We consider the leader election problem in homonymous model for a ring consisting of n processes $n \geq 2$. Each process is assigned an identifier in the set $\mathcal{L} = \{1, 2, ..., l\}$. Homonymous processes start in the same state and run the same code. Processes know the value of n and l but do not know the identifier distribution in the ring.

The ring is unidirectional, that is, each process can only directly get information from its left neighbour. The distance from process p to process q is denoted by $d(p, q)$ that is measured starting from p and moving to the right until q is reached.

Processes communicate by sending and receiving messages. Computation proceeds in rounds. Each round consists of the three steps sending, receiving, local computation.

8.3 Our result

We prove the following result:

Theorem 8.3.1. *Leader election is solvable if and only if l is greater than the largest*

proper divisor of n.

Using the Proposition 8.3.1 and Proposition 8.3.2

Proposition 8.3.1. *Leader election is impossible if l is less than or equal to the largest proper divisor of n.*

Proof. If $l = 1$ then by the classical result that leader election is unsolvable in an anonymous ring and in synchronous case, the proposition holds. Now, consider that $l \geq 2$.

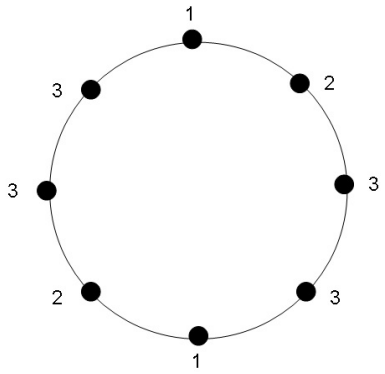

Figure 8.1: An example: $n = 8$, $l = 3$.

Let k be the largest proper divisor of n. Let $n = kr$ where r is positive integer and $r \geq 2$. We call a segment be a sequence of k processes positioned in the clockwise direction as follows: the process with identifier 1 is positioned at the first position of the segment, the process with identifier 2 at the second position of the segment,..., process having identifier $(l-1)$ at the $(l-1)$th position of the segment. $k - l + 1$ rest processes having identifier l are positioned from the lth to the kth. We can construct the ring of size n by r continuous segments. They are symmetrically distributed in the ring. For example, $n = 8, l = 3$. The distribution is figured in 8.1.

It is straightforward to verify, by induction on the number of rounds, that in every round r all processes with the identical positions of r segments are in identical states. In fact, since all processes with the identical positions of segments are the same identifier, they start in the same state. By induction, all processes with the identical positions send the same

message(s), receive the same message(s), do the same local computation, and therefore end up in the same state.

If one process in some segment decides to become a leader, then every process with the identical positions of $r-1 \geq 1$ other segments does so as well, contradicting the definition of leader election problem. Thus, the proposition holds.

□

Now we prove Proposition 8.3.2.

Definition 8.3.1. *Assume that Q is some subset of n processes and $q \in Q$. Let q_1 be the process that belongs to Q but is other than q and that is nearest to q in the left direction. We denote:*

$$d_Q(q) = \begin{cases} d(q_l, q), & \text{if } |Q| \geq 2 \\ n, & \text{if } |Q| = 1. \end{cases}$$

Lemma 8.3.1. *For every Q that is a subset of set of n processes, $\sum_{p \in Q} d_Q(p) = n$.*

Proof. All processes of Q are numbered $q_1, q_2, ...$ by some way in the clockwise direction. Thus, $d(q_i)$ is equal to the number of processes between q_i and q_{i-1} +1. Thus, $\sum_{q \in Q} d(q) = n - |Q| + |Q| = n$. □

Lemma 8.3.2. *Let $M = \{i \in \mathcal{L} | \exists p \in G(i)$ such that $d_{G(i)}(p) \geq l\}$ where $G(i)$ is group of all processes with identifier i. Thus, $M \neq \emptyset$.*

Proof. Suppose that $M = \emptyset$. Thus for every identifier i and every $p \in G(i)$, we have $d(p) < l$. By Lemma 8.3.1, for every i: $\sum_{p \in G(i)} d(p) = n < l|G(i)|$ where $|G(i)|$ is the number of processes in group $G(i)$. Thus, $ln = l(\sum_{i \in \mathcal{L}} |G(i)|) = \sum_{i \in \mathcal{L}} l|G(i)| > ln$. Contradiction. Thus, $M \neq \emptyset$ $\sum_{p \in G(i)} d(p) = n$. □

Definition 8.3.2. *An identifier m is said to be the leader identifier, if m is the greatest identifier of set M.*

Code for process p

1 Step 1: choose the leader identifier m
2 Step 2: **for each** $q \in G(m)$, $s(q) = active$
3 Let $Q = G(m)$ and $d_m = Max\{d_Q(q) | q \in Q\}$
4 **While**$(d_m < n)$
5 **for each** $q \in Q$
6 **if** $d_Q(q) < d_m$ **then** $s(q) = passive$
7 $Q = \{q \in G(m) : s(q) = active\}$
8 $d_m = Max\{d_Q(q) | q \in Q\}$
9 Step 3: choose unique process q in Q as the leader

Figure 8.2: deterministic function for choose the leader

Proposition 8.3.2. *Leader election is solvable if l is greater than the largest divisor of n that is other than n.*

Proof. We present here a simple algorithm as follows: at the first round, each process send its identifier to its neighbour. In next rounds, each process sends to its neighbour all messages that it received. By that way, after n rounds, each process can know the identifiers distribution in the ring and identically choose a leader using a deterministic function in Figure 8.2.

In fact, by the definition of leader identifier, all processes choose the same leader identifier in step 1. All processes come to step 2. If $|G(m)| = 1$ then all processes pass over the While loop and come to the step 3. Now we consider that $G(m) \geq 2$. We have $d_m < n$ and all processes enter in the loop. By Lemma 8.3.3, the number of processes of set Q decreases strictly after each iteration. When Q only contain one process then d_m of Q is equal to n and each process comes to step 3 and chooses the unique process in Q as the leader. □

Lemma 8.3.3. *Assume that if l is greater than the largest proper divisor of n. Thus, the number of processes of set Q decreases strictly after each update at Line 7.*

Proof. By update of state of processes of set Q at Line 6, after each iteration, the number

of processes of set Q decreases (Line 7). Thus, if at some iteration r, the set Q does not change at Line 7 then by Line 6, no process changes its state from active to passive. Thus, for every $q \in Q$, we have $d_Q(q) = d_m$. By Lemma 8.3.1, we have $\sum_{q \in Q} d(q) = n$. Thus, at iteration r, $d_m|Q| = n$. On the other hand, by definition of the leader identifier m and Lemma 8.3.2, at Line 2, we have $d_m \geq l$. By Lemma 8.3.4, d_m increases after each update. Thus, at the and of iteration r, we have $d_m \geq l$, contradicting the hypothesis that l is greater than the largest proper divisor of n. Thus, the number of processes of set Q decreases strictly. □

Lemma 8.3.4. *d_m increases after each update at Line 8.*

Proof. By update of state of processes of set Q at Line 6, after update of Q at Line 7, the number of processes of set Q decreases. Assume that at some iteration r, before the update of Q at Line 7, $Q = Q_1$ and after update, $Q = Q_2$ such that $Q_2 \subseteq Q_1$. If $Q_2 = Q_1$ then clearly, the value of d_m does not change at iteration r. We consider the case where $Q_2 \subset Q_1$. Let $Q_3 = Q_1 - Q_2$ where $Q_3 \neq \emptyset$. All processes of set Q_3 change their states from *active* to *passive* at Line 6. Thus, at line 6, for each $q \in Q_3 : d_{Q_3}(q) < d_m$. Since before update at line 7, $d_m = Max\{d_{Q_1}(q)|q \in Q_1\}$, there is some process q_0 such that $d_{Q_1}(q_0) = d_m$ and by (*), $q_0 \notin Q_3$. It implies that $q_0 \in Q_2$. But clearly, $d_{Q_2}(q_0) \geq d_{Q_1}(q_0)$. After update at line 8, $d_m = Max\{d_{Q_2}(q)|q \in Q_2\} \geq d_{Q_2}(q_0)$. Thus, Lemma holds.

□

List of Tables

List of Figures

Bibliography

[1] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, and D. Kirkpatrick. The bit complexity of randomized leader election on a ring. *SIAM J. Comput.*, 18(1):12–29, February 1989.

[2] Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. *Inf. Process. Lett.*, 71(3-4):155–158, 1999.

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

[4] Sergio Arévalo, Antonio Fernández Anta, Damien Imbs, Ernesto Jiménez, and Michel Raynal. Failure detectors in homonymous distributed systems (with an application to consensus). In *ICDCS*, pages 275–284. IEEE, 2012.

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

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

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

[8] Hagit Attiya and Jennifer Welch. *Distributed Computing: fundamentals, simulations and advanced topics, 2nd edition*. Wiley, 2004.

[9] Piyush Bansal, Prasant Gopal, Anuj Gupta, Kannan Srinathan, and Pranav K. Vasishta. Byzantine agreement using partial authentication. In *DISC*, volume 6950 of *LNCS*, pages 389–403, 2011.

[10] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project ‚Äú anonymity and unobservability in the internet ‚Äù. In *Proceedings of the tenth conference on Computers, freedom and privacy: challenging the assumptions*, CFP '00, pages 57–65, New York, NY, USA, 2000. ACM.

[11] Paolo Boldi and Sebastiano Vigna. Computing anonymously with arbitrary knowledge. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, PODC '99, pages 181–188, New York, NY, USA, 1999. ACM.

[12] Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In *DISC*, volume 2180 of *LNCS*, pages 33–47. Springer, 2001.

[13] François Bonnet and Michel Raynal. The price of anonymity: Optimal consensus despite asynchrony, crash, and anonymity. *TAAS*, 6(4):23, 2011.

[14] Stephen C. Bono, Christopher A. Soghoian, and Fabian Monrose. Mantis: A lightweight, server-anonymity preserving, searchable p2p network. Technical report, INFORMATION SECURITY INSTITUTE, JOHNS HOPKINS UNIVERSITY, 2004.

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

[16] J. E. Burns and Jan K. Pachl. Uniform self-stabilizing rings. *ACM Trans. Program. Lang. Syst.*, 11(2):330–344, April 1989.

[17] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[18] Bernadette Charron-Bost and André Schiper. Uniform consensus is harder than consensus. *J. Algorithms*, 51(1):15–37, 2004.

[19] David Chaum and Eugène Van Heyst. Group signatures. In *Proceedings of the 10th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'91, pages 257–265, Berlin, Heidelberg, 1991. Springer-Verlag.

[20] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, volume 547 of *LNCS*, pages 257–265, 1991.

[21] Carole Delporte-Gallet and Hugues Fauconnier. Two consensus algorithms with atomic registers and failure detector omega. In Vijay K. Garg, Roger Wattenhofer, and Kishore Kothapalli, editors, *ICDCN*, volume 5408 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2009.

[22] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec, Eric Ruppert, and Hung Tran-The. Byzantine agreement with homonyms. In *PODC*, pages 21–30. ACM, 2011.

[23] Carole Delporte-Gallet, Hugues Fauconnier, and Andreas Tielmann. Fault-tolerant consensus in unknown and anonymous networks. In *Proc. 29th IEEE International*

Conference on Distributed Computing Systems, pages 368–375. IEEE Computer Society, 2009.

[24] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, November 1974.

[25] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[26] Danny Dolev, Michael J. Fischer, Robert J. Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982.

[27] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, October 1990.

[28] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, STOC '82, pages 401–407, New York, NY, USA, 1982. ACM.

[29] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, November 1983.

[30] John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *IPTPS*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2002.

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

[32] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.

[33] Michael J. Fischer, Nancy A. Lynch, and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1981.

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

[35] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[36] S. Dov Gordon, Jonathan Katz, Ranjit Kumaresan, and Arkady Yerukhimovich. Authenticated broadcast with a partially compromised public-key infrastructure. In *SSS*, volume 6366 of *LNCS*, pages 144–158, 2010.

[37] R. Guerraoui. Revisiting the relation between non-blocking atommic commitment and consensus.

[38] Rachid Guerraoui, Petr Kouznetsov, and Bastian Pochon. A note on set agreement with omission failures. *Electr. Notes Theor. Comput. Sci.*, 81:48–58, 2003.

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

[40] Anuj Gupta, Prasant Gopal, Piyush Bansal, and Kannan Srinathan. Authenticated byzantine generals in dual failure model. In *ICDCN*, volume 5935 of *LNCS*, pages 79–91, 2010.

[41] Shing-Tsaan Huang. Leader election in uniform rings. *ACM Trans. Program. Lang. Syst.*, 15(3):563–573, July 1993.

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

[43] Valerie King and Jared Saia. Scalable byzantine computation. *SIGACT News*, 41(3):89–104, September 2010.

[44] Garg Vijay Kumar. *Elements of Distributed Computing*. Wiley-Interscience, 2002.

[45] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. In *ICDCN*, pages 348–362, 2013.

[46] Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280, April 1984.

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

[48] Qiao Lian, Zheng Zhang, Mao Yang, Ben Y. Zhao, Yafei Dai, and Xiaoming Li. An empirical study of collusion behavior in the maze p2p file-sharing system. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, ICDCS '07, pages 56–, Washington, DC, USA, 2007. IEEE Computer Society.

[49] M. C. Loui and H. H. Abu-Amara. *Memory requirements for agreement among unreliable asynchronous processes*, volume 4. JAI press, 1987.

[50] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[51] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Brief announcement: Naming and counting in anonymous unknown dynamic networks. In Marcos K. Aguilera, editor, *DISC*, volume 7611 of *Lecture Notes in Computer Science*, pages 437–438. Springer, 2012.

[52] C. Mohan, R. Strong, and S. Finkelstein. Methods for distributed transaction commit and recovery using Byzantine agreement within clusters of processors. pages 89–103.

[53] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, September 1990.

[54] James Newsome, Elaine Shi, Dawn Xiaodong Song, and Adrian Perrig. The sybil attack in sensor networks: analysis & defenses. In Kannan Ramchandran, Janos Sztipanovits, Jennifer C. Hou, and Thrasyvoulos N. Pappas, editors, *IPSN*, pages 259–268. ACM, 2004.

[55] Michael Okun. Agreement among unacquainted byzantine generals. In Pierre Fraigniaud, editor, *DISC*, volume 3724 of *Lecture Notes in Computer Science*, pages 499–500. Springer, 2005.

[56] Philippe Raipin Parvédy and Michel Raynal. Uniform agreement despite process omission failures. In *IPDPS*, page 212. IEEE Computer Society, 2003.

[57] Philippe Raipin Parvédy and Michel Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In Phillip B. Gibbons and Micah Adler, editors, *SPAA*, pages 302–310. ACM, 2004.

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

[59] Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. Software Eng.*, 12(3):477–482, 1986.

[60] Michel Raynal. Consensus in synchronous systems: A concise guided tour. In *PRDC*, pages 221–228. IEEE Computer Society, 2002.

[61] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, November 1998.

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

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

[64] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. Exploiting kad: possible uses and misuses. *SIGCOMM Comput. Commun. Rev.*, 37(5):65–70, October 2007.

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

[66] Sam Toueg, Kenneth J. Perry, and T. K. Srikanth. Fast distributed agreement. *SIAM J. Comput.*, 16(3):445–457, 1987.

[67] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35, 2005.

[68] Zhenyu Xu and Pradip K. Srimani. Self-stabilizing anonymous leader election in a tree. In *IPDPS*. IEEE Computer Society, 2005.

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